# Spring Boot

Mohamed Anouar DAHDEH

medanouar.dahdeh@gmail.com

# AVANT DE COMMENCER !

## Objectifs:

Ce cours vous apporte les connaissances et les compétences nécessaires pour appréhender :

- Les fonctionnalités du **framework SPRING** à l'aide son composant **Spring Boot**
- Son intégration dans l'IDE Spring Tool Suite
- Ses apports pour les différentes couches applicatives(web, service, DAO,..), la sécurité et pour la mise en production d'application.

## Les prérequis:

- Des connaissances de base en Java 8 sont requises.
- La connaissance de HTML, JSON, SQL et des bases de données est très utile.

# AVANT DE COMMENCER !

## Modèle pédagogique:

Formation présentielle (Cours et Travaux pratiques)

## Volume Horaire:

21H cours / 10h30 TP

## Mode d'évaluation :

| Epreuves écrites | | Epreuve pratique (Projet) |
|---|---|---|
| DS | Examen | CC |

# PLAN

- **Chapitre 1 : Introduction à Spring Boot**
- **Chapitre 2 : Application Web avec Thymeleaf**
- **Chapitre 3 : API REST**
- **Chapitre 4 : Persistance des données**
- **Chapitre 5 : Spring Security (TP)**
- **Chapitre 6 : Mise en production (TP)**


# Introduction à Spring Boot

**Introduction à Spring Boot**

- Introduction
- Framework Spring
- Spring Boot
- Construction d'une application Web Spring Boot
- Les starters
- Les annotations
- Logging
- Spring IOC container et l'injection de dépendances

**Travaux Pratiques :**

- TP1 - Création de projet Spring Boot avec STS, Spring Intializr et CLI
- TP2 - Logging
- TP3- IOC container et l'injection de dépendances

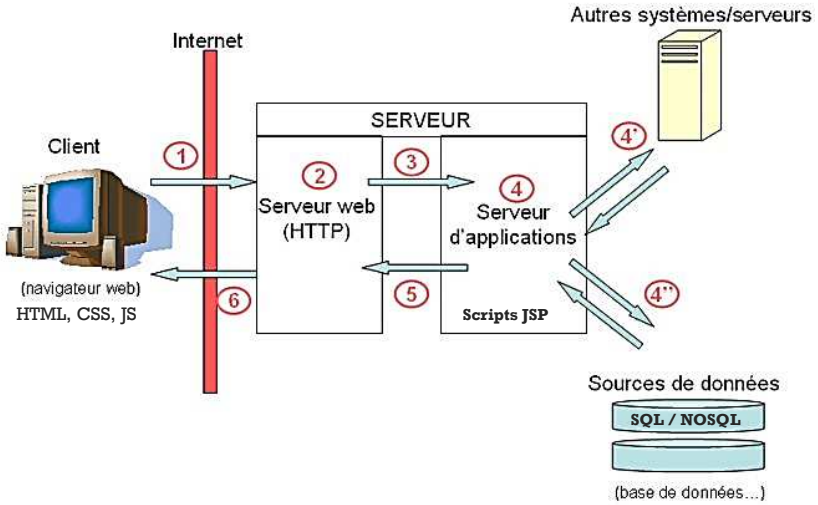
# INTRODUCTION (1/3)

- Historiquement, lorsqu'on souhaitait faire du web en Java, il était nécessaire de concevoir une application capable de servir à la fois des pages HTML statiques et des pages web dynamiques, ainsi que d'encapsuler la logique métier de l'application.
- Dans le mouvement des débuts de Java, de nombreux **Frameworks** web ont vu le jour, tels que **JSP** et **JSF**, portés par des **serveurs web**, et des **serveurs d'applications** JEE (Java Entreprise Edition) tels que **WildFly** (Anciennement JBOSS), **WebSphere**, **WebLogic**…

- **Framework (cadre de travail en français) :** une boite à outil pour le développeur.
- **Serveur Web :** Une machine ou un logiciel sert le contenu au web en utilisant le protocole HTTP.
- **Serveur d'applications :** héberge et expose la logique et les processus métier aux applications client

