

Projet Architectures Orientées Services sous
JEE
Gestion de la Notification d'Effets Indésirables

15/01/2017

Table des matières

1	Introduction	1
2	Présentation	1
2.1	Contexte	1
2.2	JavaEE	1
2.3	Spring	1
3	Technologies	2
3.1	Spring	2
3.1.1	Gradle	2
3.1.2	Maven 3	2
3.1.3	Ant	2
3.1.4	Spring Boot 1.4.2	2
3.1.5	Spring Security	3
3.1.6	Spring MVC 1.0.1	3
3.2	Tomcat	3
3.3	Thymeleaf 2.1.2	3
3.4	PostgreSQL 9.1	3
3.5	IntelliJ 2016.3	3
4	Vue d'ensemble de l'architecture du projet Spring	4
5	Données	4
5.1	Initialisation de la Base de Données	5
5.2	Diagramme relationnel	5
5.3	Liaison des Données	6
6	Les Entités	7
6.1	Description	7
6.2	Diagramme des EJB	8
7	Services	8
7.1	Description	8
7.2	Diagramme de Services	9
8	Gestionnaire de sessions	9

1 Introduction

Internet est au coeur du monde d'aujourd'hui, il est alors logique pour les structures permettant son emploi de s'agrandir et se complexifier. Nous nous centrons ici sur une des solutions trouvées pour répondre au besoin d'applications toujours plus complètes : **JavaEE** (Java Enterprise Edition).

2 Présentation

2.1 Contexte

Il est demandé, dans ce projet, de développer une application web permettant de gérer des notifications d'effets indésirables survenus à la suite de l'utilisation d'un ou de plusieurs produits cosmétiques : **NotifCosmoEI**. Ces notifications peuvent être déclarées par plusieurs types d'utilisateurs :

- Utilisateur quelconque
- Pharmacien
- Association de consommateurs
- Médecin
- Laboratoire de cosmétiques

Dans le document ci-présent nous allons décrire la solution technique et justifier l'emploi de certaines orientations technologiques et techniques.

2.2 JavaEE

JavaEE est une plate-forme fortement orientée serveur ayant pour but de développer et exécuter des applications distribuées. Elle se compose de deux grandes parties : spécifications et serveur d'application(s). L'intérêt non moindre est de permettre à des entreprises, par exemple, de développer des serveurs d'applications conformes aux spécifications, sans avoir la nécessité de re-développer les services existants. Elle permet donc de faciliter le développement d'applications avec une Architecture Orientée Service, mais pas seulement.

2.3 Spring

Spring est un framework libre, conçu pour la création et la définition de l'infrastructure d'une application Java. Son but est de faciliter le développement et les tests. L'architecture de ce framework se compose de plus de 20 modules pouvant être utilisés en fonction de la demande :

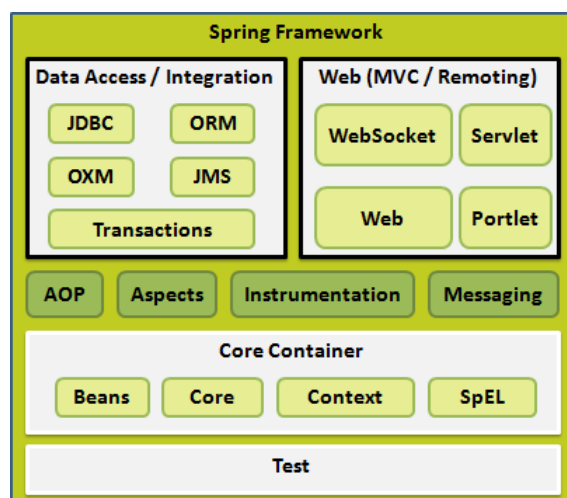


FIGURE 1 – Architecture du Framework Spring

Nous emploierons donc ce framework afin de répondre à la demande de développer une Application Web. En effet, notre intention est de nous former à un framework grandement reconnu dans le monde et à prendre en main une technologie qui sera utiliser pendant le projet Annuel TRAVEL. Grâce à la formation aux AOS pendant les cours et les travaux pratiques du parcours M2GIL, nous allons présenter une démarche descriptive alliant méthodologie et respect des conventions apprises.

