Une plate-forme de micro-services

Réalisation

Sommaire

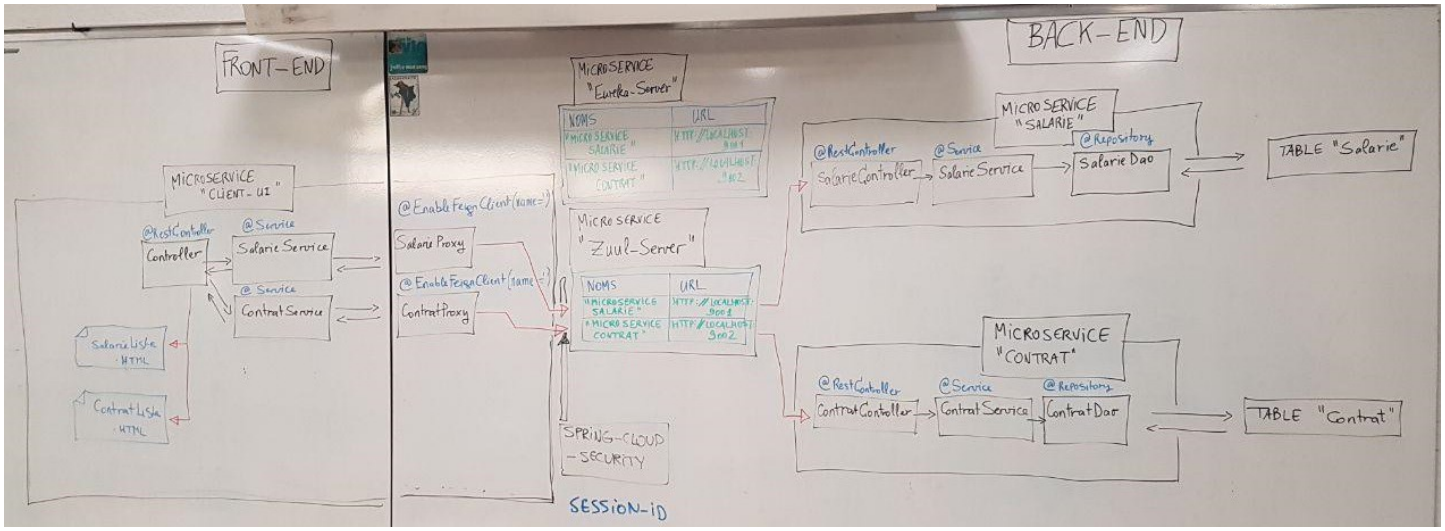
(01.) La structure interne de la plate-forme

Une **plate-forme de micro-services** est constituée des **applications** ci-dessous :

(01.01.) Les micro-services « back-end »

(01.02.) Les micro-services « front-end »

(01.03.) Les micro-services « edge »



(01.01.) Les micro-services « back-end »

Un micro-service « back-end » est une application serveur, qui possède les **caractéristiques** suivantes :

No.	Caractéristique	Détails
1	Héritage (par maven) d'une application « Spring-Boot »	Cet héritage est spécifié dans le fichier de configuration de l'application : « pom.xml »
2	Accès à une base de données	Ici, nous choisissons une BDD relationnelle.
3	Présence d'une couche de persistance (obligatoire)	Cette couche de persistance comporte les composants suivants : ▪ Des entités ▪ Des DAO.
4	Présence d'une couche « proxy »	Cette couche « proxy » comporte des composants applicatifs de type « proxy ». Ceux-ci possèdent les 2 fonctionnalités suivantes : ▪ Transformer un objet « JAVA » en un bloc « JSON » ▪ Transformer un appel de méthode JAVA en un appel « REST »
5	Présence d'une couche métier (facultative)	Cette couche métier comporte les (éventuels) traitements métiers de l'application.
6	Présence d'une couche « REST » (obligatoire)	Cette couche « REST » possède les fonctionnalités suivantes : ▪ Recevoir des appels « REST » ▪ Renvoyer des réponses « JSON »

(01.02.) Les micro-services « front-end »

Un micro-service « back-end » est une application serveur, qui possède les **caractéristiques** suivantes :

