Spring Boot

FRAMEWORK SPRING

Spring Core : Ce composant est la base de l'écosystème Spring.

- Contient le "core container" (ce qui permet l'injection de dépendances)
- Contient également Spring MVC qui permet de faire du web et de Data Access qui fournit des éléments fondamentaux pour la communication avec les bases de données.

Spring Data

- Permet de communiquer avec de nombreux types de bases de données

Spring Security

- L'un des plus critiques de Spring Framework, bien qu'il soit aussi l'un des plus complexes. Il permet de gérer l'authentification, l'autorisation, et la sécurité des API.

Spring Cloud

- Spring Framework fournit Spring Cloud pour la gestion des microservices

Spring Boot

- Composant très particulier de Spring Framework, il permet de mettre en œuvre tous les autres.
- C'est un composant au service des autres composants.
- Ses avantages qui sont :
  - l'autoconfiguration automatique de Spring ;
  - des starters de dépendances ;
  - des endpoints Actuator pour fournir des données sur l'application.

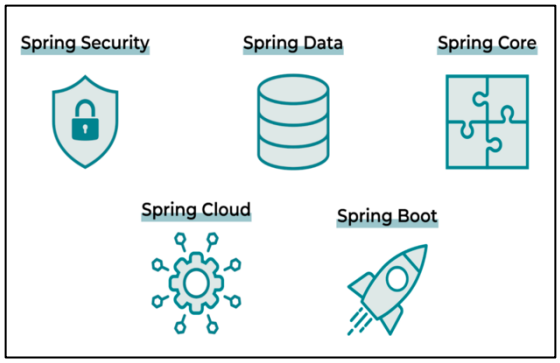

Figure 1 FRAMEWORK SPRING

SPRING BOOT

Les avantages de Spring Boot

1. Optimisation de la gestion des dépendances
2. Autoconfiguration
3. Monitoring
4. Le déploiement

Optimisation de la gestion des dépendances

- Spring Boot fournit des **starters**, qui correspondent à un ensemble de dépendances homogénéisées (associations, versions). Onpeut les comparer à des kits de dépendances.

- Nul besoin de définir les versions des dépendances explicitement dans le pom.xml : Maven les déduit grâce à la version de Spring Boot utilisée
- Tous les starters sont préfixés par "**spring-boot-starter**". Voici quelques exemples de starters :
    - spring-boot-starter-core;
    - spring-boot-starter-data-jpa;
    - spring-boot-starter-security;
    - spring-boot-starter-test;
    - spring-boot-starter-web.

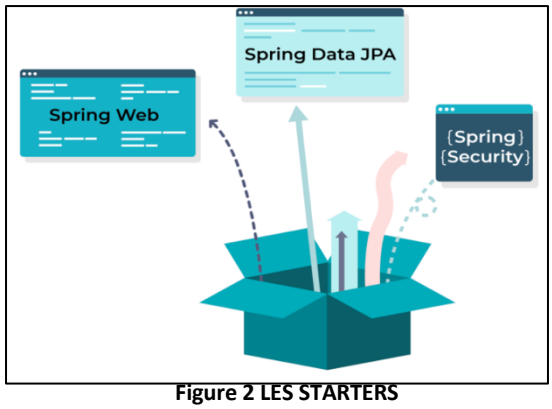

Figure 2 LES STARTERS

Autoconfiguration

- L'avantage le plus important de Spring Boot. Avec Spring Boot, il y a beaucoup moins de configuration (concernant la gestion des servlets, le chargement du contexte Spring, la connexion à la base de données).
- L'utilisation de Spring Boot, et l'annotation @SpringBootApplication placée au niveau de la classe principale (celle générée automatiquement), déclenchent automatiquement de nombreuses opérations en background qui nous sont nécessaires.

Monitoring

- Spring Boot Actuator correspond à une fonctionnalité de Spring Boot qui permet de monitorer et de manager dynamiquement le programme (API ...)

Le déploiement

- Un projet Spring Boot contient un tomcat embarqué au sein même du JAR généré. Le projet web peut donc être déployé au sein de ce tomcat embarqué.

Exemple de différences entre deux Projets, en mettant en avant les avantages de Spring Boot.