# INTRODUCTION (2/3)

- Exemple de fonctionnement d'une Application web JSP




1. Le client émet une **requête HTTP/S** pour demander une ressource au serveur
2. Côté serveur, c'est le **serveur web** (exemple : **Apache**) qui traite les requêtes HTTP entrantes.
   Seulement, un serveur web ne sait répondre qu'aux requêtes visant des ressources statiques.
   Il ne peut que renvoyer des pages HTML, des images,... existantes.
3. Ainsi, si le **serveur web** s'aperçoit que la requête reçue est destinée au **serveur d'applications**, il la lui transmet.
4. Le **serveur d'applications** (exemple : **Tomcat**) reçoit la requête à son tour.
   Il exécute donc le morceau d'application auquel est destinée la requête.
   Cela peut passer par la consultation de sources de données, comme **des bases de données** (4'').
   Ou bien par l'interrogation d'autres **serveurs ou systèmes** (4'), par exemple : Un système de paiement électronique.
5. Une fois sa réponse générée, le **serveur d'applications** la renvoie, au **serveur web**.
   Celui-ci la récupère comme s'il était lui-même allé chercher une ressource statique.
6. La **réponse** est dorénavant du simple code **HTML**, compréhensible par un navigateur.
   Le serveur web peut donc retourner la réponse au client.

Les JSP (Java Server Pages) sont une technologie Java qui permet la génération de pages web dynamiques

# INTRODUCTION (3/3)

- Deux plateformes se sont progressivement imposées au fil du temps, dans le développement web avec **Java** :
  - le standard officiel **Java EE**,
  - le framework **Spring**.
- Chacun a ses avantages.
- Les deux plateformes s'influencent, et partagent le principe de la génération des pages **côté serveur.**
- Depuis quelques années, **ce principe est largement remis en cause** dans le développement web.
- Les principales raisons sont :
  - L'incapacité des serveurs à monter en charge avec un gros volume de connexions
  - Le transfert systématique de tous les éléments de la page entre deux navigations
  - La difficulté de créer des applications réactives à cause du réseau
  - La diversité des périphériques cibles (Ordinateur, Smartphone, SmartTV, etc.)

# INTRODUCTION

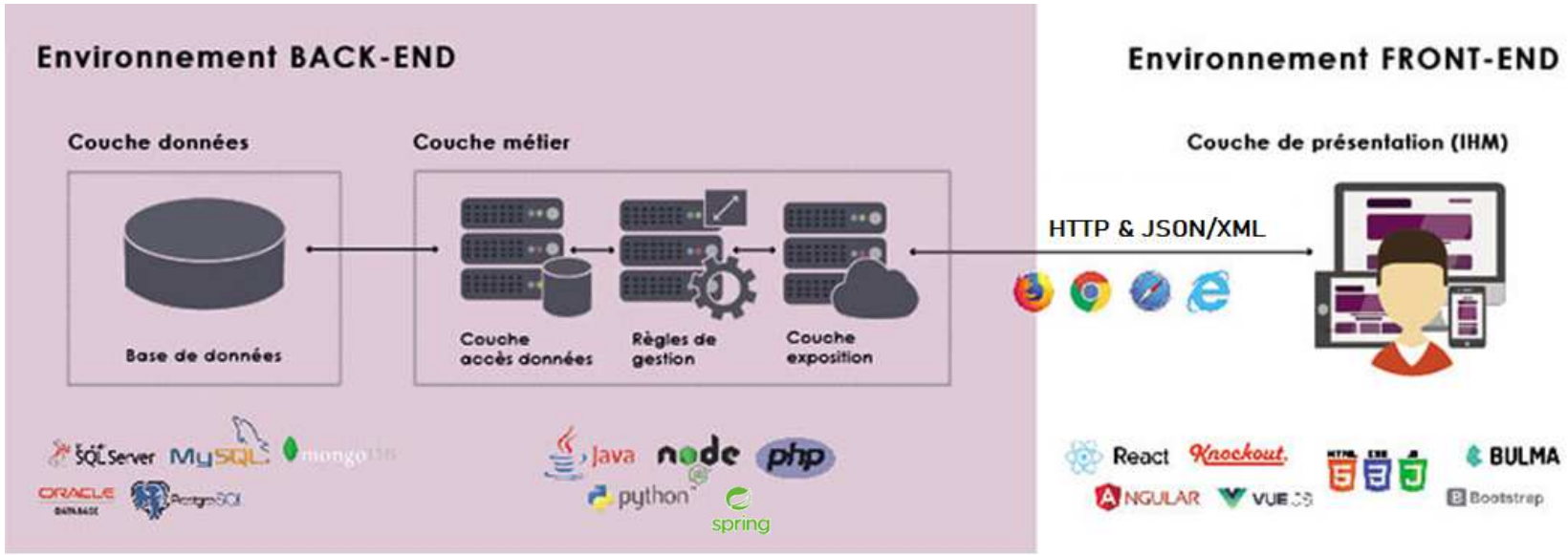
## DÉVELOPPEMENT DES API (1/4)

