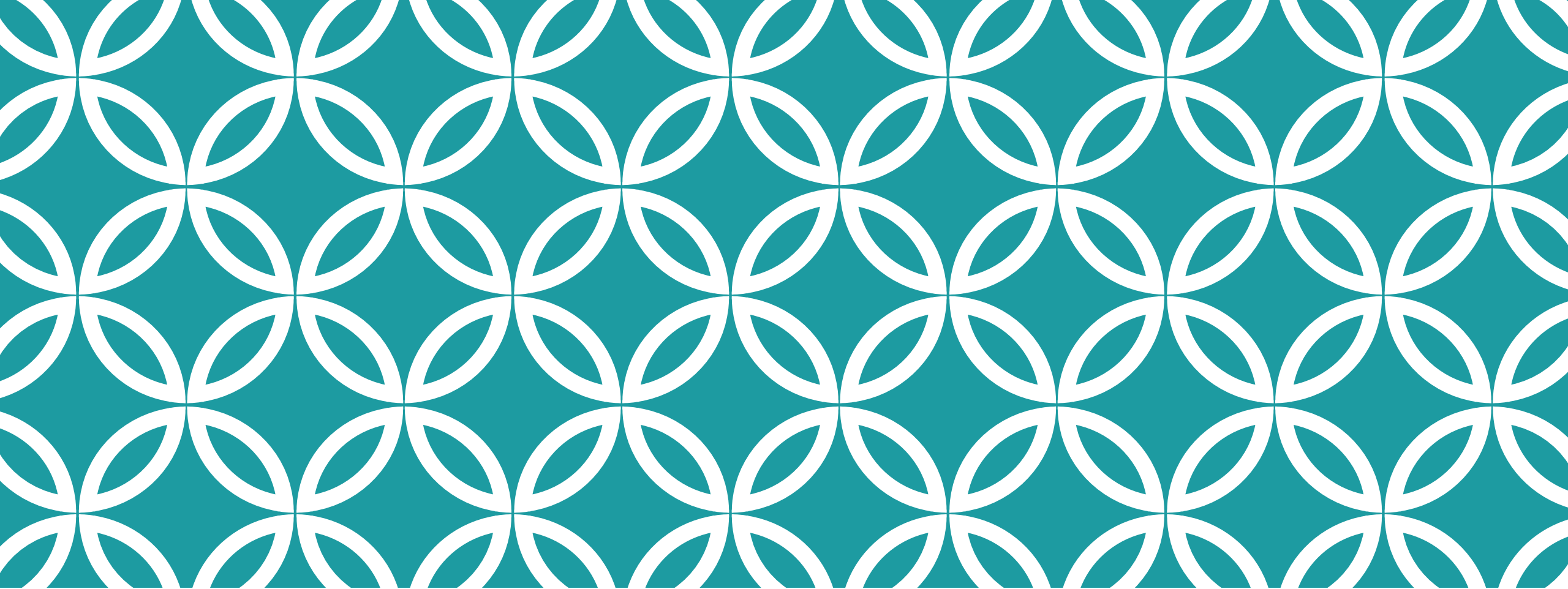


SPRING



SPRING JPA

Spring & Hibernate
Combot gagnant !

PRÉSENTATION SPRING JPA

Spring va prendre le pas

- Sur la déclaration JNDI du DataSource
- Sur la configuration JPA (*persistence.xml*)
- Configuration dans "application-context.xml"

Les objets DAO deviennent des *@Repository*

Les transactions sont gérées par service *@Transactional* (AOP)

Spring gère l'ouverture et la fermeture de *EntityManager* et *EntityManagerFactory*

CONFIGURATION

Dépendances

- **spring-orm**
- **spring-tx**

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 id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <!-- ... -->
</bean>

<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <!-- ... -->
</bean>

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```

CONFIGURATION

```
<!-- On crée le DataSource -->
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/eshop" />
  <property name="username" value="root" />
  <property name="password" value="" />
  <property name="maxTotal" value="10" />
</bean>
```

Création de la **DataSource** gérée par Spring

- Avec le pilote à utiliser
- L'URL de connexion à la base de données
- Les identifiants de connexion
- Le maximum de connexions simultanées actives

