Installation

Ce fichier expose la procédure d'installation

Installation en production

Fichiers fournis dans le fichier env.zip :

```
Fichiers de configuration de l'application
    application.yml
    application-prod.yml
Fichiers Docker :
    Dockerfile-postgres-moissoncatalogue
    elasticsearch.yml
    postgresql.yml
    swagger-editor.yml
    V0__init_user_role_database_for_docker.sql
Fichiers d'initialisation de la base
    V0__init_user_role_database.sql
    V0__init_user_role_database_for_docker.sql
Fichiers de création des tables et séquences .sql au format liquibase
    V1_0__premiere_migration_des_tables_de_base.sql
    V1_1__create_tva_table.sql
    V1_2__create_techno_cible_table.sql
    V1_3__create_offre_table.sql
    V1_4__create_discipline_table.sql
    V1_5__create_niveau_table.sql
    V1_6__create_condition_table.sql
    V1_7__create_article_papier_table.sql
    V1_8__create_article_numerique_table.sql
    V1_9__create_lep_table.sql
    V1_10__create_disponibilite_table.sql
    V1_11__create_licence_table.sql
```

Le fichier V0__init_user_role_database_for_docker.sql est fourni deux fois, par commodité, dans deux répertoires différents : docker et initialisation_de_la_base_de_donnees

Avant de lancer l'application, il faut **impérativement** configurer l'application et créer la base de données et les rôles

```
1 - Configuration du fichier de propriétés
2 . Création de la base et des scripts
3 . Créationde l'instance ElaticSearc
4 . Déploiement et urtilisation  de l'application
```

Répertoire de configuration

Comportement par défaut

Les fichiers de configuration sont fournis indépendamment du fichier ".jar", il faudra changer les paramètres du fichier spécifique à la production (voir ci-après) :

```
Fichiers de configuation:
    application.yml
    application-prod.yml
```

Une fois les paramètres fixer il faut le déposer dans un répertoire qui peut être soit :

1 - Dans un répertoire "config/" créé au meme niveau que le répertoire de base de l'application : 2 - Dans le répertoire de base de l'application.

Le répertoire de base est le répertoire de lacement du fichier ".jar".

Si les fichiers sont créés dans les deux répertoires, le répertoire config/ (1) à la précédence sur le répertoire de base (2), ce sont donc les paramètres du fichier du répertoire config/ qui seront pris en compte

Si répertoire config:

```
répertoire de base:
    /chemin/de/repertoire/de/base/
répertoire de config:
    /chemin/de/repertoire/de/base/config
```

Remarque : si aucun des deux répertoires n'est créé, l'application est lancée en "localhost" sur le port "808"

Modification du comportement par défaut

Il est possible de surcharger le comportement par défaut en précisant le chemin du répertoire lors du lacement de l'application :

```
Dem manière absolue
    java -Dspring.config.location=/chemin/du/repertoire/config -jar
moissoncatalogue.jar --spring.profiles.active
    ou
    java -jar moissoncatalogue.jar
--spring.config.location=/chemin/du/repertoire/config

Ou relative par exemple
    java -Dspring.config.location=../config -jar moissoncatalogue.jar
--spring.profiles.active/ ou
    java -jar moissoncatalogue.jar --spring.config.location=../config
--spring.profiles.active

    java -Dspring.config.location=chemin/relatif/complexe/vers/le/repertoire/config
-jar moissoncatalogue.jar
    java -jar moissoncatalogue.jar
--spring.config.location==chemin/relatif/complexe/vers/le/repertoire/config
```

Paramètres du fichier de configuration application-prod.yml :

Base de données

Dans le cas d'utilisation avec une instance postgres Il faut modifier le nom d'hôte at éventuellement le port :

```
datasource:
  type: com.zaxxer.hikari.HikariDataSource
  # Domaine (nom d'hôte à modifier, par défaut : localhost) et port éventuellement
mais généralement c'est le port par défaut.
  # Ne pas modifier le nom de la base  : moissoncatalogue
  url: jdbc:postgresql://localhost:5432/moissoncatalogue
  # Ne pas modifier username: usercatalogue
  username: usercatalogue
  password: catalogue
```

Il est possible de changer le mot de passe au sein du fichier.

