Spring & Spring boot

LES FONDAMENTAUX

Frank MARSHALL
frank@readresolve.io

Programme

Introduction

IoC – Injection de dépendances

Les composants et *beans* Spring

Les couches d'une application Spring

Spring *classique* vs Spring boot

Outils et structure d'un projet Spring boot

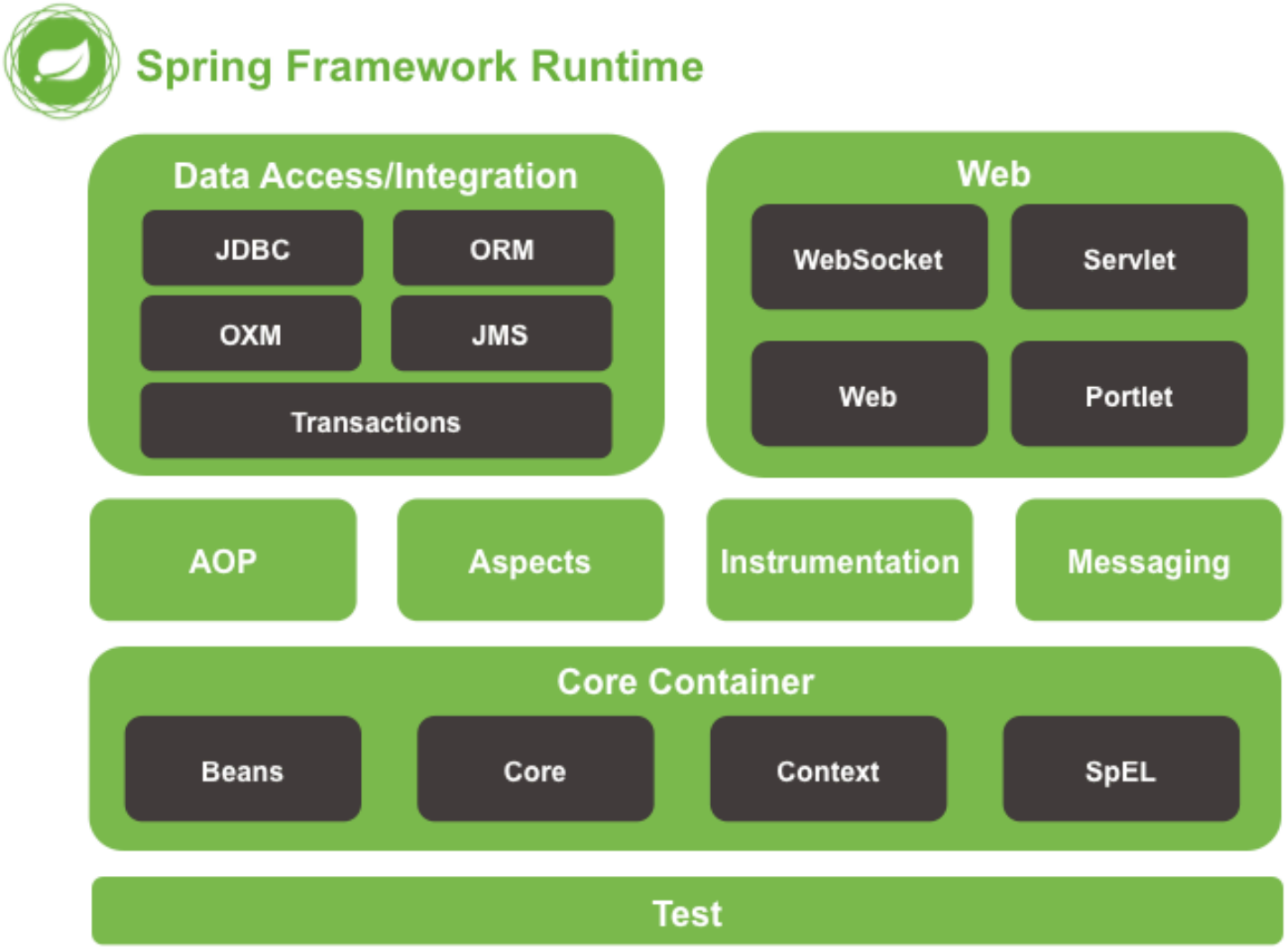
Les objectifs de Spring

Limiter le code technique et aider le développeur à se concentrer sur le fonctionnel, la valeur ajoutée de l'application