- Les informaticiens étant des personnes réfléchies et adeptes du moindre effort ont décidé de **couper ce lien fort qui existait entre la génération de la page web et le traitement des données.**
- Ainsi, les données sont désormais traitées au sein d'une **API** (Application Programming Interface) de **web services** et la gestion de la page est déportée côté client (le navigateur en l'occurrence, ou l'application mobile).
- Cette évolution représente un changement de paradigme très profond. Elle remet en cause toutes les technologies arrivées à maturité et installées dans le concept ancien où le serveur fait tout et le client ne fait rien.
- La nouvelle architecture sera donc constituée d'une partie **Back-end** qui met à disposition des informations à une partie **Front-end** à travers d'une **API**, dans ce cas, une application **client web riche** est recommandée

**Une API de Web Services** est une application qui permet d'échanger des données avec d'autres applications web. Même si ces dernières sont construites dans des langages de programmation différents.

# INTRODUCTION

## DÉVELOPPEMENT DES API (2/4)




Une **application client web riche** fonctionne de la manière suivante :

1. Le client riche (Partie Front-end) effectue une requête : HTTP & JSON/XML
2. Cette requête est transmise à l'API de web services(Partie Back-end).
3. La réponse est ensuite délivrée sous le même format que sa demande : HTTP & JSON/XML.

# INTRODUCTION

## DÉVELOPPEMENT DES API (3/4)

- **JSON** (JavaScript Objet Notation) est un langage léger d'échange de données textuelles très utilisé par les API de web services,
- Il est semblable à la syntaxe des objets JavaScript,
- **JSON** est souvent considéré comme une alternative à **XML** (Extensible Markup Language), un autre format d'échange de données

**JSON**

```
{"ordinateur":{
        "nom": "PC de jeu",
         "Composants":{
                          "cpu": "Intel i7 3.4GHz",
                          "ram": "16GB",
                          "stockage": "2TB HDD"
                          }
                  }
}
```

**XML**

```
<ordinateur>
   <name> PC de jeu </ name>
   <composants>
       <cpu> Intel i7 3.4GHz </ cpu>
        <ram> 16GB </ ram>
        <stockage> HDD 2TB </ storage>
    </ composants>
</ordinateur>
```

# INTRODUCTION

## DÉVELOPPEMENT DES API (4/4)

**Question:**

Imaginez que vous ayez un site web pour y exposer vos œuvres personnelles. Vous faites de la peinture et vous voulez que les gens soient capables d'acheter vos créations. Vous avez déjà construit votre site pour les montrer, mais maintenant vous avez besoin d'intégrer un système de paiement. Concevoir le code pour prendre en charge toutes les cartes bancaires serait incroyablement difficile. **Quelle solution vous proposez ? Même question si vous voulez repérer votre boutique sur une carte géographique.**