Si le mot de passe est changé, Il faut, le changer également dans le script d'initialisation de la base avant de le jour (voir explication dans création de la base de données ci-après) :

```
V0__init_user_role_database.sql
```

remplacer le mot de passe "catalogue" :

```
CREATE ROLE usercatalogue LOGIN NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE
NOREPLICATION PASSWORD 'catalogue';
```

par le NOUVEAU_MOT_DE_PASSE :

```
CREATE ROLE usercatalogue LOGIN NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE
NOREPLICATION PASSWORD 'NOUVEAU_MOT_DE_PASSE';
```

Remarque : Dans le cas d'utilisation avec postgres sous la forme de conteneur docker Il ne faut pas modifier le nom d'hôte, mais éventuellement le port pour le mettre en correspondance avec le fichier postgres.yaml :

```
Si dans le fichier postgresql.yml
    ports:
      - 5433:5432
Dans le fichier application-prod.yml dans datasource
    url: jdbc:postgresql://localhost:5433/moissoncatalogue
```

ElasticSearch

Dans le cas d'utilisation avec une instance, il faut modifier l'hôte le port et définir login et password :

```
# Hôte et port à modifier sans les scheme (http ou https)
uris: localhost:9200
# A modifier s'il y a une authentification ne pas dé-commenter sans authetification
dans ElasticSerch sinon l'application ne démarre pas.
# username: admin
# password: admin
```

Remarque : Dans le cas d'utilisation avec postgres sous la forme de conteneur docker Il ne faut pas modifier le nom d'hôte, mais éventuellement le port pour le mettre en correspondance avec le fichier elasticsearch.yml :

```
Si dans le fichier pelasticsearch.yml
    ports:
      - 9201:9200
Dans le fichier application-prod.yml
  elasticsearch:
    rest:
      uris: localhost:9201
```

Liquibase

```
liquibase:
  contexts: prod
  # Domaine (nom d'hôte à modifier par défaut : localhost).
  # Ne pas modifier le nom de la base  : moissoncatalogue
  url: jdbc:postgresql://localhost:5432/moissoncatalogue
```

Remarque : Dans le cas d'utilisation avec postgres sous la forme de conteneur docker Il ne faut pas modifier le nom d'hôte, mais éventuellement le port pour le mettre en correspondance avec le fichier postgresql.yaml :

```
Si dans le fichier postgresql.yml
    ports:
      - 5433:5432
Dans le fichier application-prod.yml dans liquibase
    url: jdbc:postgresql://localhost:5433/moissoncatalogue
```

Mail

Optionnel car non utilisé pour le moment :

```
mail:
  host: localhost
  port: 25
  username:
  password:
```

Création de la base de données.

Avec une instance postgres installée

Une instance de postgres est accessible.

Création de la base et des rôles.

Par défaut les tables sont créées avec Liquibase qui est une librairie open-source permettant de tracer et gérer les modifications d'une base de données. Liquibase est paramétré pour la mise en place des tables et séquences au premier démarrage de l'application, cependant avant de lancer l'application, il faut, cependant créer les roles et la base correspondante.

Puis en se connectant en root :

```
sudo -u postgres psql
```

Il faut jouer les scripts qui sont dans le fichier :

```
V0__init_user_role_database.sql est le fichier de création de la base de données
```

Il dit être possible de se connecter à la base créée :

```
    \connect moissoncatalogue
```

Création des tables automatique avec liquibase

L'application utilise Liquibase pour la création des tables. Ces dernières sont donc créées automatiquement lors du déploiement de l'application.

Remarque : Les scripts SQL sont fournis et situés dans le répertoire sql du fichier ".zip" et peuvent être utilisés tels quels pour générer l'ensemble des tables.

Pour les développeurs, les noms de fichier de scripts sont au format FlyWay et sont stockés dans le répertoire de recherche par défaut de Flyway bien que celui-ci n'est pas installé par défaut, main/resources/db/migration. La procédure d'installation et d'utilisation de Flyway est fournie à la fin du document.