3 Technologies

Dans cette section sera présenté l'intégralité des technologies utilisées dans la réalisation de l'Application Web NotifCosmoEI. Y seront présentés également les versions utilisées de ces mêmes outils.

3.1 Spring

Comme décrit plus haut nous utiliserons le framework Spring afin de développer notre application. Ce framework permet la liaison avec de nombreux outils complémentaires qui ont été également vus pendant le cursus de deuxième année de master.

Nous allons donc lister les principaux outils de développement utilisés par celui-ci et contextualiser leur emploi. Nous préciserons la version des outils en question également.



3.1.1 Gradle

Gradle est un moteur de production permettant de construire des projets Java. Il allie **Apache Maven** et **Apache Ant** pour en prendre uniquement leurs atouts. Ainsi il permet d'écrire des tâches de **build** dans un fichier **build** en utilisant le langage Groovy. Il est alors possible d'importer des tâches standards qui permettent de construire des programmes utilisant un ou plusieurs langages, Java dans notre cas.



3.1.2 Maven 3

Apache Maven est un outil pour la gestion et l'automatisation de production des projets Java. Dans notre cas il sera employé pour ses conventions simples et efficaces, de part Gradle. En effet Maven impose une arborescence et une énumération des fichiers du projet selon le concept de Convention plutôt que configuration.



3.1.3 Ant

Apache Ant est un logiciel visant à automatiser les opérations répétitives du développement de logiciel telles que la compilation, la génération de documents ou l'archivage. Utilisé par Gradle, sa tâche est de permettre la compilation, la génération, l'exécution et l'archivage sous forme distribuable (JAR, ...). Il faut savoir qu'il est très portable et flexible dans la description des tâches de construction.



C'est donc un indispensable pour fournir une application exécutable et portable.

3.1.4 Spring Boot 1.4.2

Spring Boot permet l'initialisation de nombreuses configurations automatisées :

- Initialisation du gestionnaire **DispatcherServlet** de Spring MVC
- Configuration du filtre d'encodage au sein des requêtes clients
- Configuration du routage des vues Spring afin de les situer et de les spécifier au sein du projet
- Intégration de la location des ressources statiques (.css, .js, ...)
- Configuration des ressources locales de bundles
- Intégration de l'exécution Tomcat sur lequel l'application sera déployée



— Configuration et création des gestion de pages d'erreurs (404, ...)

En outre, Spring Boot maintient l'intégralité de ces opérations pour nous.

3.1.5 Spring Security

Spring Security offre des services de sécurité complets pour des applications basées sur JavaEE. Il s'intègre nativement à un projet Spring et permet de mettre notamment l'accent sur l'authentification et l'autorisation, entre autre le contrôle d'accès, et bien d'autres. Il est utilisé ici pour conserver un très haut niveau de sécurité au sein de la spécification des Servlet JavaEE et de la spécification des EJB.



3.1.6 Spring MVC 1.0.1

Spring propose l'élaboration d'une application sur l'architecture Model View Controller grâce à Spring MVC. Nous utiliserons donc ce dernier afin de respecter les conventions vues en cours et pratiqués lors des travaux pratiques. C'est également sur ce contexte que se basent nombre de solutions logicielles et web, auxquelles nous sommes sensibilisé depuis le début de notre cursus.

3.2 Tomcat

Apache Tomcat est un conteneur web libre de servlets et JSP JavaEE. C'est un serveur HTTP, il répond alors parfaitement à la problématique du projet. Nous l'utiliserons et l'intégrerons directement en tant que serveur embarqué sur notre application par le biais de Spring (notamment par l'emploi du Spring Boot vu précédemment).



3.3 Thymeleaf 2.1.2

Thymeleaf est un moteur de template utilisé dans un environnement web pour la génération de vue pour les applications web MVC. Nous justifions sa présence par sa simplicité d'utilisation et son extension facilité. De plus il répond aux besoins soulevés par le modèle MVC.

