Architecture en couches orientée services web

Frontal (JS) =============> Web Service Java (REST) ========> Bdd MariaDb ou sys. de fichiers

Web Service Java (REST) : Application web (sans ihm) qui expose un ensemble de méthodes

L’application web doit être deloyée sur un serveur d’applications (Apache **Tomcat**, Oracle **GlassFish**, Redhat **Wildfly**, Redhat **JBoss**, Oracle **WebLogic**, IBM **WebSphere**)

**Empaquetage d’une application Java :**

Client lourd ou bibliothèque : .jar (Java ARchive)  
Application web : .war (Web ARchive)  
Application d’entreprise : .ear(Entreprise ARchive) = ensemble de war + jar

**Déploiement** :

Créer l’appli et l’empaqueter => .war  
puis déployer le .war sur un serveur d’applications Java EE

Option 2 : on peut créer un FAT JAR (jar embarquant le serveur d’applications)

**Spring =** galaxie de plusieurs frameworks (https://spring.io/projects)

Spring Framework (Core, IoC) = brique de base représentant un conteneur d’objets  
 l’application instancie au démarrage des objets qu’elle met à disposition dans le conteneur d’objets  
  
Déclaration d’un objet à prendre en compte dans le conteneur : @Component, @Service, @Respository, @Controller, @RestController, méthode avec @Bean  
  
Singleton (1 seul objet) créé par défaut le conteneur  
  
Portée d’un objet : singleton, prototype, request, session, ...

@Component  
public class Product {

private String name;

}

@Service

public class ProductService {

@Autowired // injection d’un objet depuis le conteneur Spring

private Product p;

}

**Spring Web :** brique permettant de développer des applications web/services web

@RestController  
public class ProductController {

@GetMapping(value=“/products”, produces=”application/json”)  
 public List<Product> findAll(){  
 …...

