



# SPRING

A decorative wavy line in light blue and white, running vertically along the left side of the slide.

# 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 (bien qu'on puisse l'utiliser pour)
  - 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

# PRÉSENTATION DE SPRING BOOT

- **Spring Boot** « remplace »
  - Les fichiers de configuration XML
  - Les classes de configuration
- Par une simple annotation `@EnableAutoConfiguration`
- Et ... et c'est tout !

# PRÉSENTATION DE SPRING BOOT

- **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	<code>fr.formation</code>
– Package model	<code>fr.formation.model</code>
– Package Repository	<code>fr.formation.repo</code>
– Package Controller	<code>fr.formation.controller</code>
– Package RestController	<code>fr.formation.api</code>
– ...	
  - S'il n'y a cette architecture de « package racine »
    - Ajouter l'annotation `@ComponentScan` pour scanner le ou les packages des composants **Spring**
    - Ajouter l'annotation `@EntityScan` pour scanner le ou les packages des classes modèles
    - Ajouter l'annotation `@EnableJpaRepositories` pour scanner le ou les packages des Repositories

# PRÉSENTATION DE SPRING BOOT

- **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



# PRÉSENTATION DE SPRING BOOT

- 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 !

# PRÉSENTATION DE SPRING BOOT

- 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 référer à la documentation officielle
  - <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter>



# APPLICATION

IMPLÉMENTATION DE SPRING BOOT

# APPLICATION 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>3.1.0</version>
</parent>
```

# APPLICATION SPRING BOOT

- Pour modifier la version de **JAVA** (en **JAVA 17** 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**

```
<properties>  
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  
  <java.version>17</java.version>  
</properties>
```

# APPLICATION SPRING BOOT

- On peut (idéalement) préciser une option de build **MAVEN**
  - En utilisant un plugin **Spring Boot**

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

# APPLICATION 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`

# APPLICATION SPRING BOOT

- 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
  - ...



# APPLICATION SPRING BOOT

- **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 !

A decorative wavy line in light blue and white, flowing vertically down the left side of the slide.

# INITIALIZR

STARTER DE STARTERS

# SPRING INITIALIZR

- Que serait **Spring 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**

# SPRING INITIALIZR

- 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
    - DevTools
    - Lombok
    - Web
    - JPA
    - PostgreSQL

# SPRING INITIALIZR

- Une fois prêt
  - Télécharger le projet
  - Désarchiver le projet
  - Importer le projet **MAVEN** sous Eclipse
- Et c'est prêt !

La même manipulation est possible directement depuis un IDE qui a le bon plugin

# EXERCICE

- Créer un nouveau projet avec **Spring Initializr**
  - Ne sélectionner aucune dépendance
- Tester l'exécution du projet



# 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 !");
    }
}
```





# LOGGING

BOOT & LOGGING

# SPRING BOOT LOGGING

- **Spring Boot** utilise **Log4j2** par défaut
  - Sa configuration est possible dans le fichier *application.properties*

```
# Log Package
```

```
logging.level.fr.formation.niveau.package = <NIVEAU>
```

```
# Log Hibernate
```

```
logging.level.org.hibernate.SQL = DEBUG
```

```
logging.level.org.hibernate.type.descriptor.sql.BasicBinder = TRACE
```

```
# Log Spring
```

```
logging.level.org.springframework = WARN
```

# SPRING BOOT LOGGING

- Niveau de journalisation de **Hibernate** plus avancé

```
logging.level.org.springframework.data = TRACE
logging.level.org.hibernate = DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder = TRACE
logging.level.org.hibernate.stat = DEBUG

spring.jpa.properties.hibernate.generate_statistics = true
spring.jpa.properties.hibernate.session.events.log.LOG_QUERIES_SLOWER_THAN_MS = 10
```



# DATA JPA

BOOT & DATA JPA

# SPRING BOOT DATA JPA

- On souhaite que notre application puisse utiliser une base de données
  - Avec **PostgreSQL** et **Hibernate**
  - Ajoutons le starter **data-jpa**

# SPRING BOOT DATA JPA

- On utilise **Hibernate** et **PostgreSQL**
  - Il va falloir préciser à **Spring** quelques informations de configuration
  - Dans le fichier *application.properties*

```
# Informations de connexion
spring.datasource.url = jdbc:postgresql://localhost:5432/db
spring.datasource.username = postgres
spring.datasource.password = root
```

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

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

# SPRING BOOT DATA JPA

- Ou on utilise **Hibernate** et **H2**
  - Il va falloir préciser à Spring quelques informations de configuration
  - Dans le fichier *application.properties*
    - Généralement on utilisera un script associé, que **Spring Boot** ira chercher dans les ressources
      - *schema.sql* et *data.sql*

```
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = none
```