Un autre argument vient conforter ce choix, c'est sa grande popularité dans la communauté, nous permettant d'avoir accès à un large panel d'informations lors de problèmes rencontrés.



3.4 PostgreSQL 9.1

PostgreSQL est un système de Gestion de Base de Données relationnelle et objet (SGBD). C'est un outil libre qui s'est forgé une réputation sur son comportement stable et ses possibilités de programmation étendues. Il possède également de nombreuses interfaces utilisateurs permettant le debug rapide et l'accessibilité facilitée.

Encore une fois il sera déployé par l'intermédiaire de Spring Boot et quelques pré-requis seront nécessaire (cf. Guide d'utilisation).



3.5 IntelliJ 2016.3

Pour le développement du projet nous avons utilisé l'environnement IntelliJ IDEA. En effet celui-ci est complet et est utilisé dans de nombreuses entreprises spécialisées dans les applications web. Ses fonctionnalités en JavaEE sont très attractives et comprennent des greffons sur chacun des outils vus plus haut (notamment Spring, PostgreSQL et Tomcat).



Une fois la base de données choisie il nous faut définir le modèle relationnel sur lequel nous baser, c'est-à-dire la composition de nos tables et du stockage des données.

5.1 Initialisation de la Base de Données

Une fois choisi PostgreSQL nous nous sommes attelé à l'élaboration d'une solution relationnelle dans le but de répondre au mieux au projet. La configuration par défaut que nous avons défini est :

Nom d'utilisateur PostgreSQL	Mot de Passe	Nom Base de données
postgres	postgres	jeedb

Grâce à l'utilisation de Spring il est très facile de changer ces paramètres, il suffit d'aller dans : `src/main/resources/application.properties`
Dans ce fichier nous trouvons facilement l'url de connexion à la base de données, le nom d'utilisateur et également le mot de passe.

