

SPRING



SPRING BOOT

Introduction à SPRING BOOT

PRÉSENTATION DE SPRING BOOT

Spring est un formidable outils de développement d'applications

Mais sa configuration peut être lourde, compliquée et chronophage

Spring Boot apporte deux fonctionnalités principales

- L'auto-configuration
- Les starters

PRÉSENTATION DE SPRING BOOT

Spring Boot c'est

- Démarrage rapidement du développement d'un projet Spring
- Gestion des dépendances
- Suppression de la configuration (ou d'une bonne partie)
- Mise à disposition des aspects communs non-fonctionnels d'une application
- Exposition d'un nombre conséquent de fonctionnalités par défaut

PRÉSENTATION DE SPRING BOOT

Spring Boot n'est pas

- Un outil de prototypage
- Une version de Spring « dégradée »
 - On a accès à toute la stack Spring
- Seulement pour les débutants Spring
- Seulement pour les applications de conteneurs intégrés
- Seulement pour les applications Web

AUTO-CONFIGURATION

Spring Boot remplace

- Les fichiers de configuration XML
- Les classes de configuration

Par une simple annotation @EnableAutoConfiguration

Et ... et c'est tout!

AUTO-CONFIGURATION

Spring Boot vient scanner les packages de l'application par défaut

- Il faut pour cela
 - Que l'application principale soit dans un package « racine »
 - Que toutes les classes soient dans des sous-packages de ce package « racine »
 - Exemple :

Package racine fr.formation

Package model fr.formation.model

Package DAO fr.formation.dao

Package Controller fr.formation.controller

Package RestController fr.formation.api

• ...

- Si les packages à scanner ne sont pas dans le package « racine »
 - Il faut ajouter l'annotation @ComponentScan, comme dans une configuration par classe classique

AUTO-CONFIGURATION

Spring Boot configure automatique un maximum d'éléments

- Mais il peut arriver qu'on ait des besoins en configuration spécifiques
 - Modification des valeurs Spring Boot par défaut
 - Spring Boot utilise un fichier de properties disponible dans src/main/resources/application.properties
 - C'est dans ce fichier qu'on viendra modifier les valeurs par défaut
 - Ajout de nouvelles classes de configuration
 - Annotées de @Configuration et autres annotations utiles

STARTERS

Un starter apporte un ensemble de dépendances au projet

C'est un squelette à ajouter au POM

Il est possible d'utiliser plusieurs starters

Tous les starters sont au format spring-boot-starter-NOM_DU_STARTER

Starter Web spring-boot-starter-web

Starter Data JPA spring-boot-starter-data-jpa

Un autre avantage est la gestion des versions

Plus besoin de chercher quelle version est compatible, Spring Boot l'a fait !

STARTERS

Il faut appréhender Spring Boot comme ceci

- J'ai une problématique
- Je cherche un starter qui réponde à mon besoin
- J'inclue le starter
- Je commence à travailler avec

Et pour trouver son starter, se référencer à la documentation officielle

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter



Implémentation de Spring Boot

Nous allons utiliser Spring Boot avec MAVEN

- Commençons par faire hériter notre projet du POM spring-boot-starter-parent
 - Il regroupe une liste complète de dépendances, vous évitant notamment de gérer vous-même les versions
 - Il permet de surcharger la version de Java et l'encodage des sources facilement

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.2.RELEASE</version>
</parent>
```

Pour modifier la version de Java (en Java 9 par exemple)

Modifier la propriété java.version

Pour modifier l'encodage (en UTF-8 par exemple)

Modifier les propriétés project.build.sourceEncoding et project.reporting.outputEncoding

On peut (idéalement) préciser une option de build MAVEN

En utilisant un plugin Spring Boot

Comme pour chaque application, il faut une classe principale

Voici la méthode main d'une application Spring Boot