No.	Caractéristique	Détails
1	Héritage (par maven) d'une application « Spring-Boot »	Cet héritage est spécifié dans le fichier de configuration de l'application : « pom.xml »
2	Présence d'une couche « proxy »	Cette couche « proxy » comporte des composants applicatifs de type « proxy ». Ceux-ci possèdent les 2 fonctionnalités suivantes : ▪ Transformer un objet « JAVA » en un bloc « JSON » ▪ Transformer un appel de méthode JAVA en un appel « REST »
3	Présence d'une couche métier (obligatoire)	Cette couche métier comporte les traitements métiers de l'application.
4	Présence d'une couche « REST » (obligatoire)	Cette couche « REST » possède les fonctionnalités suivantes : ▪ Recevoir des appels « REST » ▪ Renvoyer des réponses « JSON »

(01.03.) Les micro-services « edge »

Un micro-service « edge » est une application serveur, qui possède les **caractéristiques** suivantes :

No.	Caractéristique	Détails
1	Héritage (par maven) d'une application « Spring-Boot »	Cet héritage est spécifié dans le fichier de configuration de l'application : « pom.xml »
2	Accès à une base de données (facultatif)	Cette BDD existe uniquement pour un serveur d'authentification. Ici, nous choisirons une BDD relationnelle.
3	Présence d'une couche de persistance (facultative)	Cette couche de persistance existe uniquement pour un serveur d'authentification. Elle comporte les composants suivants : ▪ des entités ▪ des DAO.
4	Présence d'une couche métier (facultative)	Cette couche métier comporte les (éventuels) traitement métiers de l'application.
5	Présence d'une couche « REST » (facultative)	Cette couche « REST » est nécessaire uniquement pour un serveur d'authentification. Elle possède les fonctionnalités suivantes : ▪ Recevoir des appels « REST » ▪ Renvoyer des réponses « JSON »
6	Caractéristiques « edge »	Chaque micro-service « edge » possède un seul des types ci-dessous : ▪ Le type « serveur de découverte » : réalisé par un composant « eureka » ▪ Le type « point d'entrée « : réalisées par un composant « zuul » ▪ Le type « serveur d'authentification » : réalisées par un composant «oAuth2»

(02.) Le plan de réalisation d'un micro-service

La réalisation d'un micro-service est constituée des étapes ci-dessous. Vous devrez les parcourir en suivant l'ordre indiqué. (La succession d'étapes ci-dessous peut être considérée comme un plan de réalisation).

(02.01.) Choisir le type du micro-service

(02.02.) Renseigner le fichier de configuration « pom.xml »

(02.03.) Construire l'arborescence du projet

(02.04.) Repérer le fichier de propriétés

(02.05.) Renseigner le fichier de configuration des traceurs

(02.06.) Configurer la classe principale de l'application

(02.07.) Configurer la classe principale des tests de l'application

(02.01.) Choisir le type du micro-service

Un micro-service peut avoir l'un des types ci-dessous :

- Type « Back-end. »
- Type « Front-end. »
- Type « Edge. »

(02.02.) Renseigner le fichier de configuration « pom.xml »

Cette construction peut être effectuée avec une assistance sur le site **« starter.spring.io** ».

Ce fichier comporte **les rubriques listées ci-dessous**. Veuillez vérifier leur contenu.

(02.02.01.) Le parent du projet
(02.02.02.) Les propriétés du projet
(02.02.03.) Les dépendances du projet
(02.02.04.) La compilation du projet

(02.03.) Construire l'arborescence du projet

Décompresser le fichier « zip ».

(02.04.) Repérer le fichier de propriétés

Pour ce fichier de propriétés, vous devez choisir l'un de ces 2 noms :

- application.properties
- application.yml

Quelques informations concernant ces2 fichiers :

No.	Nom du fichier	Commentaires
1	application.properties	Représentation des données sous forme de lignes : une ligne par donnée.
2	application.yml	Représentation des données sous forme d'un arbre : une branche par donnée.