# SPRING BOOT DATA JPA

- Par défaut, le pattern **OSIV** (**O**pen**S**ession**I**n**V**iew) est activé dans **Spring Boot**
  - Laissans ainsi une session Hibernate ouverte du début jusqu'à la fin de la génération d'une vue

```
# Désactiver OSIV  
spring.jpa.open-in-view = false
```



# EXERCICE

- Ajouter le starter **data-jpa** au projet
- Ajouter la dépendance « eshop-model »
- Configurer la source de données
- Reprendre les DAO (copier le package et les fichiers)
- Modifier l'application
  - Parcourir la liste des produits, et les afficher dans la console



# WEBMVC

BOOT & MVC

# SPRING BOOT WEBMVC

- On souhaite que notre application soit une application web
  - Ajoutons le starter **web**
- On souhaite utiliser **Thymeleaf** (moteur de template par défaut avec **Spring Boot**)
  - Ajoutons le starter **thymeleaf**

# SPRING BOOT WEBMVC

- Par défaut, **Spring Boot** va chercher dans ces répertoires
  - */resources/static* Pour toutes les ressources statiques (**CSS, JS, Images, ...**)
  - */resources/templates* Pour les templates (de **Thymeleaf** par exemple)

```
# Désactiver le cache Thymeleaf  
spring.thymeleaf.cache = false
```

# 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)



# SECURITY

BOOT & SECURITY

# SPRING BOOT SECURITY

- L'API « produits » est visible par défaut, par tout le monde !
  - Ajoutons le starter **security**



# SPRING BOOT SECURITY

- Par défaut, **Spring Boot** va rendre l'accès uniquement sur authentification
  - Nom d'utilisateur **user**
  - Mot de passe *auto-généré, disponible dans la console au démarrage*
- Possible de préciser un mot de passe (et un nom, et un rôle) par défaut

```
spring.security.user.password = Not24Get  
spring.security.user.roles = ADMIN
```

# SPRING BOOT SECURITY

- Pour la suite de la configuration de la sécurité, voir le cours **Spring Security**

# EXERCICE

- Implémenter une sécurité basique



# ACTUATOR

BOOT & MONITORING

# SPRING BOOT ACTUATOR

- On souhaite monitorer l'application
  - Ajoutons le starter **actuator**
- Des nouveaux points d'accès sont disponibles
  - /actuator
  - /actuator/health

<https://docs.spring.io/spring-boot/docs/2.6.3/reference/html/actuator.html#actuator.endpoints>

# SPRING BOOT ACTUATOR

- Pour activer de nouveaux endpoints

```
management.endpoints.web.exposure.include = health,info,beans
```

- Pour activer tous les endpoints

```
management.endpoints.web.exposure.include = *
```

- Pour désactiver des endpoints

```
management.endpoints.web.exposure.exclude = beans
```

# SPRING BOOT ACTUATOR

- Cas particulier pour activer le shutdown

```
management.endpoint.shutdown.enabled = true
```

# SPRING BOOT ACTUATOR

- Pour changer le point d'accès de **Actuator**

```
management.endpoints.web.base-path = /moniteur
```

- Pour changer le point d'accès d'un « module » **Actuator**

```
management.endpoints.web.path-mapping.<module> = /ressource
```

```
management.endpoints.web.path-mapping.health = /san/te
```

- Pour changer le port de **Actuator**

```
management.server.port = 9001
```



# SPRING BOOT ACTUATOR

- Pour activer les traces **HTTP**, activer le endpoint « httptrace »

```
management.endpoints.web.exposure.include = httptrace ...
```

- Et ajouter un bean `HttpTraceRepository`

```
@Bean  
public HttpTraceRepository httpTraceRepository() {  
    return new InMemoryHttpTraceRepository();  
}
```

# SPRING BOOT ACTUATOR

- Possible d'ajouter une ressource personnalisée
  - @ReadOperation **HTTP** GET
  - @WriteOperation **HTTP** POST
  - @DeleteOperation **HTTP** DELETE
  - @Selector Paramètre PathVariable

# SPRING BOOT ACTUATOR

```
@Component
@Endpoint(id = "custom")
public class CustomActuatorEndpoint {
    @ReadOperation //GET
    public String demo() {
        return "Demo";
    }

    @ReadOperation //GET
    public Produit demoWithParam(@Selector int id) {
        Produit produit = new Produit();
        produit.setId(id);

        return produit;
    }

    @WriteOperation //POST
    public String postOperation(@Selector String name) {
        return "Write " + name;
    }

    @DeleteOperation //DELETE
    public void deleteOperation(@Selector String name) {
        // DELETE
    }
}
```