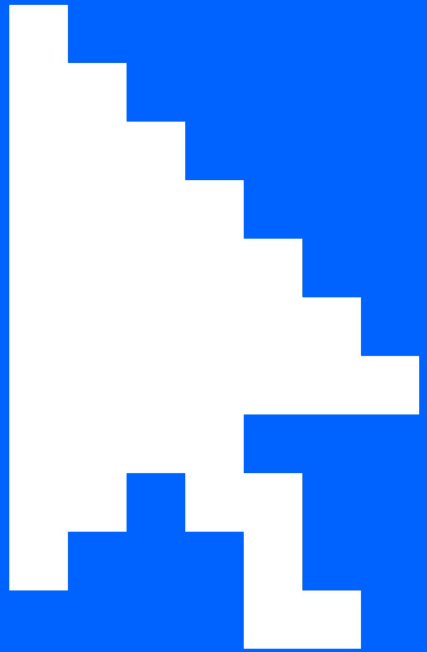
La Plateforme
La grande école du numérique pour tous

Jour 1 - Introduction à Spring Boot

Sommaire

- L'avant Java EE
- L'émergence de Spring
- Les modules Spring Framework
- IoC
- Spring boot
- Spring Initializr
- Gestion de projets et dépendances



La Plateforme
La grande école du numérique pour tous

Introduction à Java EE

Solutions entreprises

Qu’est-ce que Java EE ?

Java EE (Java Enterprise Edition), maintenant appelé **Jakarta EE**, est une plateforme de développement d’applications **entreprise** en Java.

Elle est conçue pour **faciliter** le **développement** d'applications web par exemple.

Son objectif

L'objectif de Java EE *était d'uniformiser les applications* ***entreprise*** *en offrant une architecture robuste et standardisée.*

Quels sont ses problèmes ?

Problèmes de Java EE :

❌ **Configuration XML lourde et complexe**.

❌ **Couplage fort** entre les composants (pas de flexibilité).

❌ **Difficulté à tester les applications (EJB peu testables en local)**.

❌ **Besoin d'un serveur d'application pour exécuter le code** (développement lent).

La Plateforme
La grande école du numérique pour tous

L'émergence De Spring

Solutions entreprises

Les dates clés de Java Spring

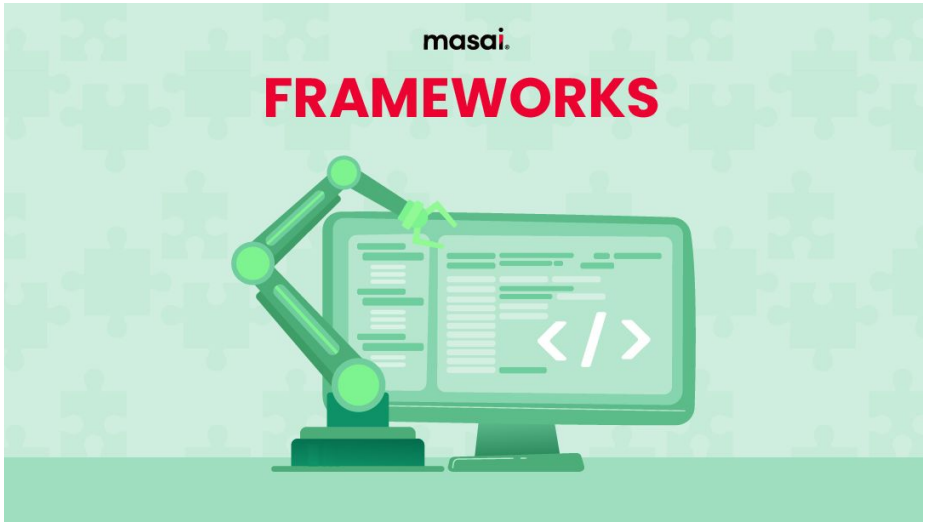
- **Octobre 2002** : **Rod Johnson** publie son livre *"Expert One-on-One J2EE Design and Development"*. Il critique la complexité de **EJB 2.0** et propose un framework plus simple.
- **Février 2003** : Avec l'aide de **Yann Caroff**, le projet devient **open source** et prend le nom **Spring**. Ce choix symbolise un "nouveau départ" après l'"hiver" du J2EE traditionnel.
- **Juin 2003** : Première version **0.9** sous licence **Apache 2.0**.
- **Mars 2004** : Sortie de la version **1.0** avec pour objectif de **simplifier l'architecture des applications d'entreprise** et d'améliorer la **testabilité et la maintenance**.
- **Octobre 2006** : Sortie de **Spring 2.0**, avec de nombreuses innovations. À cette date, Spring atteint **1 million de téléchargements** et reçoit **deux récompenses** : **Jax Innovation Award** et **Jolt Award**.
- **2007** : Version **2.5**, introduction des **configurations basées sur les annotations**.