- Ne plus se préoccuper des transactions
- Ne plus se préoccuper des APIs Servlet, JMS…

Limiter la configuration en favorisant les conventions et l'usage des annotations

Les modules Spring



L’injection de dépendances (ex-IoC)

Limiter le couplage entre les nombreux composants d’une application

Usage des interfaces mais pas que… il faut toujours faire un new et dépendre explicitement de l’implémentation dans le code

On peut limiter les « dégâts » avec des design patterns tel Factory

Spring se propose d’implémenter ces patterns pour le développeur et de fournir des mécanismes simples pour obtenir des instances (composants et beans)

Les composants

Un composant est une classe qu'on déclare comme candidate à l'auto-détection par Spring

Un @Component a des spécialisations en fonction de son rôle

- @Controller / @RestController : servir des requêtes HTTP
- @Repository : recherche, récupération et stockage d'objets
- @Service : une interface ou façade sans état entre les controllers et les repositories ou d'autres services (API externes par ex.)

@Bean

Permet de déclarer des méthodes de fabrique de bean Spring répondant à des besoins spécifiques (typiquement de la configuration)

Les @Bean sont des singletons, au même titre que les @Component

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
```

Injection

L’injection est possible seulement dans le contexte Spring, autrement dit d’objets connus et gérés par Spring

@Autowired est l’annotation qui permet d’injecter un bean géré par Spring

Au niveau du champ ou du constructeur, la bonne pratique étant d’injecter au niveau du constructeur, ce qui permet de déclarer le champ **final**

- Éviter des réassignations accidentelles
- Le bean est disponible dans le constructeur

Exemple

```
@Service
public class LoginService {

    private final UserRepository repo;

    @Autowired // Optional, single constructor
    public LoginService(UserRepository repo){
        this.repo = repo;
    }
    // …
}
```

@Qualifier(”lowStrength”)

Permet de récupérer un bean par son nom, par défaut le nom d’un @Component ou d’un @Bean est le nom de la classe ou de la méthode

Au niveau du champ ou du paramètre du constructeur

```
@Bean
public PasswordEncoder lowStrength() {
    return new BCryptPasswordEncoder(4);
}
@Bean
public PasswordEncoder highStrength() {
    return new BCryptPasswordEncoder(31);
}
```

Les couches et leurs responsabilités

CONTROLLER

SERVICE
Interface
+ impl

REPOSITORY
Interface
+ impl

Exceptions
Validations
Redirections

Facade
Assembler

Data access

DTO (Data Transfer Object)

Design pattern répondant à des problématiques de performance

Propose de ne transporter que les données utiles pour répondre à une requête et de limiter les appels à des interfaces distances (navigateur/controllers, repository/bdd)

POJOs qui représente une « vue » des entités, évitent de récupérer tout un graphe d'objets

Note : les règles de validation peuvent être appliquées sur les DTO, rendant plus lisibles les entités

Les couches et les données

CONTROLLER

SERVICE

REPOSITORY

JSON
DTO (I/O)

DTO
Entities

Entities
DTO

Spring boot


SPRING FRAMEWORK
imgflip.com
SPRING BOOT
imgflip.com

Les objectifs de Spring boot

Limiter encore plus la configuration ou plutôt une auto-configuration (scan automatique des packages)

Faciliter la gestion des dépendances (avec les starters)

Un conteneur embarqué (plus de serveur à installer et configurer, plus de web.xml)

Un grand nombre de fonctionnalités exposées par défaut

Une application Spring boot

```
@SpringBootApplication
public class Application extends      SpringBootServletInitializer {