```
@SpringBootApplication
public class Application {
   public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
   }
}
```

L'annotation @SpringBootApplication encapsule les annotations suivantes

- @Configuration
- @EnableAutoConfiguration
- @ComponentScan

La classe **Application** est la classe principale, mais aussi une classe de configuration

- On peut y ajouter des @Bean
- On peut y ajouter des annotations pour prendre des paramètres supplémentaires en compte

• ...

MAVEN configuré (avec le ou les starter(s)), classe principale écrite

- Le projet est prêt à être exécuté!
- ... Mais l'application ne fait pas grand-chose pour le moment !



SPRING INITIALIZR

Starter de Starters

SPRING INITIALIZR

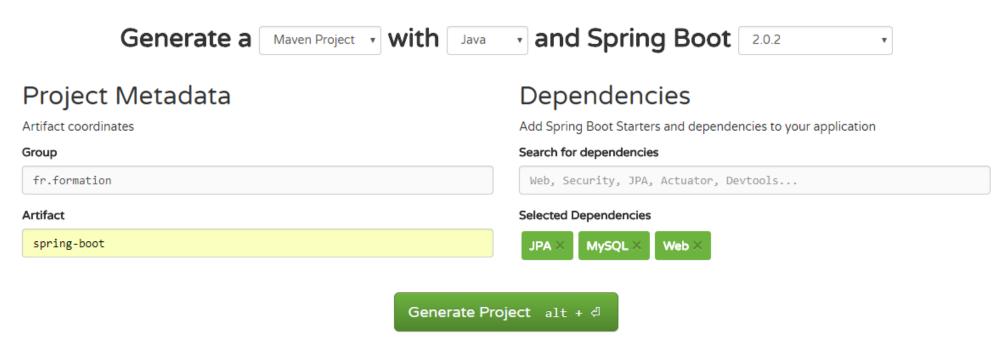
Que serait Sprint Boot sans un starter qui facilite la création d'un nouveau projet?

Spring Initializr répond parfaitement à cette problématique

- https://start.spring.io/
- Permet de générer un nouveau projet Spring Boot rapidement
- Sélectionner projet Maven avec Java
- Cliquer sur « Switch to full version » pour plus d'options

SPRING INITIALIZE

SPRING INITIALIZR bootstrap your application now



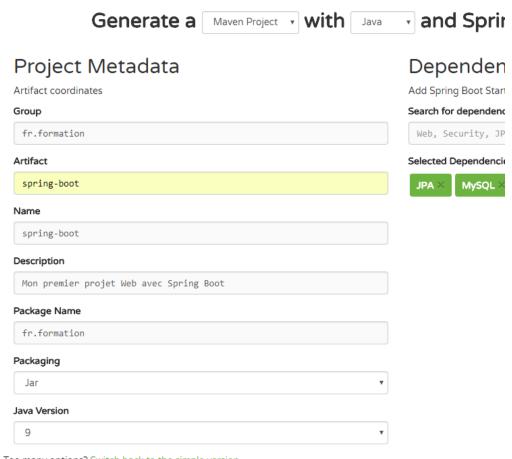
Don't know what to look for? Want more options? Switch to the full version.

SPRING INITIALIZE

Pour démarrer un nouveau projet Web

- Saisir les informations
 - Group, Artifact et Name de votre projet MAVEN
 - Votre package « racine »
 - Le packaging
 - La version de Java
- Sélectionner les dépendances
 - Web
 - JPA
 - MySQL

SPRING INITIALIZR bootstrap your application now



Too many options? Switch back to the simple version.

Generate Project alt + △

SPRING INITIALIZR

Une fois prêt

- Télécharger le projet
- Désarchiver le projet
- Importer le projet MAVEN sous Eclipse

Et c'est prêt!

SPRING INITIALIZR

La même manipulation est possible directement depuis Eclipse

- Grâce au plugin Spring Tools (Spring IDE)
- Ajout d'un nouveau projet « Spring Starter Project »
 - Qui utilise Spring Initializr
 - Toutes les options du site sont disponibles ici !

EXERCICE

Créer un nouveau projet avec Spring Initializr

Ne sélectionner aucune dépendance

Tester l'exécution du projet



SPRING BOOT CONSOLE

Application console

SPRING BOOT CONSOLE

Pour créer une application console avec Spring Boot

Définition d'une classe de Service qui implémente l'interface CommandLineRunner