Spring

Comme il supporte toutes sortes de développement applicatif, Spring a toujours été l'un des **frameworks les plus versatiles** et les **plus appréciés de tout l'écosystème Java** depuis **plus de 17 ans,** une éternité dans le monde ultrarapide de l'IT.

Spring est un framework open source développé pour construire et définir l'infrastructure d'une application Java, dont il facilite le développement et les tests.

Qu’est-ce qu’un Framework ?



Un Framework est une **boite à outils** pour un développeur web.

Un Framework contient des **composants autonomes** qui permettent de faciliter le développement d’un site web ou d’une application.

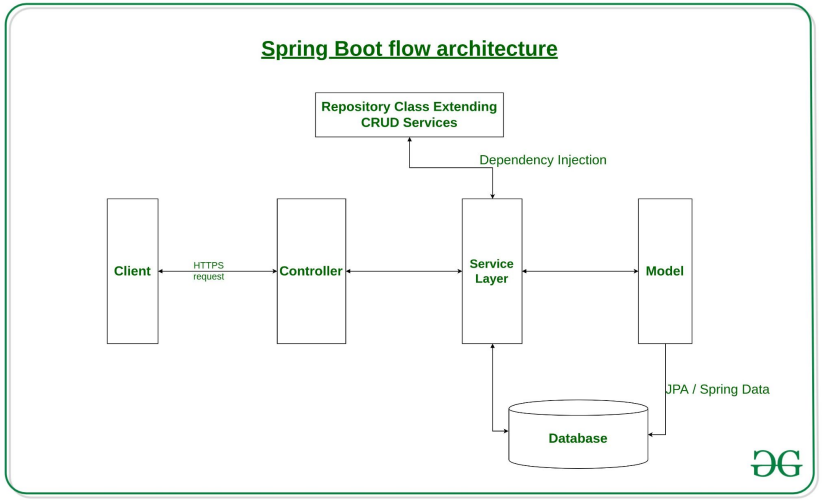
Ces composants résolvent des problèmes souvent rencontrés par les développeurs (CRUD, arborescence, normes, sécurités, etc.).

Ils permettent donc de gagner du temps lors du développement du site.

Application typique

Spring est généralement structurée en trois couches :

- **la couche présentation :** interface homme machine
- **la couche service :** interface métier avec mise en oeuvre de certaines fonctionnalités (transactions, sécurité, ...)
- **la couche accès aux données :** recherche et persistance des objets du domaine

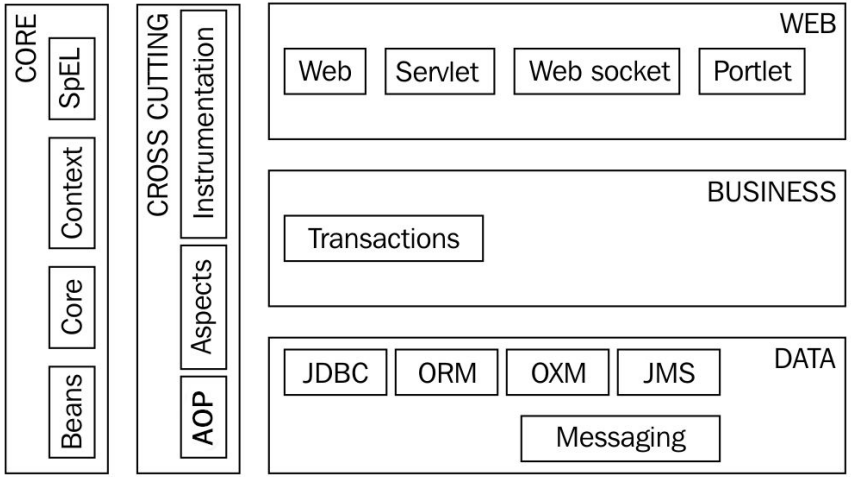


Les fonctionnalités de base de Spring

Spring simplifie le développement des applications Java. Les grands avantages du framework résident en son code source très épuré et un temps d'adaptation minime.