5.2 Diagramme relationnel

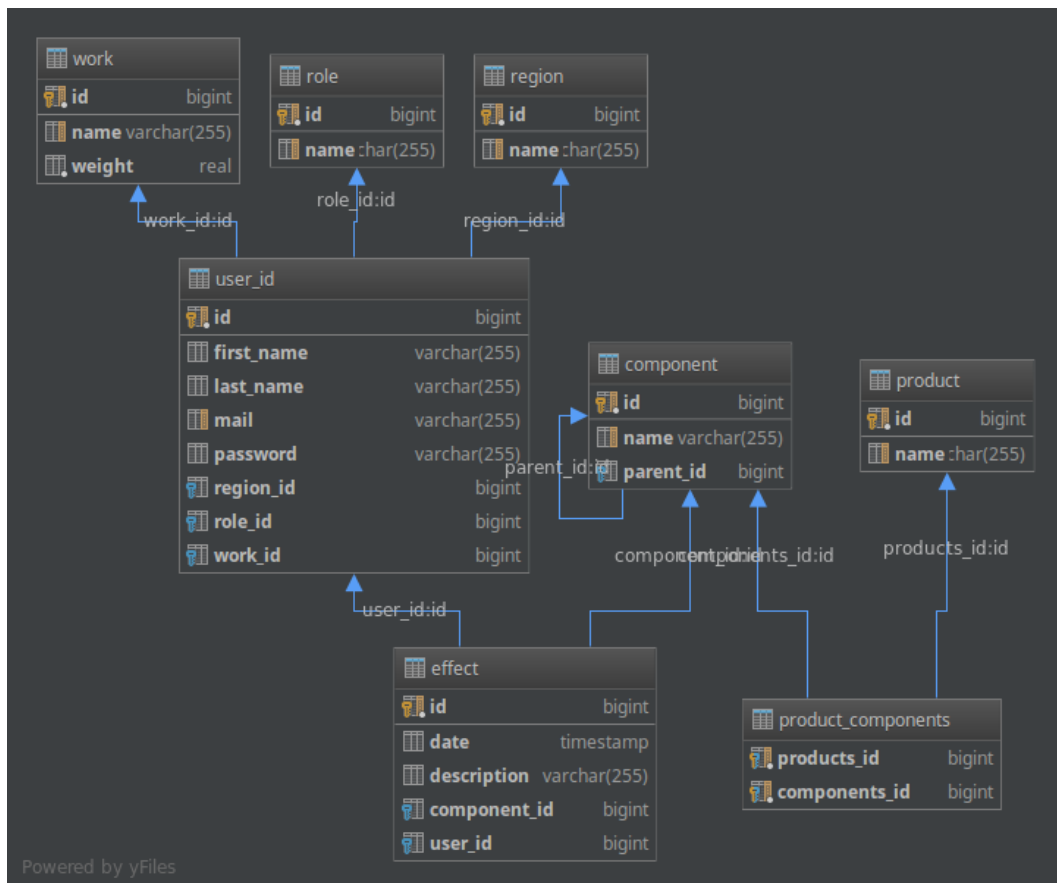


FIGURE 3 – Diagramme Relationnel

5.3 Liaison des Données

Afin de faciliter la liaison de la couche données avec la couche métier nous avons utilisé l'API offerte par Spring : Spring-Data-JPA. En effet celle-ci vise à améliorer la mise en oeuvre de la couche d'accès aux données en réduisant considérablement l'effort d'écriture du code d'implémentation en particulier pour les méthodes CRUD et de recherche.

La notion centrale dans Spring-Data-JPA est la notion de Repository. Un repository est une interface à écrire par les développeurs. Cette interface requiert la déclaration des méthodes utiles d'accès aux données.