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

}
```

… ou presque

Un fichier de propriétés (.properties ou .yml) pour notamment y paramétrer le data source

D'éventuelles configurations spécifiques (notamment la sécurité)

Mais **@SpringBootApplication** est lui-même un composant de configuration, on peut y déclarer des classes internes de configuration ou directement des méthodes @Bean

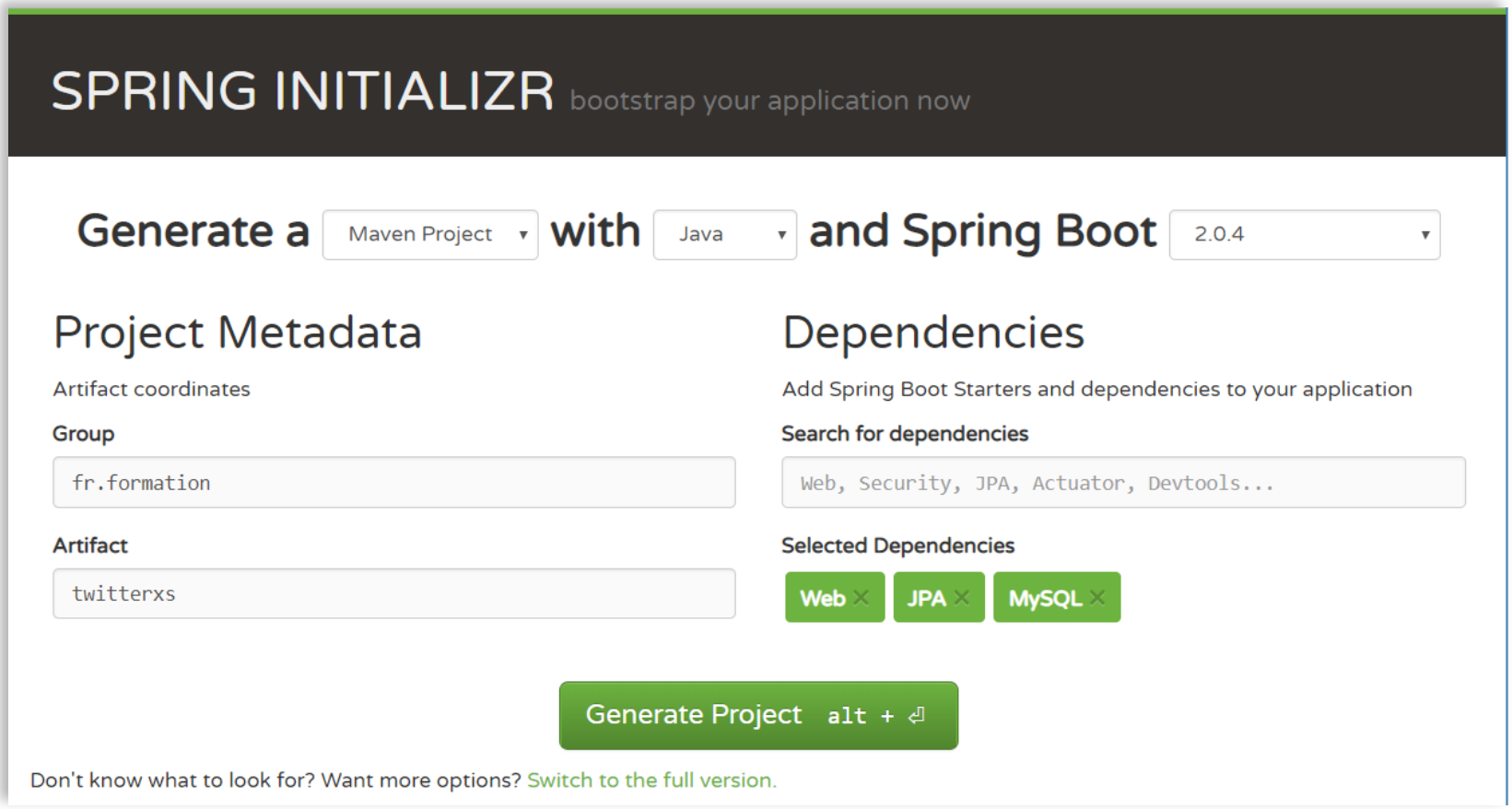
Un pom.xml Spring boot