(02.05.) Renseigner le fichier de configuration des traceurs

Pour ce fichier de configuration des traceurs, vous devrez choisir l'un des 2 noms ci-dessous :

- logback-spring.xml
- log4j2-spring.xml

Quelques informations concernant ces2 fichiers :

No.	Nom du fichier	Commentaires
1	logback-spring.xml	Configuration d'un traceur intégré dans le framework spring. **Conséquence :** Dans le fichier « pom.xml », aucune dépendance à rajouter
2	Log4j2-spring.xml	Configuration d'un traceur extérieur au framework spring. **Conséquence :** Dans le fichier « pom.xml », rajouter la dépendances correspondant à « log4j2 ».

(02.06.) Configurer la classe principale de l'application

Cette classe principale de l'application est la **classe qui permet le lancement de l'application**. Quelques informations concernant cette classe :

No.	Spécificités de la classe	Commentaires
1	Rôle de la classe	▪ Cette classe permet le lancement de l'application. ▪ Cette classe contient la méthode « main » (lançable par la JVM).
2	Nom de la classe	[$ArtifactId]Application
3	Annotation sur la classe	@SpringBootApplication

(02.07.) Configurer la classe principale des tests de l'application

Cette classe principale des tests de l'application est la **classe qui permet le lancement des tests de l'application.** Quelques informations concernant cette classe :

No.	Spécificités de la classe	Commentaires
1	Rôle de la classe	▪ Cette classe permet le lancement des tests de l'application. ▪ Cette classe contient des méthodes de test (lançables par le Framework Spring-Boot).
2	Nom de la classe	[$ArtifactId]ApplicationTest
3	Annotation sur la classe	@SpringBootTest @RunWith (SpringRunner.class)
4	Annotations sur les méthodes de test	@Test

(03.) Le fichier de configuration d'un micro-service

Le **fichier de configuration d'un micro-service** doit être renseigné en fonction du **type du micro-service** concerné. Les différents types de micro-services sont détaillés (avec leur configuration) dans les paragraphes ci-dessous :

(03.01.) Les micro-services « back-end »

(03.02.) Les micro-services « front-end »

(03.03.) Les micro-services « serveur de découverte »

(03.04.) Les micro-services « point d'entrée»

(03.05.) Les micro-services « serveur de configuration»

(03.06.) Les micro-services « serveur d'authentification»

(03.01.) Les micro-services « back-end »

No.	Dépendance	Commentaire
01	DevTools	Fonctionnalités de recompilation et redéploiement automatique
02	Lombok	Fonctionnalités de mise en place automatique dans des classes java : ▪ De constructeurs ▪ De getters ▪ De setters
03	Actuator	Fonctionnalités de rechargement de fichier de propriétés en cours d'exécution
04	Rest-Repositories	Fonctionnalités combinées de ▪ « RestController » ▪ « JpaRepository »
05	MySQL / MariaDB / H2	Fonctionnalités de connexion aux bases de données MySQL / MariaDB / H2.
06	JPA	Fonctionnalités de persistance d'objets java.
07	Web	Fonctionnalité de : ▪ Réception d'appels REST ▪ Renvoi de réponses JSON
08	Eureka-discovery	Fonctionnalité de découverte côté client
09	Ribbon	Fonctionnalité d'équilibrage de charge
10	Config-client	Fonctionnalité d'externalisation de configuration côté client

(03.02.) Les micro-services « front-end »

No.	Dépendance	Commentaire
01	DevTools	Fonctionnalités de recompilation et redéploiement automatique
02	Lombok	Fonctionnalités de mise en place automatique dans des classes java : ▪ De constructeurs ▪ De getters ▪ De setters
03	Actuator	Fonctionnalités de rechargement de fichier de propriétés en cours d'exécution
04	Thymeleaf	Fonctionnalités de fabrication assistée de pages HTML
05	Cloud-Bootstrap	Fonctionnalités de fabrication assistée de composants CSS
06	Web	Fonctionnalité de : ▪ Réception d'appels REST ▪ Renvoi de réponses JSON
07	Eureka-discovery	Fonctionnalité de découverte côté client
08	Ribbon	Fonctionnalité d'équilibrage de charge
09	Config-client	Fonctionnalité d'externalisation de configuration côté client
10	Feign	Fonctionnalités de : ▪ Transformation d'un objet « JAVA » en bloc « JSON ». ▪ Transformation d'un appel de méthode « JAVA » en appel « REST ».