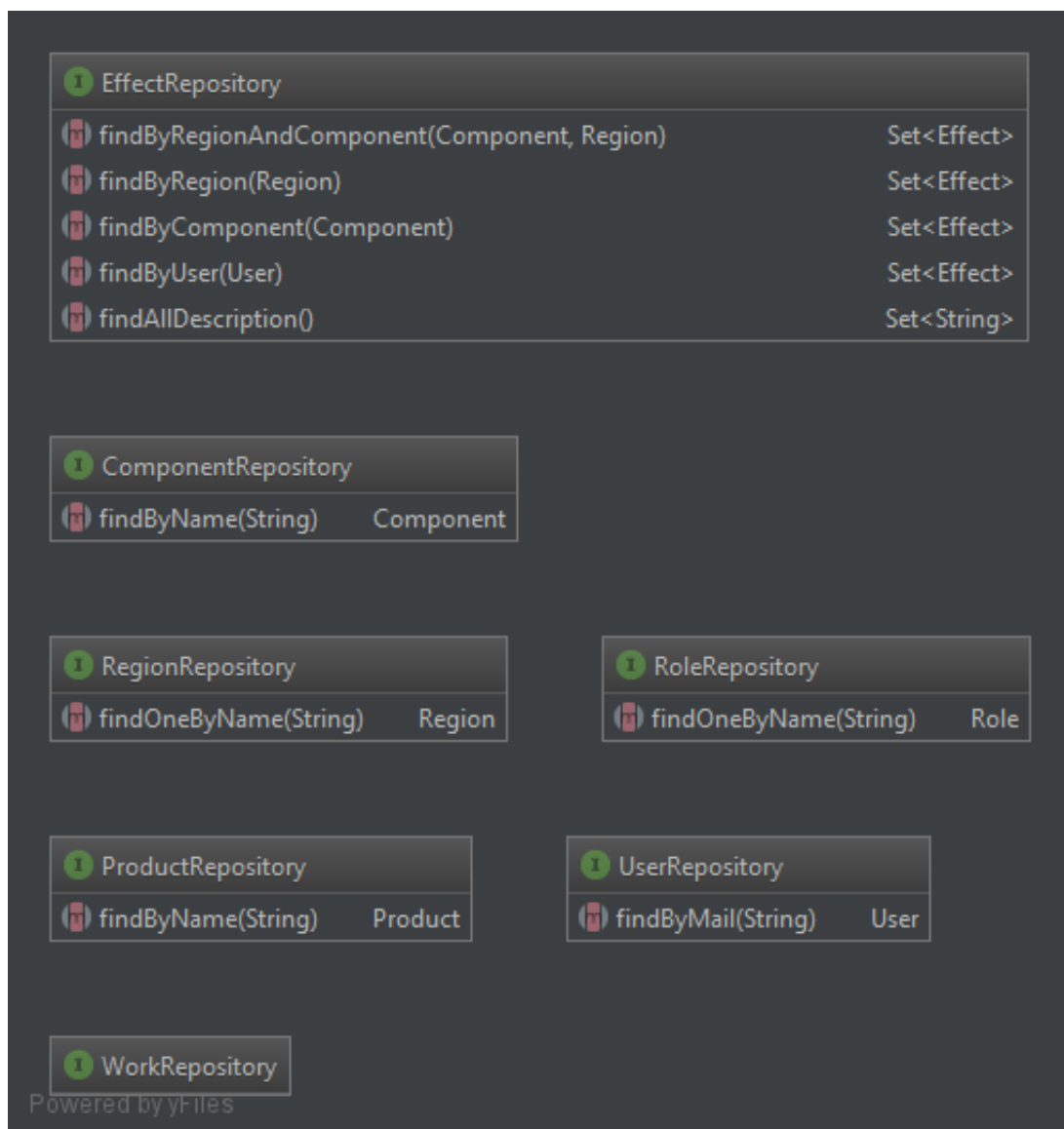
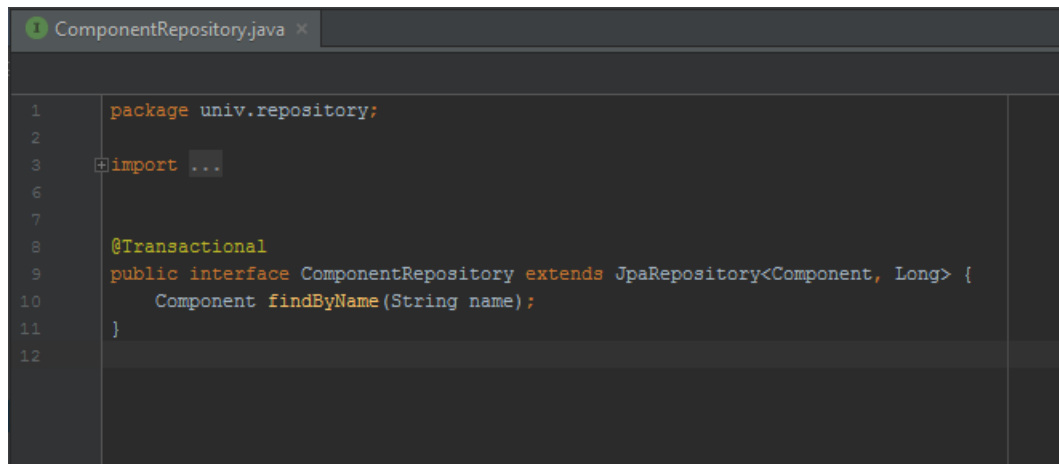


FIGURE 4 – Liste des différents Repository

Enfin, un exemple de déclaration de repository :



