Installation

Important	Avant de déployer 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 . déploiement de l'application
```

Configuration du fichier de propriétés. Le fichier de configuration est fourni indépendamment du fichier .war:

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

Une fois les paramètres fixer il faut le déposer dans un répertoire config créé au meme niveau que le répertoire de contexte de l'application:

Exemple pour tomcat 9 :

```
répertoire de config:
  /tomcat9/webapps/config
répertoire de contexte:
  /tomcat9/webapps/moisooncatalogue
```

Paramètres du fichier de configuration : Base de données

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 géné
  # Ne pas modifier le nom de la base  : moissoncatalogue
  url: jdbc:postgresql://localhost:5432/moissoncatalogue
  # Ne pas modifier username: usercatalogue
  username: usercatalogue
  password: catalogue
```

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

Important	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 PA
```

par le NOUVEAU_MOT_DE_PASSE :

```
CREATE ROLE usercatalogue LOGIN NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE NOREPLICATION PA
```

ElasticSearch

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 sinon car l'application ne dé
# username: admin
# password: admin
```

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
```

Mail

Optionnel car non utilisé pour le moment :

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

Création de la ase de données. Créations 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 créer les roles et la base correspondante.

Il faut installer postgres sur le serveur

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
```

Création des tables

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

Note	Les scripts SQL sont fournis et situés dans le répertoire db/migration et peuvent être utilisés "As is" ou avec Flyway

Important	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.

Déploiement de l'application

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
```

Note	-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:user@localhost:8080/api/authenticate
```

Authentification avec token JWT

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

Il faut dan un premier temps générer le token, avec le 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 '{"us
le curl génère un token.

"id_token" : "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImF1dGgiOiJST0xFX0FETUlOLFJPTEVfVVNFU
```

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

```
curl -H 'Accept: application/json' -H "Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOi
```

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

```
TOKEN=$(curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json' --dat
```

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

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

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 '{"us
curl -H 'Accept: application/json' -H "Authorization: Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOi
TOKEN=$(curl -X POST -H 'Accept: application/json' -H 'Content-Type: application/json' --dat
curl -H 'Accept: application/json' -H "Authorization: Bearer $TOKEN" http://localhost:8080/a
```

Swagger Les composants front-end et back-end étant séparés, l'API expose le composant back-end pour le composant frontal ou des intégrations d'applications tierces.

Les spécifications des API back-end sont exposées par l'intermédiaire de Swagger.

Pour visualiser les spécifications d'API au format JSON :

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

```
dans un navigateur ou
```

```
curl -H 'Accept: application/json' -H 'Content-Type: application/json' --data '{"username":'
```

Pour visualiser les spécifications d'API avec SwaggerN :

```
http://localhost:8080/swagger-ui/index.html
```

```
dans un navigateur
```

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' --dat
```

puis pour avoir les articles papiers et numériques (all :

```
curl -X POST -H 'Accept: application/json' -H "Authorization: Bearer $TOKEN" http://localhos
```

Testing 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
http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/dbchangelog/

<!-- <changeSet author="jhipster" 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</cor

    <changeSet author="jhipster" 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 conséquense 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</change
    <diffChangeLogFile>${project.basedir}/src/main/resources/config/liquibase/changelog/${ma
    <driver>org.postgresql.Driver</driver>
    <url>jdbc:postgresql://localhost:5432/catalogue-ng-jhipster</url>
    <defaultSchemaName></defaultSchemaName>
    <username>catalogue</username>
    <password>catalogue</password>
    <referenceUrl>hibernate:spring:fr.tech.corree.domain?dialect=io.github.jhipster.domain.u
    &amp;hibernate.physical_naming_strategy=org.hibernate.boot.model.naming.PhysicalNamingSt
    &amp;hibernate.implicit_naming_strategy=org.hibernate.boot.model.naming.ImplicitNamingSt
    <!-- <referenceUrl>hibernate:spring:fr.tech.corree.domain?dialect=io.github.jhipster.dom
    &amp;hibernate.physical_naming_strategy=fr.tech.corree.domain.naming.CataloguePhysicalNa
    &amp;hibernate.implicit_naming_strategy=org.hibernate.boot.model.naming.ImplicitNamingSt
    <verbose>true</verbose>
    <logging>debug</logging>
    #*<contexts>test</contexts> // ATTENTION JHIPSTER GENERE LE FICHIER AVEC <contexts>!test
</configuration>
<dependencies>
// Dependencies
</dependencies>
</plugin>
```

Puis en ligne de commande :

```
mvn liquibase:update
```

Note	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;

La table est créée

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 cripts 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> 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 configu
<Version> - Numéro de version de la migration. Les versions majeures et mineures peuvent êtr
<Description> - Description textuelle de la migration. La description doit être séparée des
```

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 quell
Migrate : migre un schéma de base de données vers la version actuelle. Il analyse le chemin
Baseline : Baseline une base de données existante, à l'exclusion de toutes les migrations, y
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 donnée
```

Last updated 2020-12-10 20:30:01 +0100