Avec docker

Il faut que docker et docker-compose soient installés, voir l'adresse suivante pour les instructions :

```
https://docs.docker.com/compose/install/
```

Création du conteneur

Au préalable, il est préférable de créer un volume afin de conserver les données lors de l'arrêt du conteneur Il faut, dans ce cas, dé-commenter les lignes du fichier docker-compose postgresql.yml fourni et remplacer :

```
~/volumes/moissoncatalogue/postgresql/
```

Par le chemin du volume où il est souhaité de conserver les données.

Connexion au conteneur et création de la base de données

L'utilisateur "usercatalogue" a été créé lors de la création du conteneur.

Création du conteneur avec locale "fr"

Il faut créer le conteneur avec les bonnes locales, à partir de l'image officielle

```
docker build -t  postgres-moissoncatalogue:12.5 -f ../config/Dockerfile-postgres-
moissoncatalogue .
```

(le point à la fin de la commande doit être conservé)

Démarrer le conteneur

Après avoir remplacé les bons paramètres, il faut lancer la commande suivante dans le meme répertoire que ce fichier :

```
docker-compose -f postgresql.yml up -d
```

le paramètre -d permet de lancer l'instance de docker en background

Connexion au conteneur

Connexion au conteneur avec une console :

```
docker exec -it moissoncatalogue-postgresql  bash
```

Connexion à la base de données au sein du conteneur

```
psql -U usercatalogue
```

Jouer les scripts dans le fichier :

```
V0__init_user_role_database_for_docker.sql
```

Il est possible de laisser le conteneur ou vert ou le fermer avec exit (après s'être déconnecté de la base de données avec \q)

Arrêt du conteneur

Pour arrêter le conteneur :

```
docker-compose -f postgresql.yml down
```

Elasticsearch

Création du conteneur

Comme pour la base il est possible de créer un docker :

Il est préférable de créer un volume afin de conserver les données lors de l'arrêt du conteneur Il faut, dans ce cas, dé-commenter les lignes du fichier docker-compose elasticsearch.yml fourni et remplacer :

```
~/volumes/moissoncatalogue/elasticsearch/
```

par le chemin du volume

Après avoir créé le répertoire et modifier le propriétaire

```
sudo mkdir -p ~/volumes/moissoncatalogue/elasticsearch/
sudo chown -R 1000:1000 ~/volumes/moissoncatalogue/elasticsearch/
```

Connexion au conteneur

Un fichier docker-compose est fourni et après avoir remplacé les bons paramètres, il faut lancer la commande suivante dans le meme répertoire que ce fichier :

```
docker-compose -f elasticsearch.yml up -d
```

Le paramètre -d permet de lancer l'instance de docker en background

Arrêt au conteneur

Pour arrêter le conteneur :

```
docker-compose -f elasticsearch.yml down
```

Démarrage de l'application

```
java -Dspring.config.location=../config -jar moissoncatalogue.jar
```

Vérification de la base de données

Se connecter au conteneur postgres si non connecté

Se connecter à la base puis

```
\connect moissoncatalogue
```

Lister les tables

```
\dt ou \d
```

Rest api

Les endpoints des Apis sont fournis dans le contrat d'Api fourni.

Authentification préalable

Les Apis étant sécurisées il faut s'authentifier pour y accéder.

Authentification basique avec login et mot de passe (user et password)

Il existe deux utilisateurs qui permettent de s'identifier :

1. l'utilisateur "admin" avec le password "admin" par défaut qui possède les roles ROLE_USER et ROLE_ADMIN
2. l'utilisateur "user" avec le password "user" par défaut qui possède le role ROLE_USER

L'administrateur "admin" peut accéder aux apis en lecture écriture et suppression.

L'utilisateur "user" peut accéder aux apis en lecture seule.