**Réponse:**

Intégrer une **API PayPal** et une **API Google Maps**

# INTRODUCTION

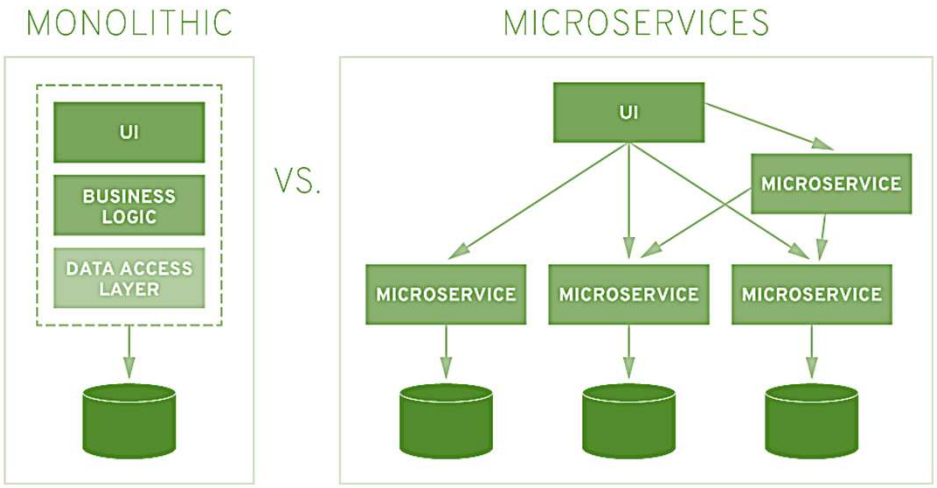
## LES MICRO SERVICES

**Problématique:**

Supposons que vous avez développé l'application, citée dans la diapositive précédente, avec **JEE**. Il est intéressant d'exploiter les capacités de différents langages pour des buts spécifiques, comme par exemple **R** pour faire du calcul statistique de vente ou **Nodejs** pour du temps réel.

**Problèmes :**

- Centralise tous les besoins
- Développer dans une seule technologie
- Devient plus complexe au fil du temps
- Une seule base de données




**Avantages:**

+ Les besoins sont distribués sur plusieurs services indépendants
+ Développés dans différents langages
+ Maintenus par plusieurs équipes
+ Différentes bases de données
+ Communiquent par le biais de HTTP et API REST

**Problèmes :** Complexité de gérer

- Le découpage de composants
- La résistance aux pannes
- La maitrise de différents langages et technologies

→ Migrer d'une application monolithique vers une architecture Micro services

# FRAMEWORK SPRING (1/2)

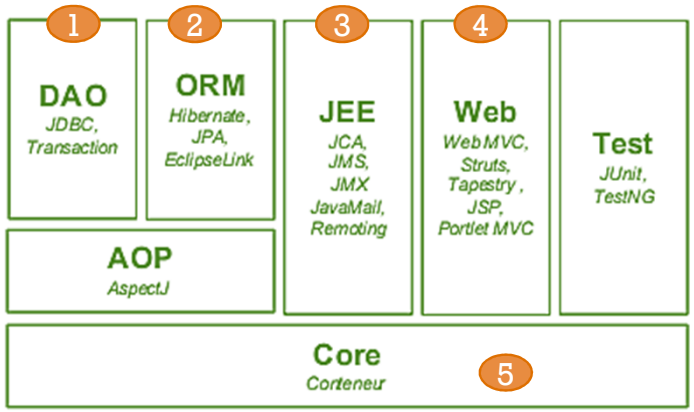
- **Spring** est un Framework Open Source rendant l'utilisation de Java EE à la fois plus simple et plus productive.

- **Spring** dispose de nombreux modules qui composent le Framework tels que :

1- **DAO** (Data Access Object), qui constitue le socle de l'accès aux dépôts de données, avec notamment une implémentation pour JDBC.

**3- Java EE**, un module d'intégration d'un ensemble de solutions populaires dans le monde de l'entreprise.

**5- Core**, le noyau, qui contient à la fois un ensemble de classes utilisées par tous les modules du Framework et le IOC Container ( Inversion Of Control Container)




*Architecture de Spring*

**2- ORM** (Object Relation Mapper), qui propose une intégration avec des outils populaires de mapping objet-relationnel, tels que Hibernate et JPA

**4- Web**, c'est un module comprenant le support de Spring pour les applications Web. Il contient notamment Spring Web MVC, la solution de Spring pour les applications Web, et propose une intégration avec de nombreux frameworks Web et des technologies de vue (Couche présentation).

**JDBC** (Java DataBase Connectivity) est une API fournie avec Java permettant de se connecter à des bases de données
**ORM** est une bibliothèque écrite dans la langue de votre choix qui encapsule le code nécessaire pour manipuler les données, donc vous n'utilisez plus SQL; vous interagissez directement avec un objet dans la même langue que vous utilisez.

# FRAMEWORK SPRING (2/2)

- **Spring** utilise certaines technologies comme :
  - XML Schema (fichier XML de configuration),
  - Les @annotations (apparues avec Java SE 5.0, très pratiques pour enrichir simplement l'environnement et les méta informations de classes).
  - Exemple d'annotation : **@Override**
    - Cette annotation est utilisée pour indiquer que la méthode remplace sa méthode de classe parente.

En Java, **une annotation** est une façon d'ajouter des méta-données à un code source.
Elles peuvent être ajoutées aux classes, méthodes, attributs, paramètres, variables locales et paquets.