```
<project>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.4.RELEASE</version>
		<relativePath />
	</parent>
	<dependencies>
		<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
	</dependencies>
</project>
```

application.properties minimaliste

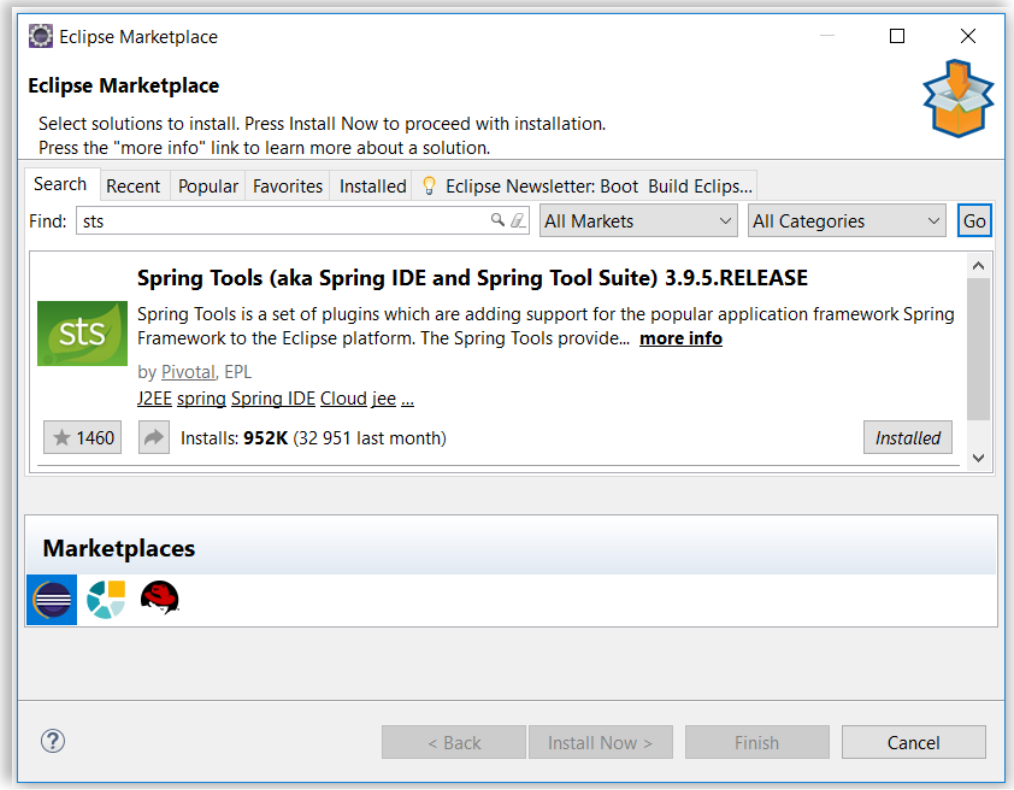
```
server.port=8081
server.servlet.context-path=/twitterxs

spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/twitterxs?useSSL=false
spring.datasource.username=root
spring.datasource.password=

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

Spring boot Initializr



Spring Tools Suite

Une distribution Eclipse avec STS embarquée ou bien ajout du plugin depuis la Marketplace