```
@Service
public class ConsoleApplication implements CommandLineRunner {
    @Override
    public void run(String... args) throws Exception {
        System.out.println("Je suis une application Spring Console !");
    }
}
```



SPRING BOOT DATA JPA

Boot & Data JPA

SPRING BOOT DATA JPA

On souhaite que notre application puisse utiliser une base de données

- Avec MySQL et Hibernate
- Ajoutons le starter data-jpa

SPRING BOOT DATA JPA

On utilise Hibernate et MySQL

- Il va falloir préciser à Spring quelques informations de configuration
- Dans le fichier application.properties

```
# Informations de connexion
spring.datasource.url = jdbc:mysql://localhost:3306/eshop
spring.datasource.username = root
spring.datasource.password =

# Afficher les requêtes SQL
spring.jpa.show-sql = true

# DDL (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update

# Dialect
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

EXERCICE

Ajouter le starter data-jpa au projet

Ajouter la dépendance « eshop-model »

Configurer la source de données

Reprendre la DAO Repository **Produit**

Modifier l'application

Parcourir la liste des produits, et les afficher dans la console



SPRING BOOT WEBMVC

Boot & MVC

SPRING BOOT WEBMVC

On souhaite que notre application soit une application web

Ajoutons le starter web

On souhaite utiliser Thymeleaf

- Ajoutons le starter thymeleaf
- Ajoutons la dépendance thymeleaf-layout-dialect (sans sa version, Spring Boot la gère !)

SPRING BOOT WEBMVC

Par défaut, Boot va chercher dans ces répertoires

- /resources/static
 Pour toutes les ressources statiques (CSS, JS, Images, ...)
- /resources/templatesPour les templates (de Thymeleaf par exemple)

EXERCICE

Ajouter les starters web et thymeleaf

Supprimer le @Service de la classe Console

Créer un @Controller HomeController

Mappée sur « / », retourne la vue « home »

Exécuter l'adresse http://localhost:8080/

EXERCICE

Ajouter un @Controller « ProduitController »

- Mappée sur « /produits »
- La vue retourne la liste des produits dans un tableau (style Bootstrap)



Boot & REST

Ici, les mécanismes REST sont déjà chargés avec le starter web

On peut aller plus loin et exposer nos Repositories sous forme d'API

Ajoutons le starter data-rest

Pour ne pas gêner les contrôleurs MVC « classiques »

- Modifions le mapping de base de l'API REST
 - Dans le fichier application.properties

spring.data.rest.basePath = /api

EXERCICE

Ajouter le starter data-rest

Exécuter les adresses

- http://localhost:8080/api/
- http://localhost:8080/api/produits
 - Comment est-ce possible ?

Par défaut, Data Rest expose toutes les Repositories

- Le nom de la classe modèle avec un « s »
- Donc pour Produit, c'est « produits »

EXERCICE

Ajouter la signature suivante à la DAO

```
@RestResource(path="by-libelle")
public Produit findByLibelle(@Param("libelle") String libelle);
```

Se rendre à l'adresse http://localhost:8080/produits/search

• Que remarquez-vous ?

Compléter la DAO pour rechercher un produit dont le nom contient la recherche



Boot & Security

L'API « produits » est visible par défaut, par tout le monde !

Ajoutons le starter security

Par défaut, Boot va rendre l'accès <u>uniquement</u> sur authentification

Nom d'utilisateur user

Mot de passe auto-généré, disponible dans la console

Pour surcharger la configuration Boot Security par défaut

- Création d'une classe de configuration SecurityConfig qui hérite de WebSecurityConfigurerAdapter
- Ré-implémentation de la méthode configure
 - Dans l'exemple, définition de 2 utilisateurs, avec des mots de passe non chiffrés et des rôles

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    public void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication()
        .withUser("user").password("{noop}user").roles("USER").and()
        .withUser("admin").password("{noop}admin").roles("USER", "ADMIN");
    }
}
```

Contrairement à une configuration Spring Security classique

- Boot est capable de détecter le @Service UserDetailsService
- En revanche il lui faut toujours le bean PasswordEncoder

Comme pour une configuration Spring Security classique

- Il faut configurer les ressources accessibles, la page de connexion, ...
- Il faut activer les annotations @Secured, @PreAuthorize et @PostAuthorize

EXERCICE

Implémenter la sécurité, avec des utilisateurs issus d'une base de données

- Les utilisateurs peuvent voir les produits
- Les administrateurs peuvent voir et modifier les produits