Pour accéder aux apis il faut utiliser curl, postman ou insomnia designer

1 - Avec curl pour accéder il faut préciser l'"user" et le "password" :

```
Pour obtenir les informations sur l'utilisateur
curl -v http://admin:admin@localhost:8080/api/account
ou
curl -vu admin:admin http://localhost:8080/api/account

Pour obtenir la liste des arcticles numériques :
curl -vu admin:admin http://localhost:8080/api/article-numeriques
curl -vu user:user http://localhost:8080/api/article-numeriques
```

-v permet d'activer le mode verbose -vu étant équivalent à -v -u

2 - Avec Postman ou Insomnia il faut saisir les url en prenant soin de bien spécifier le verbe (GET, POST, PUT etc…)

Dans l'onglet authentication ou auth il faut saisir le login et le mot de passe

```
Le endpoint suivant retourne le login de l'utilisateur authentifié avec le verbe GET
http://user:passwortd@localhost:8080/api/authenticate
```

Authentification avec token JWT

L'api est livrée avec une authentification par token JWT.

Il faut dans un premier temps générer le token, avec l'endpoint http://user:user@localhost:8080/api/authenticate et le verbe POST

```
curl  -X POST -H 'Accept: application/json' -H 'Content-Type: application/json' --data
'{"username":"admin","password":"admin"}' http://localhost:8080/api/authenticate
le curl génère un token.

"id_token" :
"eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGgiOiJST0xFX0FETUlOLFJPTEVfVVNFUiIsImV4
cCI6MTYwOTAwMzc2M30.bh8fQMGXawP354wGS1qG_KxSCD1_7hmthQej6DZmUWQdlW8J2Lo1j0EH27m9FJiv_o
6vS6hu1iUzAi4lt8uegw"
```

Il faut ensuite le passer en paramètre dans le header de la requête

```
curl -H 'Accept: application/json' -H "Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGgiOiJST0xFX0FETUlOLFJPTEVfVVNFUiIsImV4c
CI6MTYwOTAwMzc2M30.bh8fQMGXawP354wGS1qG_KxSCD1_7hmthQej6DZmUWQdlW8J2Lo1j0EH27m9FJiv_o6
vS6hu1iUzAi4lt8uegw" http://localhost:8080/api/account
```

Sous linux il est possible d'automatiser il faut au préalable installer un "parser" Json. Par exemple après avoir installé le parser jq il est possible de stocker le token dans une variable d'environnement (ici TOKEN)

```
TOKEN=$(curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json'
--data '{"username":"admin","password":"admin"}'
http://localhost:8080/api/authenticate | jq -r '.id_token')
```

Il suffit en suite de passer le token de la manière suivante :

```
curl -H 'Accept: application/json' -H "Authorization: Bearer $TOKEN"
http://localhost:8080/api/account
```

Sous postman ou insomnia il faut faire le POST avec l'url et ajouter dans le body ce qui correspond au –data du curl (format json) :

```
{"username":"admin","password":"admin"}
```

Récapitulatif :

```
curl  -X POST -H 'Accept: application/json' -H 'Content-Type: application/json' --data
'{"username":"admin","password":"admin"}' http://localhost:8080/api/authenticate

curl -H 'Accept: application/json' -H "Authorization: Bearer
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGgiOiJST0xFX0FETUlOLFJPTEVfVVNFUiIsImV4c
CI6MTYwNjUwMTMxOH0.5ldyoVOtvIFIt3E4GlmTcfRg82XjodEcObIJ9JRqT3OU4KY_QOYfi7ELqge8xgQsdke
ne0-0S8F4zamU845Auw" http://localhost:8080/api/account

TOKEN=$(curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json'
--data '{"username":"admin","password":"admin"}'
http://localhost:8080/api/authenticate | jq -r '.id_token')

curl -H 'Accept: application/json' -H "Authorization: Bearer $TOKEN"
http://localhost:8080/api/account
```

Swagger

Les composants front-end et back-end étant séparés, l'API expose le composant back-end pour le composant frontal.

Le fichier d'Api est fourni dans le répertoire env/

Pour modifier le fichier de définition api.yml, Swagger-Editor. Il est possible d'utiliser l'outil en ligne ou bien

Démarrez une instance locale de swagger-editor à l'aide de docker en exécutant :

```
docker-compose -f src env/swagger-editor.yml up -d.
```

L'éditeur sera adressable à l'adresse http://localhost:7742.