(03.03.) Les micro-services « serveur de découverte »

No.	Dépendance	Commentaire
01	Eureka-server	Fonctionnalité de découverte côté serveur
02	Config-client	Fonctionnalité d'externalisation de configuration côté client

(03.04.) Les micro-services « point d'entrée»

No.	Dépendance	Commentaire
01	Eureka-discovery	Fonctionnalité de découverte côté client
02	Config-client	Fonctionnalité d'externalisation de configuration côté client
03	Zuul	Fonctionnalité de point d'entrée de requête HTTP

(03.05.) Les micro-services « serveur de configuration»

No.	Dépendance	Commentaire
01	Config-server	Fonctionnalité d'externalisation de configuration côté serveur

(03.06.) Les micro-services « serveur d'authentification»

[à compléter]

(04.) Le fichier de propriétés d'un micro-service

Le **fichier de propriétés d'un micro-service** est chargé en RAM au démarrage du micro-service.
Ce fichier est constitué des **rubriques ci-dessous**.

(04.01.) Configuration du port d'écoute du traçage

(04.02.) Configuration du port d'écoute du serveur HTTP

(04.03.) Configuration « Actuator »

(04.04.) Configuration « Eureka »

(04.05.) Configuration « Ribbon »

(04.06.) Configuration « Spring-Application »

(04.07.) Configuration « Spring-Datasource »

(04.08.) Configuration « Spring-JPA »

(04.09.) Configuration « Spring-Security »

(04.01.) Configuration du port d'écoute du traçage

```
#########################################################
# ---- (01.)LOGGING-CONFIGURATION ----
#########################################################
logging:
  config: "classpath:log4j2-spring.xml"
```

(04.02.) Configuration du port d'écoute du serveur HTTP

```
#########################################################
# ---- (02.)SERVER-PORT-CONFIGURATION ----
#########################################################
server:
  port: 9010
```

(04.03.) Configuration «Actuator»

```
#########################################################
# ---- (03.)ACTUATOR-CONFIGURATION ----
#########################################################
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

(04.04.) Configuration «Eureka»

```
#########################################################
# ---- (04.)EUREKA-CONFIGURATION ----
#########################################################
eureka:
  client:
    serviceUrl:
      defaultZone: "http://localhost:9102/eureka/"
```

(04.05.) Configuration «Ribbon»

```
#########################################################
# ---- (05.)RIBBON-CONFIGURATION ----
#########################################################
microservice-produit:
  ribbon:
    listOfServers : "localhost:9010,localhost:9011"
```

(04.06.) Configuration «Spring-Application»

```
#########################################################
# ---- (05.)SPRING-CONFIGURATION ----
#########################################################
spring:
  # ---- (01.01.)SPRING-APPLICATION-CONFIGURATION ----
  application:
    name: "produitServer"
```

(04.07.) Configuration «Spring-Datasource»

```
#########################################################
# ---- (05.)SPRING-CONFIGURATION ----
#########################################################
spring:
  # ---- (05.02.)SPRING-DATASOURCE-CONFIGURATION ----
  datasource:
    url: "jdbc:mariadb://localhost:3306/produitdb?createDatabaseIfNotExist=true"
    username: "root"
    password: "tcharou"
    driver-class-name: "org.mariadb.jdbc.Driver"
    sql-script-encoding: UTF-8
    data: "classpath:import.sql"
```

(04.08.) Configuration «Spring-JPA»

```
#########################################################
# ---- (05.)SPRING-CONFIGURATION ----
#########################################################
spring:
  # ---- (05.03.)SPRING-JPA-CONFIGURATION ----
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: "create"
    properties:
      hibernate:
        dialect: "org.hibernate.dialect.MariaDB5Dialect"
```

(04.09.) Configuration «Spring-Security»

```
#########################################################
# ---- (05.)SPRING-CONFIGURATION ----
#########################################################
spring:
  # ---- (05.04.)SPRING-SECURITY-CONFIGURATION ----
  security:
    user:
      name: "utilisateur"
      password: "mdp"
```

(05.) Le fichier de configuration du traçage

Ce fichier possède les nom et emplacement indiqués ci-dessous :

1	Nom	▪ logback-spring.xml ▪ log4j2-spring.xml
2	Emplacement	[racine du projet]/src/main/resources

(05.01.) La rubrique : « Properties »

Cette rubrique est constituée du bloc ci-dessous. Veuillez le récupérer , svp.