- **un conteneur léger** implémentant le design pattern IoC pour la gestion des objets et de leurs dépendances en offrant des fonctionnalités avancées concernant la configuration et l'injection automatique. Un de ses points forts est d'être **non intrusif** dans le code de l'application tout en permettant l'**assemblage d'objets faiblement couplés.**
- **une gestion des transactions par déclaration** offrant une abstraction du gestionnaire de transactions sous-jacent
- **faciliter le développement des DAO de la couche de persistance** en utilisant JDBC, JPA, JDO ou une solution open source comme Hibernate, iBatis, ... et une hiérarchie d'exceptions
- **un support pour un usage interne à Spring** (notamment dans les transactions) ou personnalisé de l'AOP qui peut être mis en oeuvre avec Spring AOP pour les objets gérés par le conteneur et/ou avec AspectJ
- **faciliter la testabilité** de l'application

Les modules Spring Framework



Parmi ces modules, il y a les modules fondamentaux qui font partie du noyau du Spring Framework (*core*) :

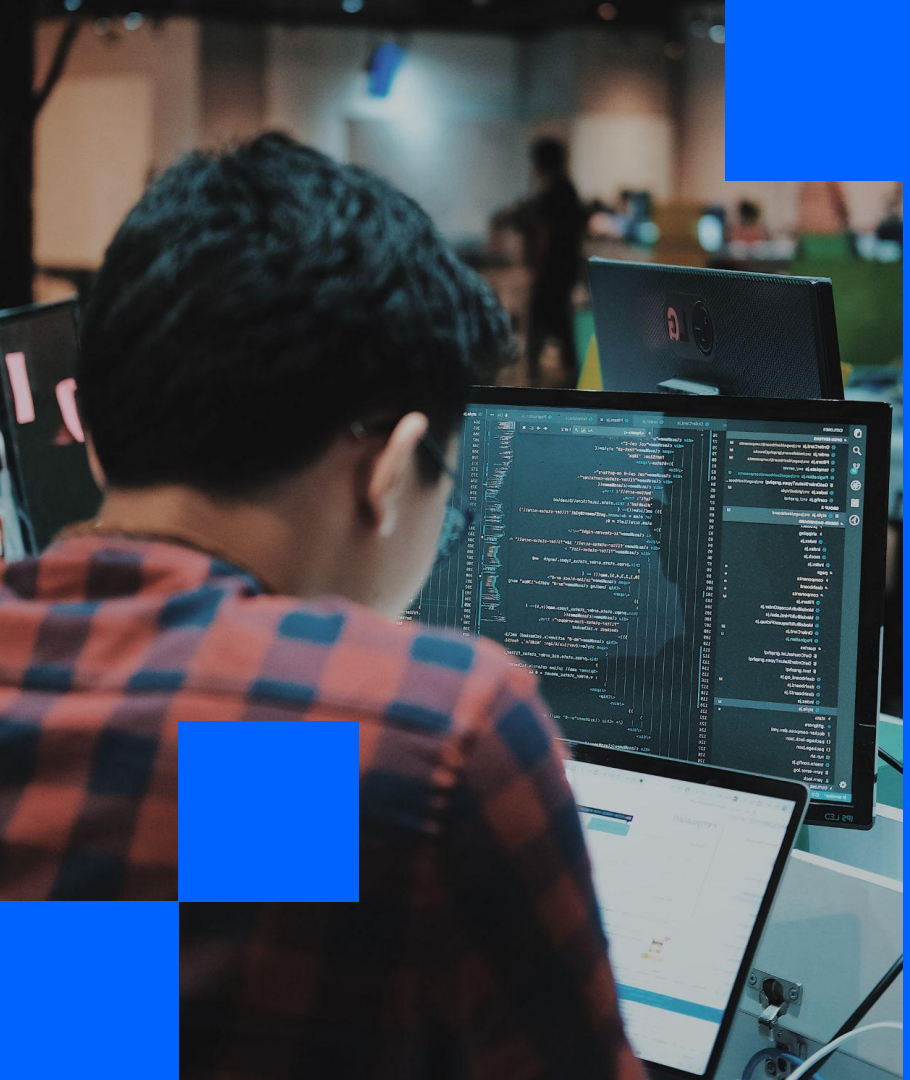
Core : Les classes fondamentales utilisées par tous les autre modules

Beans : Le module qui permet de manipuler les objets Java et de créer des *beans*

Context : Ce module introduit la notion de contexte d'application et fournit plusieurs implémentations de ces contextes. Avec ce module, il est possible de construire un conteneur léger IoC.

SpEL : Ce module fournit un interpréteur pour le langage d'expression intégré au Spring Framework (*Spring Expression Language* ou *SpEL*).