Visualisation de l'Api au format json

Pour visualiser les spécifications d'API au format JSON dans un navigateur :

```
http://localhost:8080/v2/api-docs
```

Pour visualiser les spécifications d'API au format JSON dans la console :

```
curl -H 'Accept: application/json' -H 'Content-Type: application/json' --data
'{"username":"admin","password":"admin"}' http://localhost:8080/v2/api-docs
```

Remplacer *localhost:8080* par le bon *host* et le bon *port*.

Endpoints

Lancer la sauvegarde des json en base de donnée.

Les paramètres de sauvegarde sont num, pap ou all, ils doivent être ajoutés à la find du endpoint :

```
TOKEN=$(curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json'
--data '{"username":"admin","password":"admin"}'
http://localhost:8080/api/authenticate | jq -r '.id_token')
```

Puis pour avoir les articles papiers et numériques (all) :

```
curl -X POST -H 'Accept: application/json' -H "Authorization: Bearer $TOKEN"
http://localhost:8080/api/json/all
```

Installations optionnelles pour les développeurs.

Ces étapes sont optionnelles et ne sont pas nécessaires pour démarrer l'application en production.

Testing Pour les développeurs (optionnel)

Création des tables du contexte test

Lors de la création des tables avec liquibase, le pom possède un <contexts>!test></contexts> il faut donc le modifier car la table jhi_date_time_wrapper est absente de la base or elle est nécessaire aux tests <contexts>test></contexts>.

Si le préfixe n'est pas jhi il faut adapter le nom de la table avec le bon préfixe, par ex avec préfixe moisson :

```
moisson_date_time_wrapper
```

La table correspondante est dans le package :

```
{basePackage}.repository.timezone:
...
@Entity
@Table(name = "moisson_date_time_wrapper")
public class DateTimeWrapper implements Serializable {...}
```

Procédure de rattrapage :

Créer un fichier yyyyMMddHHmmss_initial_schema_test.xml avec le contenu suivant (les changeset doivent avoir des id différents qui n'existent pas dans la table databasechangelog)

```
<databaseChangeLog
xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.6.xsd
http://www.liquibase.org/xml/ns/dbchangelog-ext
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd">

<!-- <changeSet author="moisson" id="yyyyMMddHHmmss" context="test">
Il faut supprimer context="test" pour la génération
ou bien de modifier le pom en transformant <contexts>!test</contexts> en
<contexts>test</contexts> dans le pom.xml-->

    <changeSet author="moisson" id="yyyyMMddHHmmss">
        <createTable tableName="jhi_date_time_wrapper">
            <column  name="id" type="BIGINT">
                <constraints primaryKey="true"
primaryKeyName="jhi_date_time_wrapperPK"/>
            </column>
            <column name="instant" type="timestamp"/>
            <column name="local_date_time" type="timestamp"/>
            <column name="offset_date_time" type="timestamp"/>
            <column name="zoned_date_time" type="timestamp"/>
            <column name="local_time" type="time"/>
            <column name="offset_time" type="time"/>
            <column name="local_date" type="date"/>
        </createTable>
    </changeSet>
</databaseChangeLog>
```

Rajouter la ligne dans master.xml :

```
<include file="config/liquibase/changelog/yyyyMMddHHmmss_initial_schema_test.xml"
relativeToChangelogFile="false">
```