Spring Tools Suite

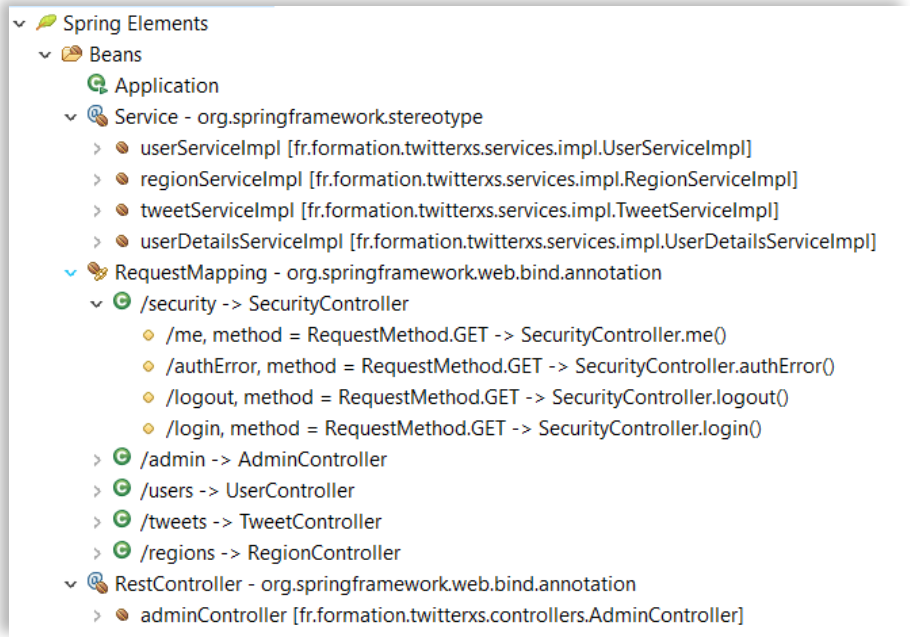
Un outils d'aide à la création et à la maintenance d'un projet Spring (boot ou pas)

Créer un projet Initializr depuis Eclipse

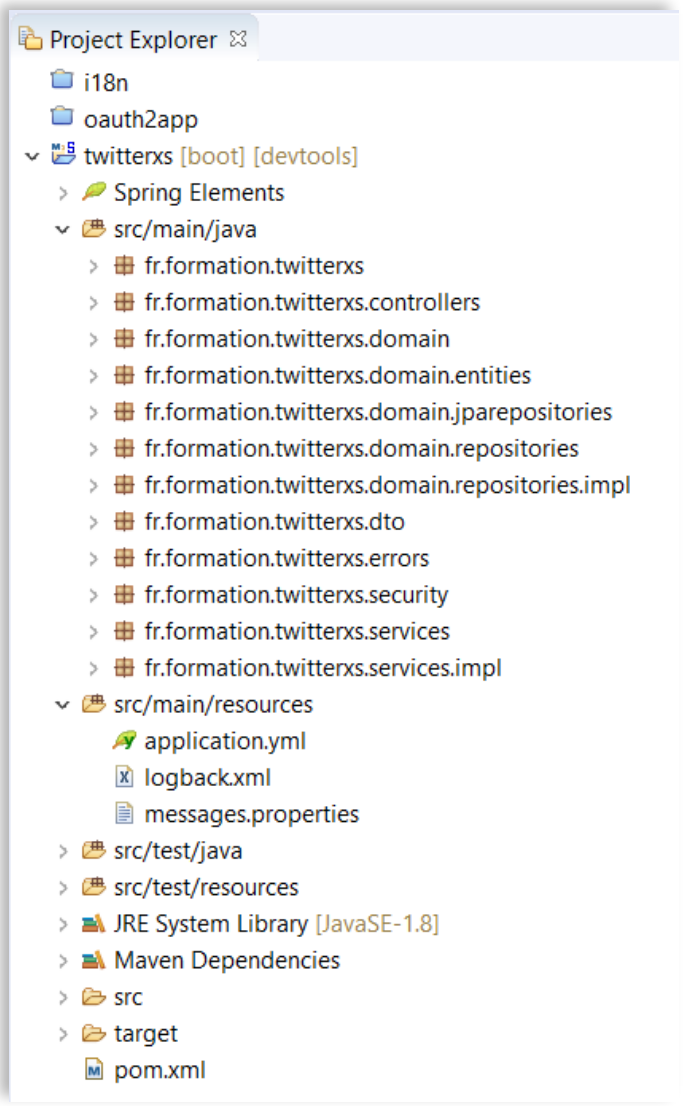
Règles de validation

Un explorateur de beans et des request mapping

Aide à la saisie avec l'auto-complétion



Structure d’un projet Spring boot



Bibliographie