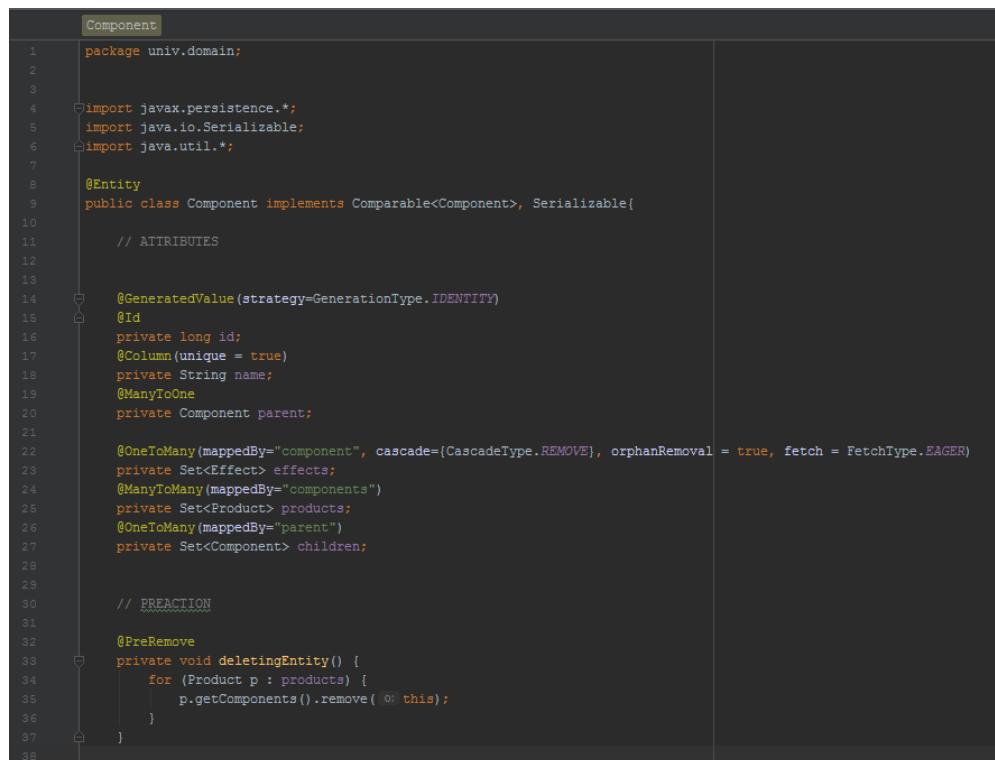
```
1 package univ.repository;
2
3 import ...
4
5
6
7
8 @Transactional
9 public interface ComponentRepository extends JpaRepository<Component, Long> {
10     Component findByName(String name);
11 }
12
```

FIGURE 5 – Exemple de Repository

6 Les Entités

6.1 Description

Dans un projet Spring, une entité se réfère étroitement à un entité que nous pourrions trouver dans un projet JavaEE natif. En effet nous privilégions les annotations dans le but de simplifier la lecture et l'affectation des tâches.



```
1 package univ.domain;
2
3
4 import javax.persistence.*;
5 import java.io.Serializable;
6 import java.util.*;
7
8 @Entity
9 public class Component implements Comparable<Component>, Serializable{
10
11     // ATTRIBUTES
12
13
14     @GeneratedValue(strategy=GenerationType.IDENTITY)
15     @Id
16     private long id;
17     @Column(unique = true)
18     private String name;
19     @ManyToOne
20     private Component parent;
21
22     @OneToMany(mappedBy="component", cascade={CascadeType.REMOVE}, orphanRemoval = true, fetch = FetchType.EAGER)
23     private Set<Effect> effects;
24     @ManyToMany(mappedBy="components")
25     private Set<Product> products;
26     @OneToMany(mappedBy="parent")
27     private Set<Component> children;
28
29
30     // PREACTION
31
32     @PreRemove
33     private void deletingEntity() {
34         for (Product p : products) {
35             p.getComponents().remove( @ this);
36         }
37     }
38 }
```