Attention si context="test" est conservé au niveau du pom.xml il faut modifier le pom.xml en conséquence au niveau du plugin liquibase :

```
<plugin>
<groupId>org.liquibase</groupId>
<artifactId>liquibase-maven-plugin</artifactId>
<version>${liquibase.version}</version>
<configuration>

<changeLogFile>${project.basedir}/src/main/resources/config/liquibase/master.xml</chan
geLogFile>

<diffChangeLogFile>${project.basedir}/src/main/resources/config/liquibase/changelog/${
maven.build.timestamp}_changelog.xml</diffChangeLogFile>
    <driver>org.postgresql.Driver</driver>
    <url>jdbc:postgresql://localhost:5432/moissoncatalogue</url>
    <defaultSchemaName></defaultSchemaName>
    <username>usercatalogue</username>
    <password>catalogue</password>

<referenceUrl>hibernate:spring:fr.tech.corree.domain?dialect=io.github.jhipster.domain
.util.FixedPostgreSQL10Dialect

&amp;hibernate.physical_naming_strategy=org.hibernate.boot.model.naming.PhysicalNaming
StrategyStandardImpl

&amp;hibernate.implicit_naming_strategy=org.hibernate.boot.model.naming.ImplicitNaming
StrategyJpaCompliantImpl</referenceUrl>
    <!--
<referenceUrl>hibernate:spring:fr.tech.corree.domain?dialect=io.github.jhipster.domain
.util.FixedPostgreSQL10Dialect

&amp;hibernate.physical_naming_strategy=fr.tech.corree.domain.naming.CataloguePhysical
NamingStrategyImpl

&amp;hibernate.implicit_naming_strategy=org.hibernate.boot.model.naming.ImplicitNaming
StrategyJpaCompliantImpl</referenceUrl> -->
    <verbose>true</verbose>
    <logging>debug</logging>
    #*<contexts>test</contexts> // ATTENTION JHIPSTER GENERE LE FICHIER AVEC
<contexts>!test</contexts>*#
</configuration>
<dependencies>
// Dependencies
</dependencies>
</plugin>
```

La table est créée

Puis en ligne de commande :

```
mvn liquibase:update
```

Si la table est bloquée avec l'erreur mvn tourne en boucle en attendant la libération du verrou il faut, passer cette requête :

```
UPDATE DATABASECHANGELOGLOCK SET LOCKED=false, LOCKGRANTED=null, LOCKEDBY=null where
ID=1;
```

Installation Flyway (Optionnel)

Il est possible de jouer les scripts avec Flyway au lieu de Liquibase. Flyway gère la "convention over configuration, c'est-à-dire que une fois le plug-in installer il va scruter conventionnellement dans main/resources/db/migration et jouer les scripts qui s'y trouvent avec un ordre prédéfini par les noms de fichiers(voir plus bas)

Pour installer un plugin Flyway Maven, il faut ajouter la définition de plugin suivante dans le pom.xml :

```
<plugin>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-maven-plugin</artifactId>
    <version>4.0.3</version>
</plugin>
```

Il faut vérifier la dernière version du plugin disponible sur Maven Central. Ce plugin Maven peut être configuré de quatre manières différentes. Consulter la documentation pour obtenir une liste de toutes les propriétés configurables.

1. Configuration du plugin

Configurer le plugin directement via la balise <configuration></configuration> dans la définition du plugin de notre pom.xml :

```
<plugin>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-maven-plugin</artifactId>
    <version>4.0.3</version>
    <configuration>
        <user>databaseUser</user>
        <password>databasePassword</password>
        <schemas>
            <schema>schemaName</schema>
        </schemas>
        ...
    </configuration>
</plugin>
```

1. Propriétés Maven

Configurer le plugin en spécifiant des propriétés configurables comme propriétés Maven dans notre pom :

```
<project>
    ...
    <properties>
        <flyway.user>databaseUser</flyway.user>
        <flyway.password>databasePassword</flyway.password>
        <flyway.schemas>schemaName</flyway.schemas>
        ...
    </properties>
    ...
</project>
```

1. Fichier de configuration externe

Configuration du plugin dans un fichier.properties séparé :

```
flyway.user=databaseUser
flyway.password=databasePassword
flyway.schemas=schemaName
...
```

Le nom du fichier de configuration par défaut est flyway.properties et doit résider dans le même répertoire que le fichier pom.xml. Le codage est spécifié par flyway.encoding (la valeur par défaut est UTF-8).

