

25/04/2023

Version 4



SPRING

JÉRÉMY PERROUULT



SPRING JPA

SPRING & HIBERNATE

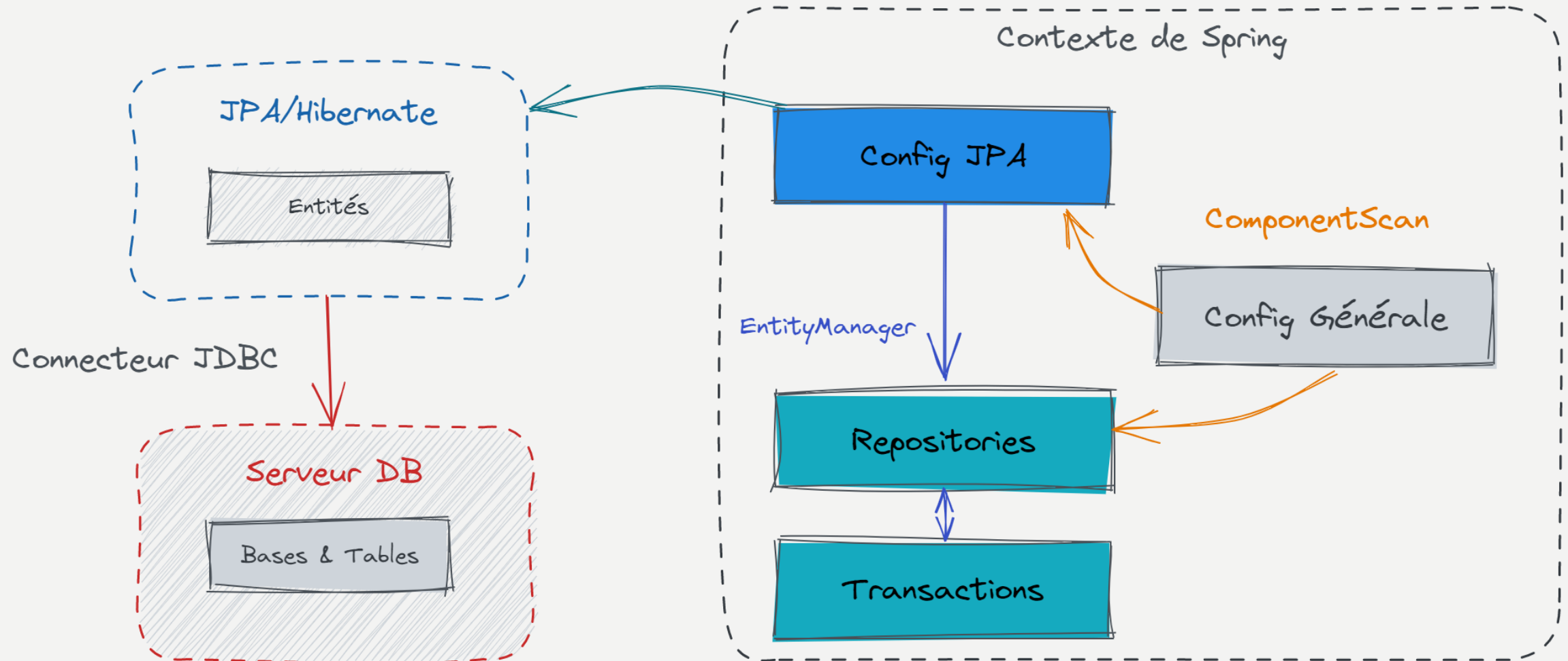
PRÉSENTATION SPRING JPA

- **Spring** va prendre le pas
 - Sur la déclaration JNDI du **DataSource**
 - Sur la configuration **JPA** (*persistence.xml*)
- Les objets **Repository** sont annotées de `@Repository`
- Les transactions sont gérées par service *@Transactional* (**AOP**)
- **Spring** ouvre et ferme `EntityManager` et `EntityManagerFactory`

PRÉSENTATION SPRING JPA

- Dépendances
 - **spring-orm**
 - Brique **ORM**
 - **spring-tx**
 - Gestionnaire de Transactions
 - **commons-dbcp2** OU **HikariCP**
 - L'un des deux pour configurer un gestionnaire de pool de connexions