CONFIGURATION

```
@Bean
public BasicDataSource dataSource() {
    BasicDataSource dataSource = new BasicDataSource();

    dataSource.setDriverClassName("com.mysql.jdbc.Driver");
    dataSource.setUrl("jdbc:mysql://localhost:3306/eshop");
    dataSource.setUsername("root");
    dataSource.setPassword("");
    dataSource.setMaxTotal(10);

    return dataSource;
}
```


CONFIGURATION

```
<!-- On crée un entityManagerFactory local à partir de la dataSource -->
<bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="packagesToScan" value="fr.formation.model" />

  <!-- On précise le provider ... -->
  <property name="jpaVendorAdapter">
    <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter" />
  </property>

  <!-- On précise les propriétés ... -->
  <property name="jpaProperties">
    <props>
      <prop key="hibernate.hbm2ddl.auto">update</prop>
      <prop key="hibernate.dialect">org.hibernate.dialect.MySQL5InnoDBDialect</prop>
      <prop key="hibernate.show_sql">true</prop>
      <prop key="hibernate.format_sql">true</prop>
    </props>
  </property>
</bean>
```

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(BasicDataSource dataSource) {
    LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();
    JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();

    emf.setDataSource(dataSource);
    emf.setPackagesToScan("fr.formation.model");
    emf.setJpaVendorAdapter(vendorAdapter);
    emf.setJpaProperties(this.hibernateProperties());

    return emf;
}
```

CONFIGURATION

```
private Properties hibernateProperties() {  
    Properties properties = new Properties();  
  
    properties.setProperty("hibernate.hbm2ddl.auto", "update");  
    properties.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5InnoDBDialect");  
    properties.setProperty("hibernate.show_sql", "true");  
    properties.setProperty("hibernate.format_sql", "true");  
  
    return properties;  
}
```

CONFIGURATION

```
<!-- On crée le transactionManager pour JPA avec entityManagerFactory -->
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<!-- On active les annotations @Transactional avec transactionManager -->
<tx:annotation-driven transaction-manager="transactionManager" />
```

Création du **JpaTransactionManager** géré par Spring

- En lui passant l'**EntityManagerFactory** géré par Spring

Activation des annotations **@Transactional**

- Qu'on utilisera sur nos **DAO**
- Permet d'écouter toutes les méthodes de nos **DAO** grâce à *AOP*

CONFIGURATION

```
@Bean
public JpaTransactionManager transactionManager(EntityManagerFactory emf) {
    JpaTransactionManager transactionManager = new JpaTransactionManager();

    transactionManager.setEntityManagerFactory(emf);
    return transactionManager;
}
```

Annoter la classe de configuration de **@EnableTransactionManagement**

- Pour activer l'annotation **@Transactional**

CONFIGURATION

Activation de la gestion 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**

Configuration XML

```
<!-- On active la translation d'exception -->  
<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor" />
```

Configuration Java

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

UTILISATION

Déclaration du *bean* **DAO**

- **@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
@Transactional
public class DAOProduitRepository implements IDAOProduit {
    @PersistenceContext
    private EntityManager em;

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

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

    //...
}
```


EXERCICE

Créer un nouveau projet « eshop-jpa » (Maven)

- Référencer « eshop-model » (référencer les classes métier)
- Transférer toutes les DAO dans ce nouveau projet
- Implémenter et paramétrer Spring JPA
- Ajouter une nouvelle classes **DAOProduitRepository**
 - Cette **DAO** est configurée avec Spring

Modifier le projet « eshop-java »