Critères	Spring Boot	Java EE
Simplicité et Configuration	Auto-configuration avec Spring Boot Starters Annotations simplifiant le code Moins de XML	Configuration XML lourde Nécessite un serveur d'application
Gestion des Dépendances	Gérée via Maven/Gradle avec BOM Versions compatibles automatiquement	Dépendances gérées manuellement Risque de conflits de versions
Déploiement	Application autonome Serveur intégré	Nécessite un serveur d'applications externe
Flexibilité et Modularité	Choix des modules nécessaires Adapté aux monolithes et microservices	Spécifications rigides Moins adapté aux microservices
Microservices	Nativement supporté avec **Spring Cloud**	Conçu initialement pour les monolithes
Sécurité	Intégration facile avec **Spring Security**	Sécurité plus complexe à mettre en place
Performance et Légèreté	Moins gourmand en ressources Facilement scalable	Plus lourd en ressources
Intégration avec DevOps	Compatible avec **Docker, CI/CD** Monitoring avec **Spring Boot Actuator**	Nécessite des outils externes pour DevOps
Écosystème et Communauté	Grande communauté open-source Documentation riche et mise à jour	Communauté active mais en perte de vitesse



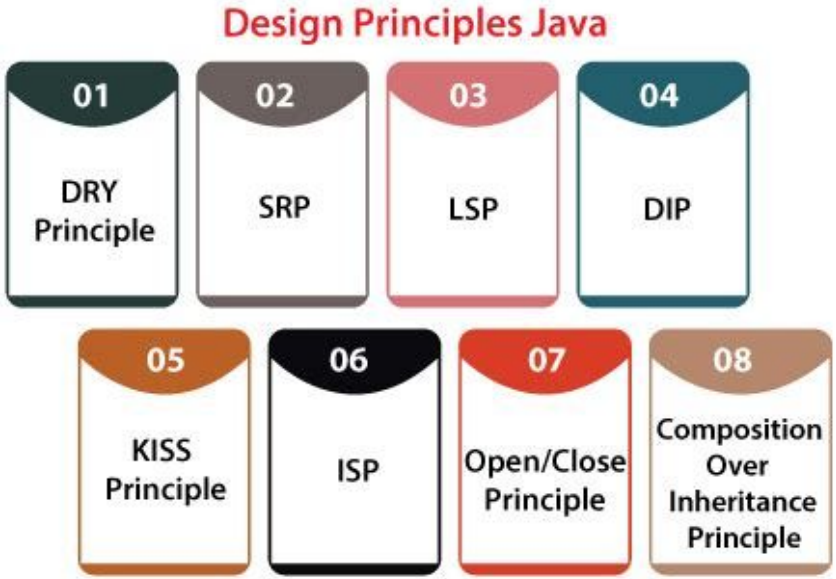
Principes de base de Spring

Design Principle

C'est une **règle générale** qui guide la conception logicielle pour améliorer la maintenabilité, la lisibilité et l'évolutivité.

Exemples : **SOLID, KISS, DRY, YAGNI**.

Il ne s'agit pas d'une implémentation spécifique, mais plutôt d'une philosophie de développement.

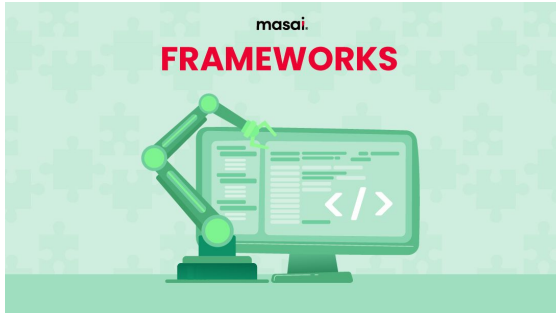


IoC (Inversion de Contrôle)

À la **base du Spring Framework,** on trouve un unique principe de conception : **l'inversion de contrôle.**

= façon de **concevoir l'architecture d'une application** en se basant sur le mécanisme objet de **l'injection de dépendance.**

Il consiste à **inverser le flux de contrôle traditionnel** dans un programme. Au lieu que le programme **gère directement l'instanciation et la gestion de ses dépendances**, il **délègue** cette responsabilité à un **framework ou un conteneur**.