Pour utiliser un autre nom (par exemple customConfig.properties) comme fichier de configuration, il doit être spécifié explicitement lors de l'appel de la commande Maven :

```
$ mvn -Dflyway.configFile=customConfig.properties
```

1. Propriétés du système

Toutes les propriétés de configuration peuvent également être spécifiées en tant que propriétés systèmes lors de l'appel de Maven sur la ligne de commande :

```
$ mvn -Dflyway.user=databaseUser -Dflyway.password=databasePassword
  -Dflyway.schemas=schemaName
```

Voici un ordre de priorité lorsqu'une configuration est spécifiée de plusieurs manières :

```
Propriétés du système
Fichier de configuration externe
Propriétés de Maven
Plugin configuration
```

Première Migration

Pour définir la première migration, Flyway adhère à la convention de dénomination suivante pour les scripts de migration :

```
<Préfixe><Version> __ <Description>.sql
```

Où:

```
<Préfixe> - Le préfixe par défaut est V , qui peut être configuré dans le fichier de
configuration ci-dessus à l'aide de la propriété flyway.sqlMigrationPrefix .
<Version> - Numéro de version de la migration. Les versions majeures et mineures
peuvent être séparées par un trait de soulignement . La version de migration doit
toujours commencer par 1.
<Description> - Description textuelle de la migration. La description doit être
séparée des numéros de version par un double trait de soulignement.
```

Exemple :

```
V1_1_0__ma_premiere_migration.sql
```

Ensuite appeler la commande :

```
mvn clean flyway:migrate
```

Deuxième Migration

Une deuxième migration est faite en créant un deuxième fichier de migration avec le nom :

```
V2_0_0_ma_deuxieme_migration.sql
```

Le 2 est une convention, en fait toute version supérieure à la première est considérée come une deuxième migration

Pour vérifier que les deux migrations ont bien réussi il faut appeler la commande Maven suivante :

```
mvn flyway:info
```

Désactivation de Flyway dans Spring Boot

Il faut définir la propriété spring.flyway.enabled dans le fichier application-{profile}.properties :

```
spring.flyway.enabled=false
```

Comment fonctionne Flyway

Pour savoir quelles migrations ont déjà été appliquées, quand et par qui, Flyway ajoute une table de comptabilité spéciale au schéma.

Cette table de métadonnées suit également les sommes de contrôle de migration et indique si les migrations ont réussi ou non.

Le framework effectue les étapes suivantes pour s'adapter aux schémas de base de données en évolution :

1. Il vérifie un schéma de base de données pour localiser sa table de métadonnées (SCHEMA_VERSION par défaut). Si la table de métadonnées n'existe pas, elle en créera une.
2. Il analyse un chemin de classe d'application pour les migrations disponibles
3. Il compare les migrations à la table de métadonnées. Si un numéro de version est inférieur ou égal à une version marquée comme actuelle, il est ignoré : par conséquent il ne faut pas modifier la numérotation des scripts après la première migration.
4. Il marque toutes les migrations restantes comme des migrations en attente. Ceux-ci sont triés en fonction du numéro de version et sont exécutés dans l'ordre.
5. Au fur et à mesure que chaque migration est appliquée, la table de métadonnées est mise à jour en conséquence.

Commandes mvn

Flyway prend en charge les commandes de base suivantes pour gérer les migrations de bases de données.

Info : imprime l'état / la version actuelle d'un schéma de base de données. Il imprime quelles migrations sont en attente, quelles migrations ont été appliquées, quel est l'état des migrations appliquées et quand elles ont été appliquées.
Migrate : migre un schéma de base de données vers la version actuelle. Il analyse le chemin de classe pour les migrations disponibles et applique les migrations en attente.
Baseline : Baseline une base de données existante, à l'exclusion de toutes les migrations, y compris baselineVersion . Baseline aide à démarrer avec Flyway dans une base de données existante. Les migrations plus récentes peuvent alors être appliquées normalement.
Validate : valide le schéma de base de données actuel par rapport aux migrations disponibles.
Repair : réparations de la table de métadonnées.
Clean : supprime tous les objets dans un schéma configuré. Tous les objets de base de données sont supprimés. Bien sûr, vous ne devez jamais utiliser clean sur une base de données de production.