```
<Properties>
   <Property name="log-path">./logExecute</Property>
</Properties>
```

(05.02.) La rubrique : « Appenders »

Cette rubrique est constituée du bloc ci-dessous. Veuillez le récupérer, svp.

```
<!-- ============================================================================ -->
<!-- (01.) Appenders :                                                            -->
<!--   - (01.A.) Chaque Appender pointe sur le fichier qui lui est associe        -->
<!--   - (01.B.) Chaque Appender ecrit dans ce fichier                            -->
<!-- ============================================================================ -->
<Appenders>
   <Console name="Console-Appender" target="SYSTEM_OUT">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </Console>

   <File name="UtilisateurServer-File-Appender" fileName="${log-path}/UtilisateurServer.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="Controller-File-Appender" fileName="${log-path}/UtilisateurServer-controller.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="Service-File-Appender" fileName="${log-path}/UtilisateurServer-service.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="Exception-File-Appender" fileName="${log-path}/UtilisateurServer-exception.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="Model-File-Appender" fileName="${log-path}/UtilisateurServer-model.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="Spring-File-Appender" fileName="${log-path}/UtilisateurServer-spring.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="SpringCloud-File-Appender" fileName="${log-path}/UtilisateurServer-springCloud.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="SpringBoot-File-Appender" fileName="${log-path}/UtilisateurServer-springBoot.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>

   <File name="SpringContext-File-Appender" fileName="${log-path}/UtilisateurServer-springContext.log">
      <PatternLayout>
         <pattern>[%-5level] %d{yyyy-MM-dd HH:mm:ss.SSS} [%t] %c{1} - %msg%n</pattern>
      </PatternLayout>
   </File>
</Appenders>
```

(05.03.) La rubrique : « Loggers »

Cette rubrique est constituée du bloc ci-dessous. Veuillez le récupérer, svp.

```
<!-- ============================================================================ -->
<!-- (02.) Loggers :                                                              -->
<!--   - (02.A.) Chaque Logger scanne le package qui lui est associe              -->
<!--   - (02.B.) Chaque Logger récupère les logs qu'il trouve dans les classes    -->
<!-- ============================================================================ -->
<Loggers>
   <Root>
      <AppenderRef ref="Console-Appender" level="all" />
   </Root>

   <Logger name="fr.tacticrh.utilisateur" level="all" additivity="true">
      <AppenderRef ref="UtilisateurServer-File-Appender" level="all" />
   </Logger>

   <Logger name="fr.tacticrh.utilisateur.web.controller" level="all" additivity="true">
      <AppenderRef ref="Controller-File-Appender" level="all" />
   </Logger>

   <Logger name="fr.tacticrh.utilisateur.business.service" level="all" additivity="true">
      <AppenderRef ref="Service-File-Appender" level="all" />
   </Logger>

   <Logger name="fr.tacticrh.utilisateur.business.exception" level="all" additivity="true">
      <AppenderRef ref="Exception-File-Appender" level="all" />
   </Logger>

   <Logger name="fr.tacticrh.utilisateur.persistence.model" level="all" additivity="true">
      <AppenderRef ref="Model-File-Appender" level="all" />
   </Logger>

   <Logger name="org.springframework" level="info" additivity="true">
      <AppenderRef ref="Spring-File-Appender" level="all" />
   </Logger>

   <Logger name="org.springframework.cloud" level="info" additivity="true">
      <AppenderRef ref="SpringCloud-File-Appender" level="all" />
   </Logger>

   <Logger name="org.springframework.boot" level="info" additivity="true">
      <AppenderRef ref="SpringBoot-File-Appender" level="all" />
   </Logger>