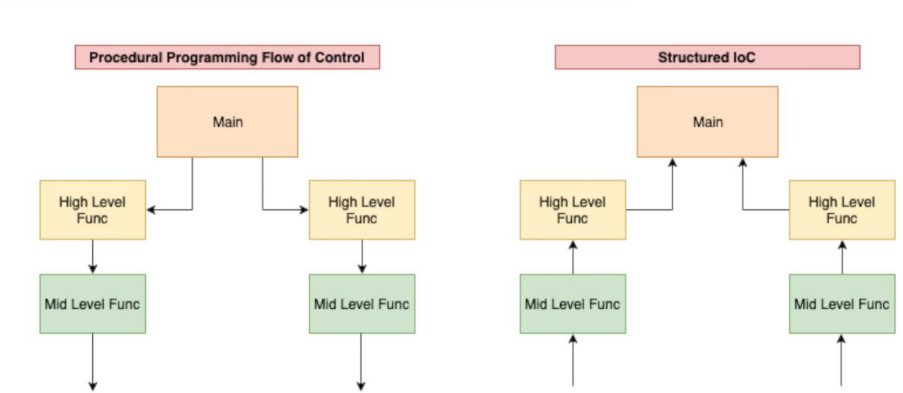
IoC (Inversion de Contrôle)

Problème sans IoC :

- Dans une application classique, les objets **créent directement** leurs dépendances via le mot-clé new.
- Cela rend le code **couplé, difficile à tester et à maintenir**.

Avec IoC :

- **Les dépendances sont injectées** automatiquement.
- Le code devient **plus modulaire, flexible et testable**.



Exemples concrets

exemple vie courante :

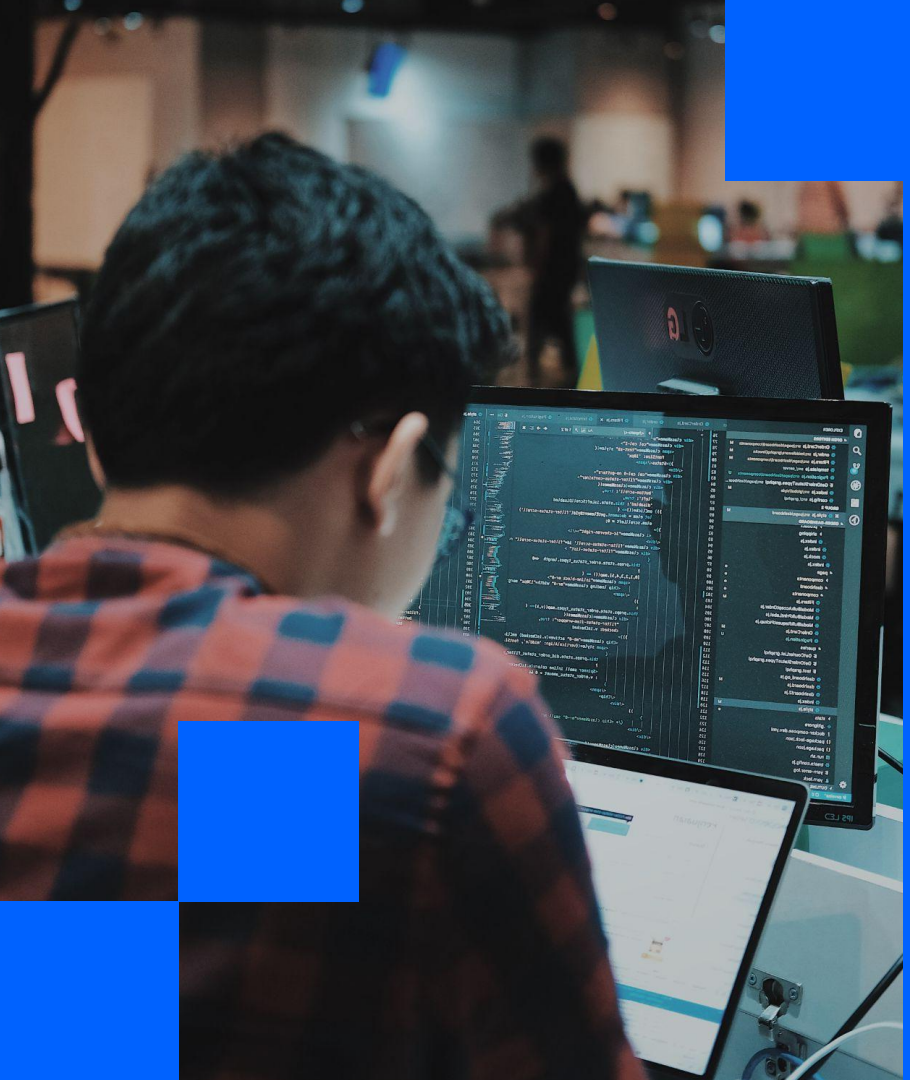
Le chef cuisine, quelqu'un d'autre fait les courses, quelqu'un d'autre lave