}

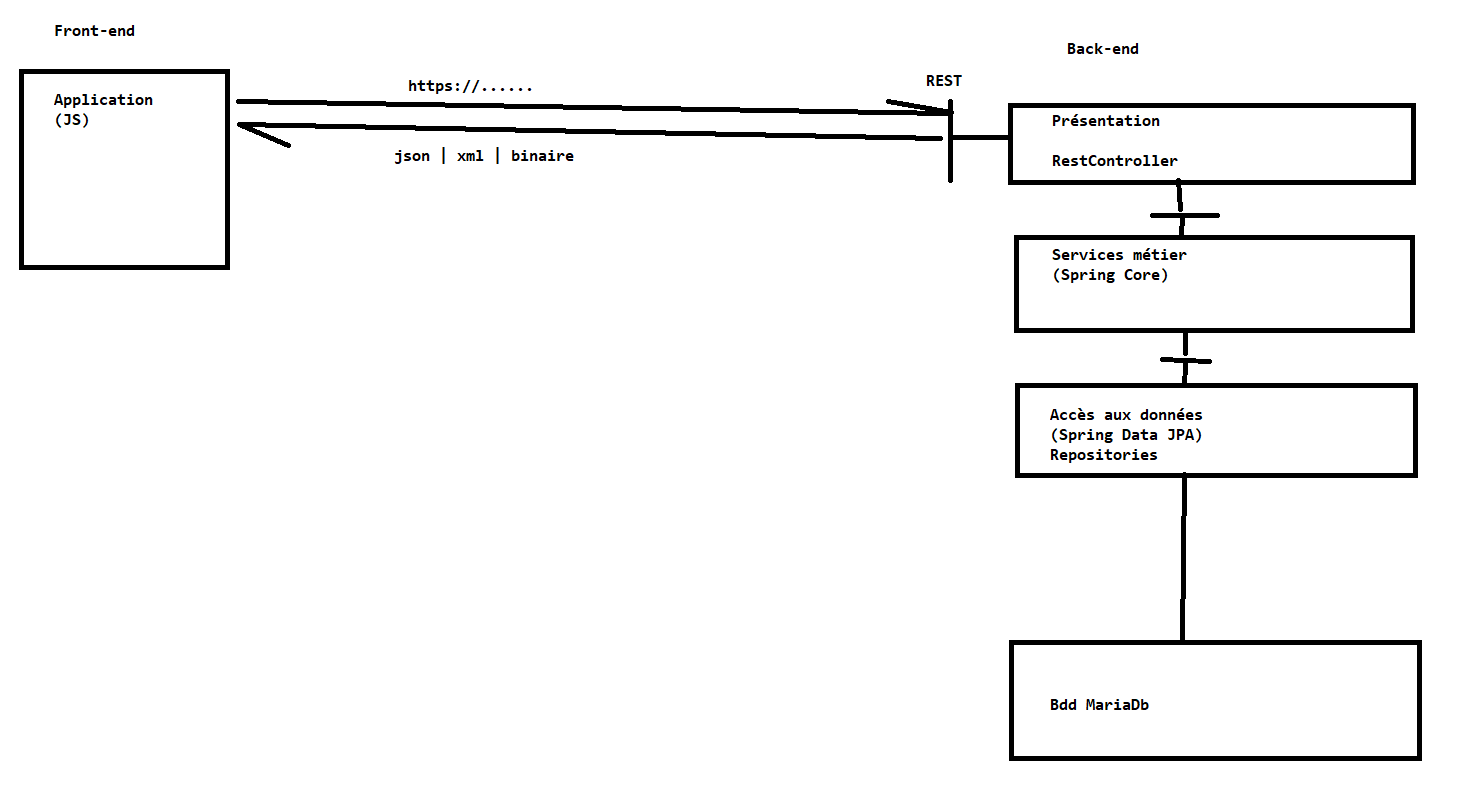
}

**Spring Data JPA : Api de persistance + surcouche Spring (moteur ORM : Hibernate)**  
 ORM : framework de mapping relationnel objet. Il permet de piloter le schéma de la bdd depuis le code et génère les req sql associées

**Spring Security** : gestion de la sécurité dans une application

**Spring Boot** : framework permettant de faciliter la configuration et le déploiement d’une application Spring

….



Un projet Spring Boot permet de créer une application déployable :   
briques sélectionnées : Spring Web, Spring Data JPA + driver MariaDb

On peut lancer le projet avec un fichier de configuration différent en paramétrant comme argument :

--spring.config.location=classpath:**application-dev.properties**

**--------------------------**

Entité : classe mappée avec une table en Bdd  
 unmapped class => @Entity

Une entité doit obligatoirement avoir une clé primaire (@Id) + constructeur sans paramètres

**Annotations les plus courantes :**

@Table : personnaliser le nom ou le schéma de la table  
@Id : clé primaire  
@IdClass ou @EmbeddedId : clé primaire composée  
@Column : personnaliser une colonne

Relations :  
1 à n / n à 1 :

Une catégorie possède plusieurs produits  
  
t\_categories ( id, intitule)  
t\_produits (id, nom, category\_id)

© Category  
© Product

Dans les associations, on a 2 types de chargement :   
- immédiat (Eager), par défaut dans OneToOne, ManyToOne  
- tardif (Lazy), par défaut OneToMany, ManyToMany

------------

Relation n à n (ManyToMany)

Product possède plusieurs fournisseurs  
1 fournisseur livre plusieurs produits  
  
t\_products(id, description, price, version)  
t\_suppliers(id, version, name)  
t\_products\_suppliers(product\_id, supplier\_id)

------------------

Relation 1 à 1

1 manager encadre un joueur  
1 joueur est encadré par 1 et unique manager

Plusieurs représentations en BDD d’une relation OneToOne :   
- 1 seule table qui regroupe les infos du manager et du joueur  
 @Embeddable sur la classe qu’on souhaite intégrer (pas de table créée)  
 @Embedded au niveau de l’entité englobante (attention au nommage des colonnes)

* Association par clé primaire (même clé dans les 2 tables)

T\_managers t\_players  
1 Man1 1 player1

* Association par clé étrangère avec une contrainte d’unicité :

T\_managers t\_players  
id|name id|name|manager\_id (unique)  
1 Man1 23 play2 1  
 65 play5 1 (car il y a une contrainte d’unicité)