   <Logger name="org.springframework.context" level="info" additivity="true">
      <AppenderRef ref="SpringContext-File-Appender" level="all" />
   </Logger>
</Loggers>
```

(06.) La classe principale de l'application

La classe principale de l'application possède les noms, package et répertoire indiqués ci-dessous :

Nom	[$ArtifactId]Application
Package	Package principal (ce package coïncide avec le « groupId »)
Répertoire	src/main/java

Le contenu de ce fichier est détaillé dans les rubriques ci-dessous :

(06.01.) Les annotations sur la classe

Cette classe principale porte des annotations spécifiques en fonction du type de micro-service concerné. Pour chacun des type s de micro-service indiqué ci-dessous, les annotations sont fournies.

(06.01.01.) Annotations sur une classe principale de type « Back-End »

(06.01.02.) Annotations sur une classe principale de type « Front-End »

(06.01.03.) Annotations sur une classe principale de type «Eureka-Server »

(06.01.04.) Annotations sur une classe principale de type «Zuul-Server »

(06.01.05.) Annotations sur une classe principale de type «Config-Server »

(06.01.01.) Annotations sur une classe principale de type « Back-End »

Sur une classe principale de type « Back-End » :

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@EnableEurekaClient
@SpringBootApplication
public class MProduitApplication {...}
```

(06.01.02.) Annotations sur une classe principale de type « Front-End »

Sur une classe principale de type « Front-End » :

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@EnableFeignClients(basePackages= 'com.clientui.feign.proxy')
@EnableEurekaClient
@SpringBootApplication
public class ClientUIApplication {...}
```

(06.01.03.) Annotations sur une classe principale de type «Eureka-Server »

Sur une classe principale de type « Eureka-Server » :

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@EnableEurekaServer
@SpringBootApplication
public class EurekaServerApplication {...}
```

(06.01.04.) Annotations sur une classe principale de type «Zuul-Server »

Sur une classe principale de type « Zuul-Server » :

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@EnableEurekaClient
@EnableZuulProxy
@SpringBootApplication
public class ZuulServerApplication {...}
```

(06.01.05.) Annotations sur une classe principale de type «Config-Server »

Sur une classe principale de type « Config-Server » :

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {...}
```

(06.02.) L'attribut « Logger »

L'attribut « Logger » est déclaré dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
/**
 * <b>OBJET QUI POSSEDE LES FONCTIONNALITES SUIVANTES : </b> <br/>
 * <br/>
 * Les fonctionnalités d'écriture de messages de log dans la console.
 */
private static final Logger LOGGER = LoggerFactory.getLogger(XXXApplication.class);
```

(06.03.) La méthode « main »

La méthode « main » est implémentée dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
/**
 * <b>METHODE D'ENTREE DE L'APPLICATION</b><br/>
 *
 * @param args
 */
public static void main(String[] args) {

   LOGGER.info("CLASS : XXXApplication -- METHOD : main - BEGIN");
   SpringApplication.run(XXXApplication.class, args);
   LOGGER.info("CLASS : XXXApplication -- METHOD : main -- END");
}
```

(07.) La classe de test principale de l'application

La classe de test principale de l'application possède les noms, package et répertoire indiqués ci-dessous :

Nom	[$ArtifactId]ApplicationTest
Package	Package principal (ce package coïncide avec le « groupId » dans le fichier « pom.xml »)
Répertoire	src/test/java

Le contenu de ce fichier est détaillé dans les rubriques ci-dessous :

(07.01.) Les annotations sur la classe

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class XXXApplicationTest {...}
```

(07.02.) L'attribut « Logger »

L'attribut « Logger » est déclaré dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
  /**
   * <b>OBJET QUI POSSEDE LES FONCTIONNALITES SUIVANTES : </b> <br/>
   * <br/>
   * Les fonctionnalités d'écriture de messages de log dans la console.
   */
  private static final Logger LOGGER = LoggerFactory.getLogger(XXXApplicationTest.class);
```

(07.03.) L'attribut « DaoTest »