- Chacun son rôle = faible couplage
- Si une personne manque, on peut facilement la remplacer
- Le "manager" (Spring) coordonne tout

exemple informatique :

Habituellement, si je veux développer un programme qui utilise des bibliothèques tierces, je vais appeler les objets ou les fonctions de ces bibliothèques dans mon programme. **Le flux de contrôle est géré par mon propre code qui doit réaliser les appels au code tiers.**

Dans une approche IoC, le flux de contrôle est orienté du code tiers vers le code de mon application. **Le code que je fournis sera sous le contrôle du code tiers. L**'IoC est en fait le principe qui est mis en application par la plupart des frameworks.

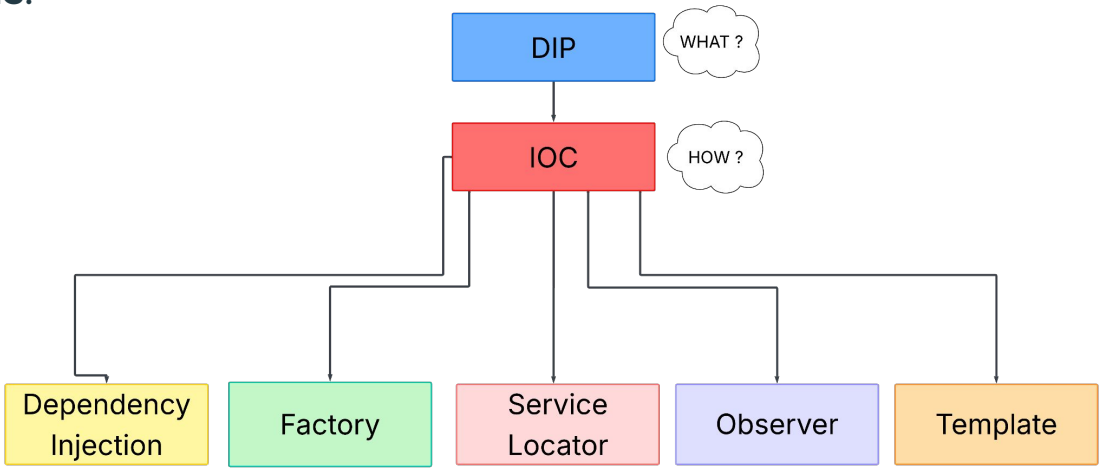


Injection de dépendances

design pattern Dependency Injection

L'**Injection de Dépendances** est un **design pattern** qui permet de **fournir les dépendances d'une classe depuis l'extérieur**, plutôt que la classe ne les crée elle-même. Il s'agit d'une application concrète du principe **IoC**.

Spring Boot utilise un **contenant IoC** qui gère la création et l'injection des objets grâce à des **annotations**.



UserService est **couplée** à UserRepository
Difficile de tester (comment mocker UserRepository ?)
Difficile de changer d'implémentation
UserService doit connaître comment créer UserRepository

```java
// SANS IoC - Problématique
public class UserService {
    private UserRepository userRepository;

    public UserService() {
        // La classe crée elle-même sa dépendance
        this.userRepository = new UserRepository();
    }
}
```

Spring scanne les classes avec les annotations (@Service, @Repository, @Component)
Spring crée un "conteneur" avec toutes ces classes (appelées "beans")
Spring analyse les dépendances (qui a besoin de quoi ?)
Spring injecte automatiquement les bonnes dépendances au bon endroit

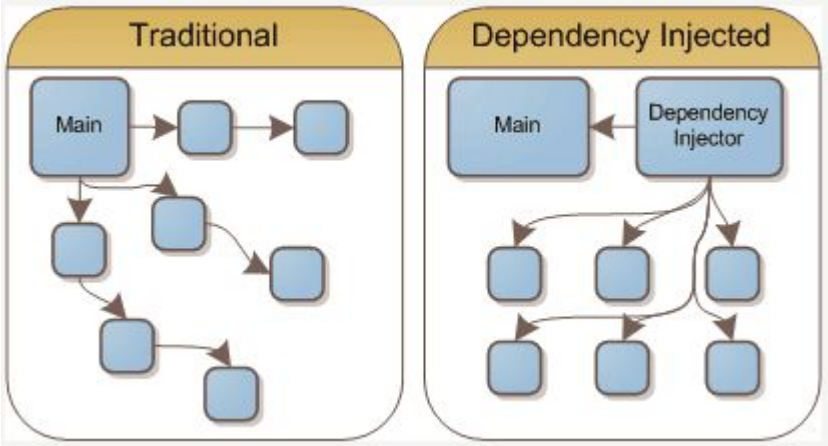
```java
// AVEC IoC - Solution Spring
@Service
public class UserService {
    private final UserRepository userRepository;

    // Spring injecte automatiquement UserRepository
    @Autowired
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
}

@Repository
public class UserRepository {
    // Logique d'accès aux données
}
```