Répertoires	**Projet « Webwithoutsb » Sans Spring Boot**	**Projet « Webwithoutsb » Avec Spring Boot**
src/main/java	**JpaConfig.java** : créer manuellement	**WebwithsbApplication.java** : c'est une classe créée automatiquement par Spring Boot
src/main/resources	•**contextFront.xml** : configuration manuelle du scanning et du viewresolver •**META-INFO/persistence.xml** : contient la configuration de la BDD	•**Pas besoin de contextFront** : Spring Boot s'en occupe •**application.properties** : contient la configuration de la BDD. Et plus simple à aborder que XML. •**template/home.html** : rendu HTML à fournir.
src/main/webapp	**2 fichiers :** •**template/home.jsp** : rendu HTML à fournir •**web.xml** : fournit de la configuration pour la gestion des servlets	Ce dossier **n'existe pas** ! Spring Boot n'a pas besoin de tout ça

fichiers pom.xml	**8** dépendances + **2** dépendances dans le **dependencyManagement** pour les versions	**5** dépendances **sans avoir défini les versions**

L'annotation @SpringBootApplication

L'IoC container va instancier pour vous des classes, puis si nécessaire les injecter dans d'autres instances de classe.

Pour qu'une classe soit manipulée par l'IoC container, il est nécessaire de l'indiquer explicitement à Spring.

L'annotation @SpringBootApplication va permettre à l'IoC container de manipuler la classe concernée.

@SpringBootApplication est composé de 3 autres annotations :

- @SpringBootConfiguration : la classe sera utilisée comme une classe de configuration.
- @EnableAutoConfiguration : active la fonctionnalité d'autoconfiguration de Spring Boot.
- @ComponentScan : active le "scanning" de classes dans le package de la classe et dans ses sous-packages. Sans cette annotation, l'IoC container ne tiendra pas compte des sous classes.

Spring Boot fournit une interface nommée "CommandLineRunner". En implémentant cette interface, la classe sera obligée de déclarer la méthode "public void run(String... args) throws Exception".

Si la classe est un bean (chargée dans le contexte Spring), Spring Boot exécutera la méthode run au démarrage.

Deux choix :

- soit modifier la classe HelloWorldSpringBApplication afin qu'elle implémente CommandLineRunner et la méthode run, avec comme corps de méthode un "System.out.prinln("Hello World!")" ;
- soit créer une nouvelle classe qui implémente CommandLineRunner, la méthode run , et qui a une annotation @Component (au-dessus du nom de la classe).

En résumé

- Au sein d'une application Spring Boot, écrire du code implique de définir les beans utilisés par Spring.
- On peut exécuter du code grâce à l'implémentation de l'interface CommandLineRunner et de la méthode run.
- Pour qu'une classe soit déclarée en tant que bean, on l'annote @Component.
- Pour qu'un bean soit injecté dans un attribut, on annote l'attribut @Autowired.

SPRING BOOT : TEST

- **@SpringBootTest** est une annotation fournie par Spring Boot.
- Elle **permet lors de l'exécution des tests d'initialiser le contexte Spring**. Les beans de l'application peuvent alors être utilisés.
- Pour rappel : un test s'exécute de façon unitaire et autonome.
- Par défaut, le test n'a donc aucune connaissance du contexte Spring. Dans le cas d'une application Spring Boot, c'est un vrai problème !

- Le problème est résolu grâce à l'annotation @SpringBootTest.
- La méthode contextLoads est **annotée @Test** (annotation qui provient de JUnit), et n'a pas de contenu

SPRING BOOT : API

- Une entreprise myHR souhaite offrir un service de gestion d'employés aux petites entreprises.
- L'idée est d'offrir une suite d'outils (application web, application mobile) prête à l'emploi.
- Pour lancer ce projet, myHR souhaite avant tout mettre à disposition une API qui permettra à toutes les autres applications d'accéder aux mêmes données.
- L'idée étant de gérer des employés, l'API devra donc offrir un CRUD pour les données des employés.
- Les données seront dans une base de données H2 : H2 est une base de données relationnelle Java très légère, qui par défaut fonctionne en "in memory". Au démarrage du programme, la structure de la base est construite ; et lorsque le programme s'arrête, le contenu de la base de données est supprimé.
- L'API Rest devra donc exposer des Endpoints correspondant aux actions du CRUD, et communiquer avec la base de données pour récupérer ou modifier les informations des employés.