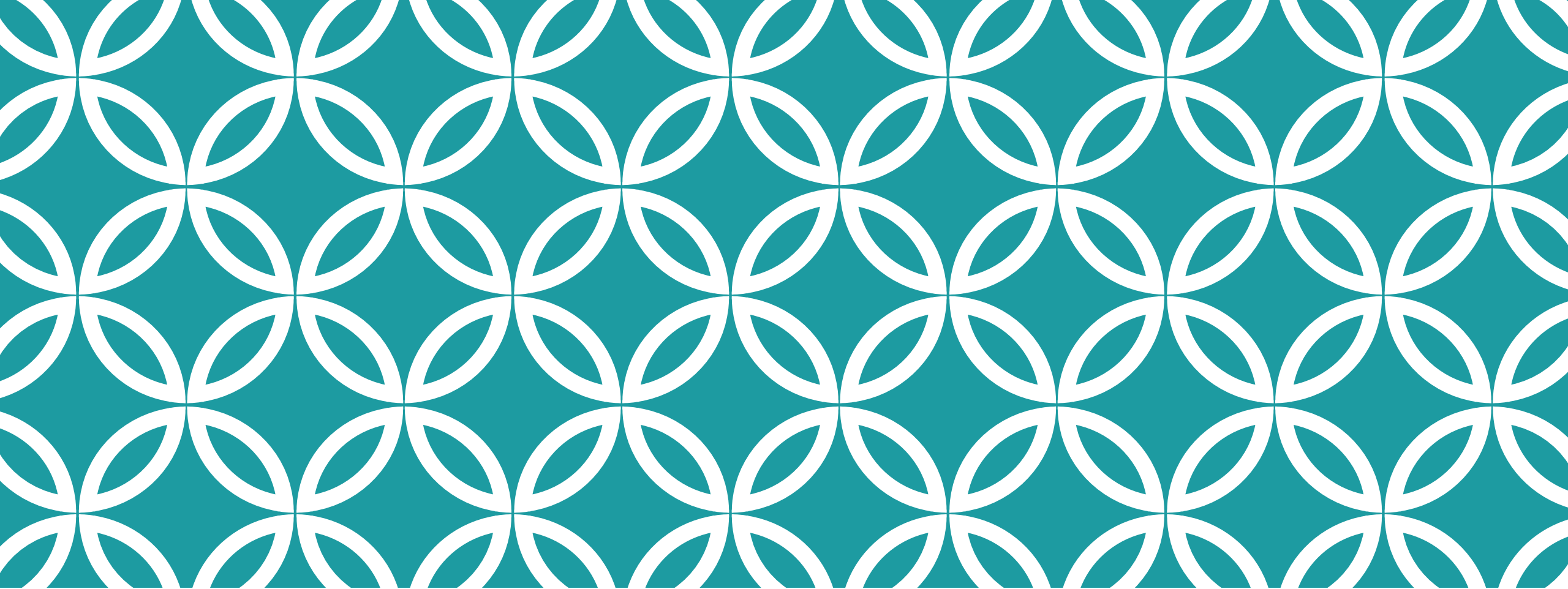
- Ajouter la dépendance au projet « shop-jpa »
- Ajouter la dépendance **common-dbcp2**
- Configurer Spring JPA dans ce projet
- Supprimer la configuration Hibernate faite jusqu'ici

Modifier le programme **Generator**

- Il doit fonctionner avec les DAO Spring JPA

Modifier le programme **App**

- Lister les produits et leur fournisseur, avec les DAO Spring JPA



PROPRIÉTÉS

Fichier de configuration
properties

CONFIGURATION AVEC PROPERTIES

Mise en place d'une configuration avec un fichier .properties (clé = valeur)

- Fichier main/resources/data-source.properties

```
sql.url = jdbc:mysql://localhost:3306/eshop
```

CONFIGURATION AVEC PROPERTIES

On précise à Spring d'aller récupérer ce fichier properties

```
<context:property-placeholder location="classpath:data-source.properties" />
```

On manipule les propriétés avec une SpEL (Spring Expression Language) \${ }

- Dans le fichier de configuration XML

```
<bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource">  
  <property name="url" value="${sql.url}" />  
</bean>
```

- Sur l'attribut d'une classe

```
@Value("${sql.url}")  
private String url;
```

CONFIGURATION AVEC PROPERTIES

On précise à Spring le ou les fichiers properties avec l'annotation `@PropertySource`

Puis on injecte les propriétés dans un objet de type **Environment**

```
@Configuration
@PropertySource("classpath:data-source.properties")
public class AppConfig {
    @Autowired
    private Environment env;

    @Bean("dataSource")
    public BasicDataSource dataSource() {
        BasicDataSource dataSource = new BasicDataSource();
        dataSource.setUrl(env.getProperty("sql.url"));
        return dataSource;
    }
}
```

EXERCICE

Mettre en place les éléments de configuration de la source de données dans un fichier de .properties

- URL base de données
- Nom d'utilisateur
- Mot de passe
- Nombre de connexions simultanées