Intérêts du DI

- ➔ **Réduction du couplage** → Les classes ne sont plus responsables de créer leurs dépendances.
- ➔ **Facilité de test** → Permet de substituer facilement les dépendances par des mocks dans les tests.
- ➔ **Code plus lisible et maintenable** → La gestion des objets est centralisée.



La Plateforme
La grande école du numérique pour tous

Spring boot

Solutions entreprises

L’histoire de Java Spring Boot

- **Octobre 2012** : À la demande de **Mike Youngstrom**, une extension de **Spring Framework** est proposée pour permettre une **architecture web sans conteneur**.
- **Début 2013** : Développement du projet **Spring Boot** pour répondre à cette problématique.
- **Avril 2014** : Sortie de **Spring Boot 1.0**, qui vise à **simplifier la gestion des dépendances** et **accélérer le développement** d'applications autonomes basées sur Spring.
- **Problèmes résolus par Spring Boot** :
  - Gestion automatique des dépendances (Spring Boot Starters).
  - Réduction des efforts manuels pour **maintenir et sécuriser les librairies**.
  - Simplification du développement d’applications **indépendantes et prêtes pour la production**.
- **Aujourd’hui** : Spring Boot est devenu **la norme** pour la majorité des nouvelles applications **Spring**, facilitant **l'intégration, la configuration et le déploiement**

Pourquoi utiliser Spring Boot ?

- **Légèreté** : Spring Boot a la particularité d'être très léger et d'embarquer avec lui le strict minimum pour faire tourner votre service.
- **Intégration facilitée** : Spring Boot s'intègre particulièrement bien dans une architecture orientée micro services.
- **Simplicité de prise en main** : Spring Boot permet donc de créer une API de services très simplement.

Pourquoi utiliser Spring Boot plutôt que Spring seul ?

Spring est un **framework généraliste.**

Spring Boot est une **extension** qui **simplifie** la configuration et permet de **créer rapidement** des applications autonomes.
Il **simplifie** la gestion des dépendances, **automatise** la configuration, **intègre** un serveur web embarqué et **facilite** le déploiement.

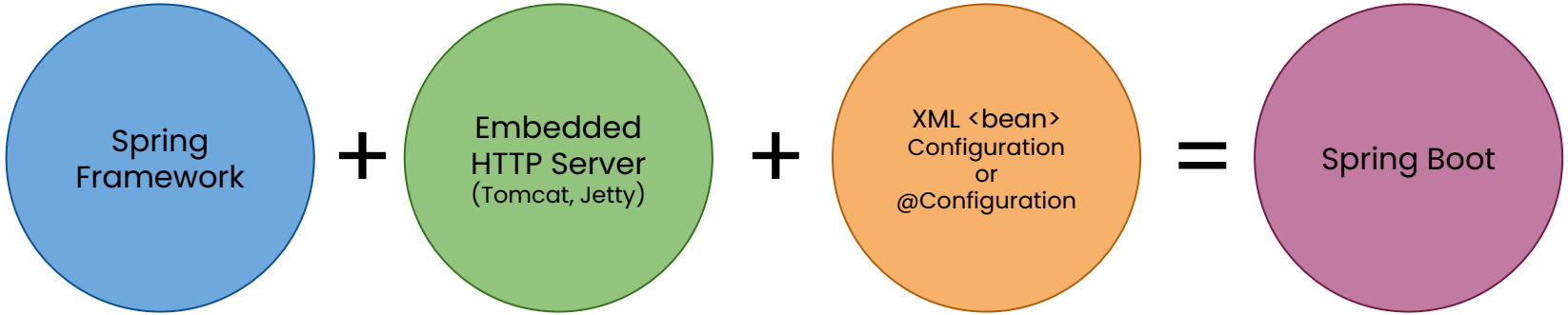
Pourquoi Spring Boot est-il utilisé dans les projets modernes ?