FIGURE 6 – Exemple d'Entity

6.2 Diagramme des EJB

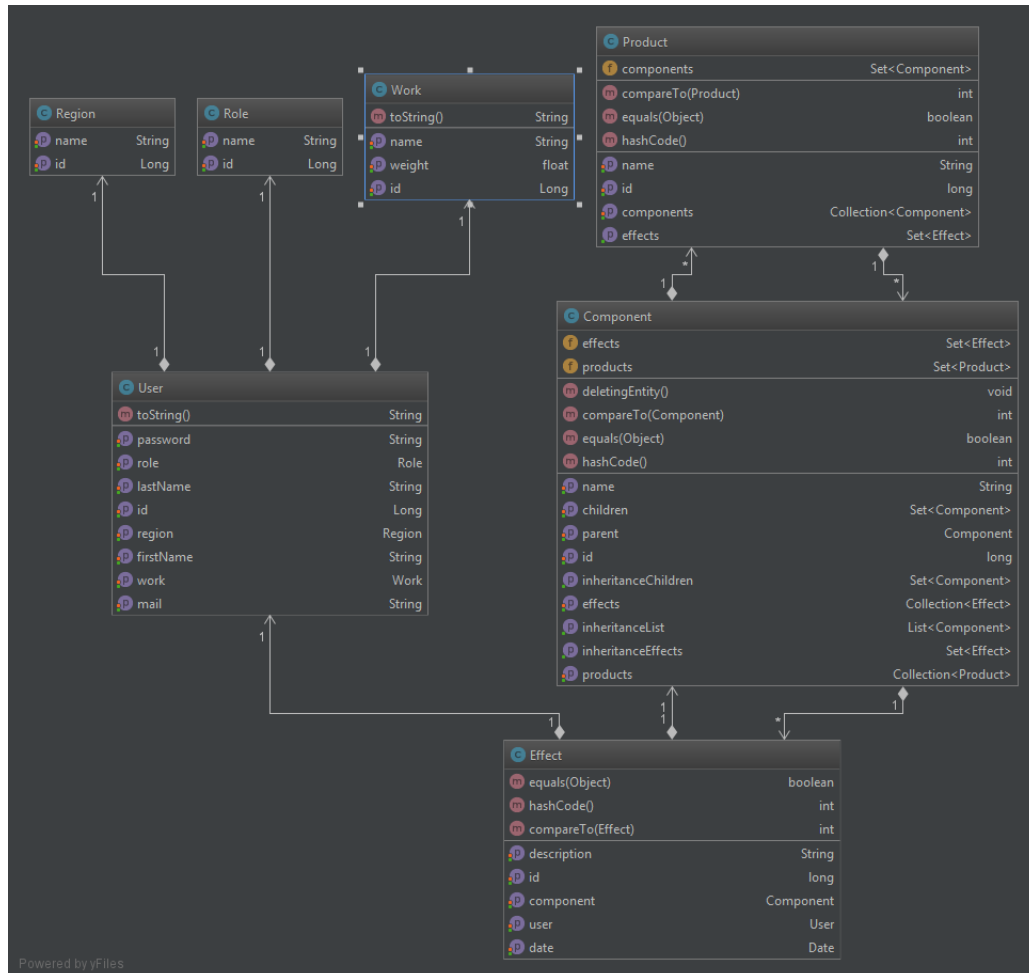


FIGURE 7 – Liste des différents Repository

7 Services

7.1 Description

Puisque nous sommes dans une Architecture Orientée Service, nous utilisons Spring afin de répondre à ce pré-requis. Dans cette couche, les services peuvent communiquer entre eux et permettre une meilleure productivité. Il faut noter que cette couche utilise les Repository dans le but de contextualiser convenablement les entités. Les services vont donc s'occuper de lier la couche métier avec la couche données dans le but de déclarer un contrat d'utilisation. Ce contrat permettra à nos contrôleurs de connaître les conditions d'utilisation d'un service et de ne préoccuper que de sa définition.

Il est à noter également qu'un service a pour but d'être réutilisé par la suite et est donc prévu pour persister au-delà d'un unique projet, d'où l'intérêt de son utilisation.