PRÉSENTATION SPRING JPA





CONFIGURATION

CONFIGURATION DE SPRING JPA

CONFIGURATION

- Utilisation de 3 beans
 - DataSource
 - EntityManagerFactory
 - Utilise la référence du **DataSource**
 - Précise les options comme le **Provider** et les options du **Provider**
 - TransactionManager
 - Utilise la référence de l'EntityManagerFactory
- Activation des annotations `@Transactional`
 - Utilise la référence du TransactionManager

CONFIGURATION

@Bean

```
public DataSource dataSource() {  
    BasicDataSource dataSource = new BasicDataSource();  
  
    dataSource.setDriverClassName("org.postgresql.Driver");  
    dataSource.setUrl("jdbc:postgresql://localhost:5432/eshop");  
    dataSource.setUsername("postgres");  
    dataSource.setPassword("root");  
    dataSource.setMaxTotal(10);  
  
    return dataSource;  
}
```


CONFIGURATION

- Création de l'**EntityManagerFactory** géré par **Spring**
 - En lui passant la **DataSource** gérée par **Spring**
 - En lui précisant le **provider** (implémentation **JPA**, **Hibernate** dans notre cas)
 - En lui précisant les options du **provider** (les options de **Hibernate**)

CONFIGURATION

@Bean

```
public LocalContainerEntityManagerFactoryBean entityManagerFactory(DataSource dataSource) {  
    LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();  
    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();  
    Properties properties = new Properties();  
  
    properties.setProperty("hibernate.hbm2ddl.auto", "update");  
    properties.setProperty("hibernate.show_sql", "true");  
    properties.setProperty("hibernate.format_sql", "false");  
  
    emf.setDataSource(dataSource);  
    emf.setPackagesToScan("fr.formation.model");  
    emf.setJpaVendorAdapter(vendorAdapter);  
    emf.setJpaProperties(properties);  
  
    return emf;  
}
```

CONFIGURATION

- Création du `JpaTransactionManager` géré par **Spring**
 - En lui passant l' `EntityManagerFactory` géré par **Spring**
- Activation des annotations `@Transactional`
 - Qu'on utilisera sur nos `Repository`
 - Permet d'écouter toutes les méthodes de nos `Repository` grâce à **AOP**

CONFIGURATION

- Annoter la classe de configuration de `@EnableTransactionManagement`
 - Pour activer l'annotation `@Transactional`

`@Bean`

```
public JpaTransactionManager transactionManager(EntityManagerFactory emf) {  
    JpaTransactionManager transactionManager = new JpaTransactionManager();  
  
    transactionManager.setEntityManagerFactory(emf);  
    return transactionManager;  
}
```

CONFIGURATION

- Activation de la traduction des **Exceptions Spring DAO**
 - Permet de traduire les Exceptions levées pendant le traitement de *Persistence* (HibernateException, PersistenceException, ...) en Exception *DataAccessException*
 - Fonctionne sur les classes annotées de `@Repository`

```
@Bean
public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
    return new PersistenceExceptionTranslationPostProcessor();
}
```

UTILISATION

- Déclaration du **bean**
 - `@Repository` (ne pas oublier de scanner les packages !)
- Déclaration de l'utilisation des transactions **Spring**
 - `@Transactional` sur la classe ou sur les méthodes concernées
- Récupération de `EntityManager` (injection de la dépendance)
 - `@PersistenceContext`

UTILISATION

```
@Repository
public class ProduitRepositorySpring implements IProduitRepository {
    @PersistenceContext
    private EntityManager em;

    @Override
    public List<Produit> findAll() {
        return this.em
            .createQuery("select p from Produit p", Produit.class)
            .getResultList();
    }

    @Override
    @Transactional
    public Produit save(Produit entity) {
        this.em.persist(entity);
        return entity;
    }
}
```

EXERCICE

- Implémenter **SPRING JPA**
 - D'abord sans fichier de configuration *.properties*
 - Puis avec un fichier *data-source.local.properties*
 - Fichier qu'on pourra ignorer dans **GIT**