L'attribut « DaoTest » est déclaré dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
  /**
   * <b>OBJET QUI POSSEDE LES FONCTIONNALITES SUIVANTES : </b> <br/>
   * <br/>
   * Les fonctionnalités des test sur les DAO de l'application.
   */
  @Autowired
  private DaoTest daoTest;
```

(07.04.) La méthode « contextLoad »

La méthode « contextLoad » est implémentée dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
  /**
   * <b>METHODE PRINCIPALE DE TEST DE L'APPLICATION</b><br/>
   *
   * @param args
   */
  @Test
  public void contextLoad() {

     LOGGER.info("CLASS : XXXApplicationTest -- METHOD : contextLoad - BEGIN");
     SpringApplication.run(XXXApplicationTest.class, args);
     LOGGER.info("CLASS : XXXApplicationTest -- METHOD : contextLoad -- END");
  }
```

(08.) La classe de test des DAO de l'application

La classe de test principale de l'application possède les noms, package et répertoire indiqués ci-dessous :

Nom	DaoTest
Package	Sous-package « dao » du package principal (le package principal coïncide avec le « groupId » dans le fichier « pom.xml »)
Répertoire	src/test/java

Le contenu de ce fichier est détaillé dans les rubriques ci-dessous :

(08.01.) Les annotations sur la classe

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@Component
public class DaoTest {...}
```

(08.02.) L'attribut « Logger »

L'attribut « Logger » est déclaré dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
  /**
   * <b>OBJET QUI POSSEDE LES FONCTIONNALITES SUIVANTES : </b> <br/>
   * <br/>
   * Les fonctionnalités d'écriture de messages de log dans la console.
   */
  private static final Logger LOGGER = LoggerFactory.getLogger(DaoTest.class);
```

(08.03.) Les attributs « roleDao » et « utilisateurDao »

Les attributs « roleDao » et « utilisateurDao » sont déclarés dans les 2 blocs de code ci-dessous. Veuillez les récupérer, svp.

```
  /**
   * <b>COMPOSANT DE PERSISTANCE RELATIF A L'ENTITE CI-DESSOUS :</b><br/>
   * <br/>
   * ->ENTITE : 'Role'.<br/>
   */
   @Autowired
   private RoleDao roleDao;
```

```
    /**
    * <b>COMPOSANT DE PERSISTANCE RELATIF A L'ENTITE CI-DESSOUS :</b><br/>
    * <br/>
    * ->ENTITE : 'Utilisateur'.<br/>
    */
    @Autowired
    private UtilisateurDao utilisateurDao;
```

(08.04.) La méthode publique « execute »

La méthode publique « execute » est implémentée dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
    /**
     * <b>EXECUTION DE LA SEQUENCE DE TEST COMPLETE</b><br/>
     */
    public void execute() {
      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : execute -- BEGIN");

      this.setUp();
      this.allTests();
      this.tearDown();

      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : execute -- END");
    }
```

(08.05.) Les méthodes privées « setUp », « tearDow » et « allTests »

Les méthodes privées « setUp », « tearDown » et « allTests » sont déclarées dans les 3 blocs de code ci-dessous. Veuillez les récupérer, svp.

```
    /**
     * <b>ALIMENTER LA BASE DE DONNEES</b><br/>
     */
    private void setUp() {
      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : setUp -- BEGIN");

      Role roleUti = this.roleDao.findByLibelle("Utilisateur"   );
      Role roleCdt = this.roleDao.findByLibelle("Candidat"      );
      Role roleMgr = this.roleDao.findByLibelle("Manager"       );
      Role roleAdm = this.roleDao.findByLibelle("Administrateur");

      Personne personneYDH = new Personne("De-Hanot" , "Yves"        , "01/01/2000");
      Personne personneTDA = new Personne("Dalgalian", "Tcharou"     , "01/01/2000");
      Personne personneABC = new Personne("Bachri"   , "Amin"        , "01/01/2000");
      Personne personneJJP = new Personne("Pagan"    , "Jean-Jacques", "01/01/2000");