	Avantages de Spring Boot
Simplicité & Productivité	Configuration automatique avec **Spring Boot Starters** Serveurs embarqués
Support des Microservices	Intégration avec **Spring Cloud** Léger et modulaire, parfait pour le cloud
Intégration DevOps & Cloud	Compatible **Docker** Intégration avec **CI/CD** Monitoring avancé avec **Spring Boot Actuator**
Sécurité & Gestion des Dépendances	Sécurisation simplifiée avec **Spring Security** Gestion automatique des dépendances
Écosystème Riche & Communauté	Support de **JPA/Hibernate** pour les bases de données Compatible avec **REST, GraphQL et WebSockets** **Grande communauté open-source** et documentation complète

Spring Boot est-il adapté aux petits projets ou seulement aux grandes applications ?

Type de projet	Pourquoi utiliser Spring Boot ?
🟢 **Petits projets**	Permet un **démarrage rapide** grâce aux configurations automatiques Serveur embarqué → **pas besoin d'un serveur externe** Gestion simplifiée des dépendances avec **Spring Boot Starters**
🟡 **Projets moyens**	Favorise une **architecture modulaire** et évolutive Intégration facile avec des bases de données Sécurité intégrée avec **Spring Security**
🔴 **Grandes applications**	Support des **microservices** avec **Spring Cloud** Monitoring et gestion avancée avec **Spring Boot Actuator** Facilement scalable dans des environnements **cloud**

Spring vs Spring Boot



La Plateforme
La grande école du numérique pour tous

Spring Initializr

Spring Initializr

Spring Initializr est un outil **générateur de projet Spring Boot** qui permet aux développeurs de **créer rapidement une application Spring Boot pré-configurée** en quelques clics.

Spring Initializr

Pourquoi utiliser Spring Initializr ?

- ➔ **Gain de temps** → Génère automatiquement la structure du projet sans avoir à tout configurer manuellement.
- ➔ **Simplicité** → Permet de sélectionner les dépendances nécessaires (Spring Web, JPA, Security, etc.).
- ➔ **Éviter les erreurs de configuration** → Les fichiers **pom.xml (Maven)** ou **build.gradle (Gradle)** sont générés avec les bonnes versions de dépendances.
- ➔ **Compatible avec plusieurs IDE** → Importation directe dans IntelliJ IDEA, Eclipse, VS Code, etc.

La Plateforme
La grande école du numérique pour tous

Gestion de projet et dépendances

Solutions entreprises

Maven/ Gradle

Outils de gestion de projet et d'automatisation de build en Java. Ils permettent d'automatiser **la compilation, la gestion des dépendances, le packaging et le déploiement** d'une application.

Critère	Maven	Gradle
Langage	XML (pom.xml)	Groovy/Kotlin (build.gradle)
Performance	Plus lent, exécution **séquentielle**	Plus rapide, exécution **parallèle** et **incrémentale**
Simplicité	Facile à apprendre, basé sur des conventions	Plus flexible mais nécessite un apprentissage
Personnalisation	Peu flexible	Très flexible (scripts puissants)
Utilisation	Java, Spring Boot, gros projets d'entreprise	Android, microservices, projets modernes
Gestion des dépendances	Maven Central Repository	Gradle Plugin Portal + Maven Central

Quand utiliser qui ?

Utiliser Maven si :

- ➔ Vous travaillez sur un projet **classique Java / Spring Boot**.
- ➔ Vous voulez **suivre des conventions standard** sans trop de personnalisation.
- ➔ Vous avez une **grande équipe** qui veut un outil **stable et mature**.

Utiliser Gradle si :

- ➔ Vous développez une **application Android**.
- ➔ Vous voulez un **build plus rapide** et **performant**.
- ➔ Vous avez besoin de **personnaliser fortement le processus de build**.

Le fichier pom.xml dans un projet Spring Boot

Le fichier **pom.xml (Project Object Model)** est un fichier **Maven** utilisé pour :

- **Gérer les dépendances** (bibliothèques et frameworks nécessaires)
- **Configurer le projet** (version Java, plugins, options de compilation)
- **Automatiser la compilation et le packaging** de l'application
- **Définir les versions compatibles** des dépendances grâce au **Spring Boot Starter Parent**

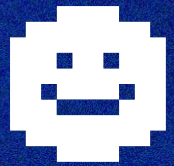
C'est un élément clé d'un projet Spring Boot géré avec Maven !

Ajout de dépendances

Une dépendance est une **bibliothèque externe** (ou module) que l'application utilise pour fonctionner correctement. Ces dépendances permettent d'ajouter des fonctionnalités sans avoir à les coder soi-même.

```xml
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>
            spring-boot-starter-web
        </artifactId>
    </dependency>
</dependencies>
```

```
mvn clean install
```



La Plateforme