7.2 Diagramme de Services

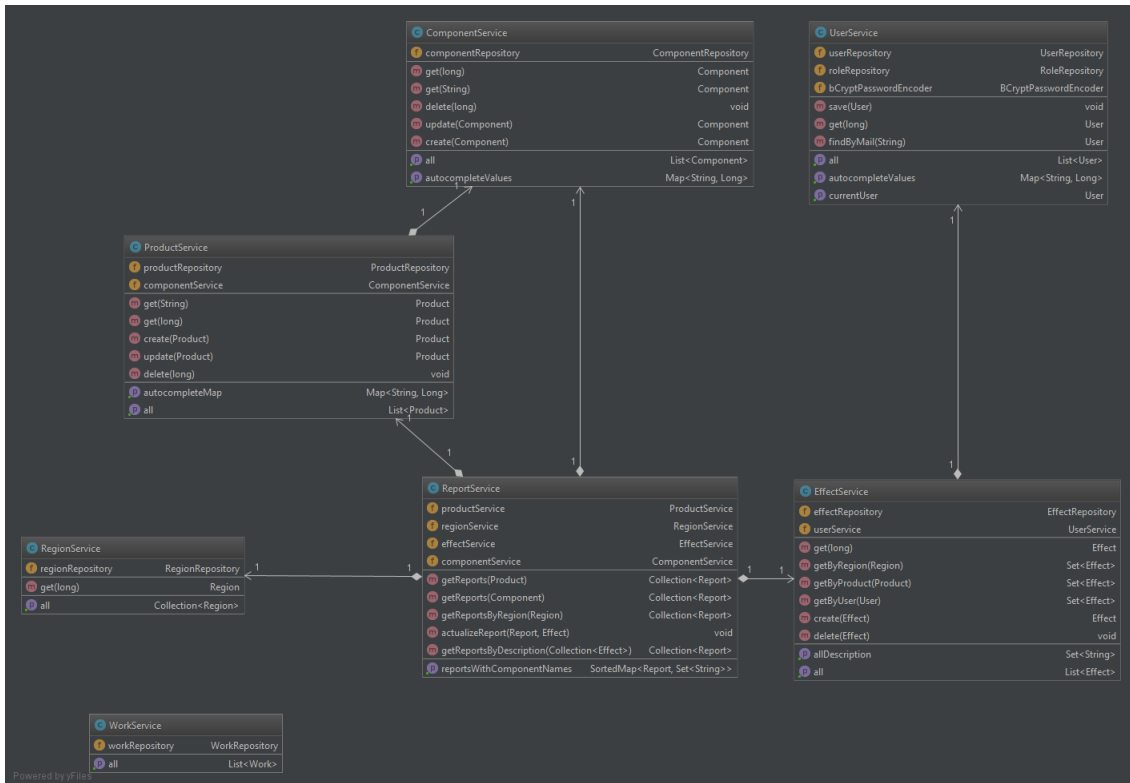


FIGURE 8 – Liste des différents Repository

8 Gestionnaire de sessions

Dans le but de gérer la sécurité au sein de notre application nous avons décidé d'utiliser Spring Security. En effet celui-ci nous permet de rapidement contrôler les différentes authentifications et connexions. Grâce à un fichier de mapping et à la définition d'un fichier de configuration, nous pouvons rapidement et efficacement définir le contrat de sécurité de notre application.

```

1 package univ.config;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Bean;
5 import org.springframework.context.annotation.Configuration;
6 import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
7 import org.springframework.security.config.annotation.web.builders.HttpSecurity;
8 import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
9 import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
10 import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
11
12 import javax.sql.DataSource;
13
14 @Configuration
15 @EnableWebSecurity
16 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
17
18     @Autowired
19     @Override
20     DataSource dataSource;
21
22     @Bean
23     public BCryptPasswordEncoder bCryptPasswordEncoder() {
24         return new BCryptPasswordEncoder();
25     }
26
27     @Override
28     public void configure(AuthenticationManagerBuilder auth) throws Exception {
29         auth.jdbcAuthentication().dataSource(dataSource)
30             .usersByUsernameQuery(
31                 "select user_id, password, role_id from user_id where mail=?")
32             .authoritiesByUsernameQuery(
33                 "select user_id, mail, role.name from user_id, role where role_id=user_id.role_id and user_id.mail=?");
34     }
35 }

```

FIGURE 9 – Liste des différents Repository