      Utilisateur utilisateurYDH = new Utilisateur(personneYDH, "yves.de-hanot@afpa.fr"     , "ydh", roleUti);
      Utilisateur utilisateurTDA = new Utilisateur(personneTDA, "tcharou.dalgalian@afpa.fr" , "tda", roleCdt);
      Utilisateur utilisateurABC = new Utilisateur(personneABC, "amin.bachri@afpa.fr"       , "abc", roleMgr);
      Utilisateur utilisateurJJP = new Utilisateur(personneJJP, "jean-jacques.pagan@afpa.fr", "jjp", roleAdm);

      this.utilisateurDao.save(utilisateurYDH);
      this.utilisateurDao.save(utilisateurTDA);
      this.utilisateurDao.save(utilisateurABC);
      this.utilisateurDao.save(utilisateurJJP);

      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : setUp -- END");
    }
```

```
    /**
     * <b>VIDER LA BASE DE DONNEES</b><br/>
     */
    private void tearDown() {
      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : tearDown -- BEGIN");

      List<Utilisateur> utilisateursUti = this.utilisateurDao.findByRole_Libelle("Utilisateur"   );
      List<Utilisateur> utilisateursCdt = this.utilisateurDao.findByRole_Libelle("Candidat"      );
      List<Utilisateur> utilisateursMgr = this.utilisateurDao.findByRole_Libelle("Manager"       );
      List<Utilisateur> utilisateursAdm = this.utilisateurDao.findByRole_Libelle("Administrateur");

      for (Utilisateur utilisateur : utilisateursUti) {
         LOGGER.info("utilisateur to be deleted: " + utilisateur.toString());
         this.utilisateurDao.deleteById(utilisateur.getId());
      }

      for (Utilisateur utilisateur : utilisateursCdt) {
         LOGGER.info("candidat to be deleted: " + utilisateur.toString());
         this.utilisateurDao.deleteById(utilisateur.getId());
      }

      for (Utilisateur utilisateur : utilisateursMgr) {
         LOGGER.info("manager to be deleted: " + utilisateur.toString());
         this.utilisateurDao.deleteById(utilisateur.getId());
      }

      for (Utilisateur utilisateur : utilisateursAdm) {
         LOGGER.info("administrateur to be deleted: " + utilisateur.toString());
         this.utilisateurDao.deleteById(utilisateur.getId());
      }
      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : tearDown -- END");
  }
```

```
    /**
     * <b>EFFECTUER LES TESTS SUR LES DAO</b><br/>
     */
    private void allTests() {
      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : allTests -- BEGIN");
      try {
         Thread.sleep(50000);

      } catch (InterruptedException e) {
         LOGGER.info("Mise en attente du thread courant - Interruption du thread");
      }
      LOGGER.info("CLASS : " + this.getClass().getSimpleName() + " -- METHOD : allTests -- END");
  }
```

(09.) La classe de test des services de l'application

La classe de test principale de l'application possède les noms, package et répertoire indiqués ci-dessous :

Nom	ServiceTest
Package	Sous-package « service » du package principal (le package principal coïncide avec le « groupId » dans le fichier « pom.xml »)
Répertoire	src/test/java

Le contenu de ce fichier est détaillé dans les rubriques ci-dessous :

(09.01.) Les annotations sur la classe

Les annotations à poser sur la classe sont indiquées ci-dessous. Veuillez les récupérer, svp.

```
@Component
public class ServiceTest {...}
```

(09.02.) L'attribut « Logger »

L'attribut « Logger » est déclaré dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
  /**
   * <b>OBJET QUI POSSEDE LES FONCTIONNALITES SUIVANTES : </b> <br/>
   * <br/>
   * Les fonctionnalités d'écriture de messages de log dans la console.
   */
  private static final Logger LOGGER = LoggerFactory.getLogger(ServiceTest.class);
```

(09.03.) La méthode « execute »

La méthode « contextLoad » est implémentée dans le bloc de code ci-dessous. Veuillez le récupérer, svp.

```
  /**
   * <b>METHODE DE TEST DES SERVICES DE L'APPLICATION</b><br/>
   *
   * @param args
   */
  public void execute() {

     LOGGER.info("CLASS : ServiceTest -- METHOD : execute - BEGIN");

     // TODO IMPLEMENT YOUR SERVICE-TEST HERE

     LOGGER.info("CLASS : ServiceTest -- METHOD : execute -- END");
  }
```