**KELLNER Maxime 29/12/2019**

**ATTIA Michael**

**SOAMALALA Ravaka**

**TP noté de Config Poste de Travail**

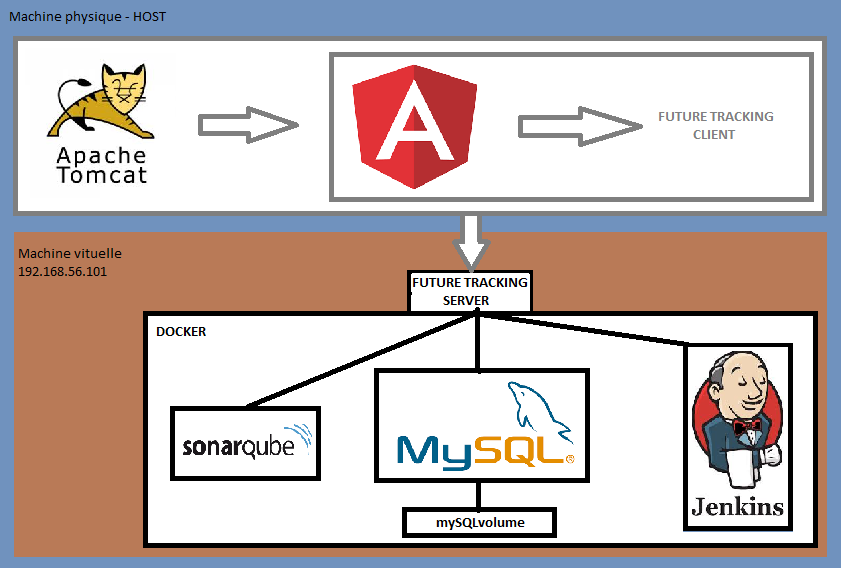
Contexte :

Future Tracking est une startup qui a mis au point des systèmes innovants pour générer des livres de recette au format PDF à partir d’une base de données de recettes. Elle a obtenu une importante levée de fonds et doit embaucher de nouveaux développeurs.

Afin d’accélérer la mise en place et faciliter le démarrage des nouveaux arrivants, le Directeur technique demande à l’équipe aux compétences Full Stack qui a une bonne expérience des outils et process de développement, d’organiser au plus vite l’installation et la configuration des nouveaux postes de développement.

L’équipe chargée de la création de l’environnement de développement doit réaliser des sprints d’une semaine.

**ARCHITECTURE N-TIERS**



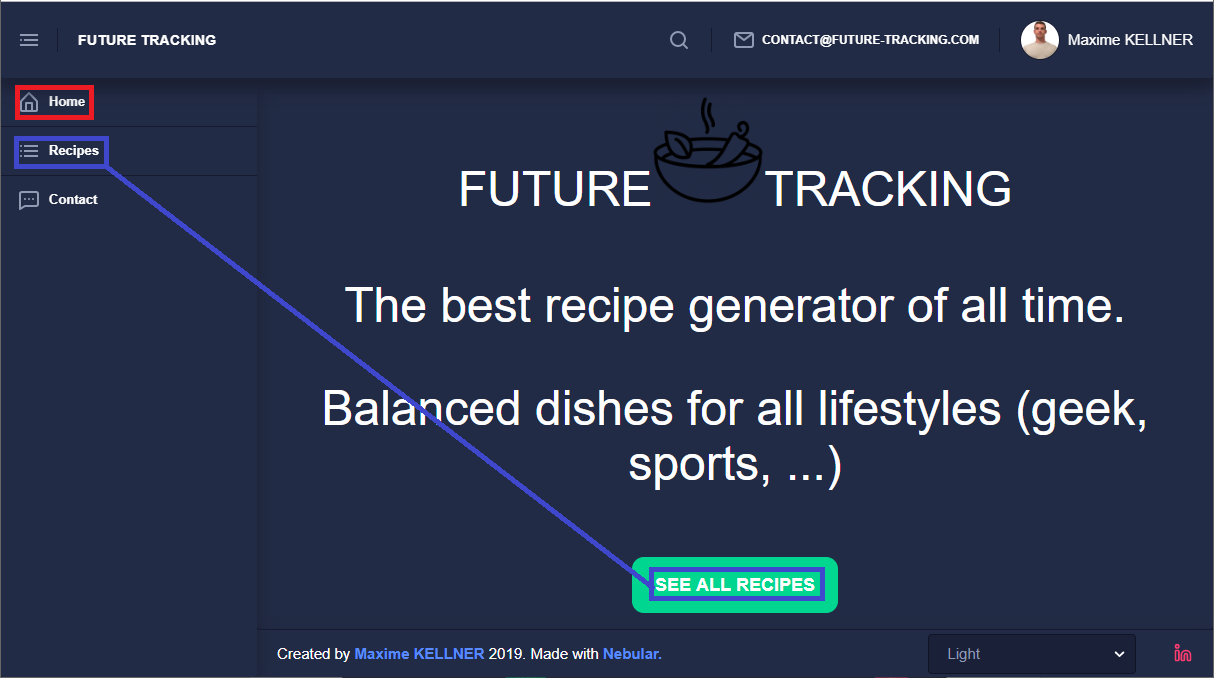
Sur notre machine physique, nous aurons le client « FutureTracking Client » créé avec Angular et qui correspond au site de recette, celui-ci sera déployé sur Apache Tomcat.

Ensuite, nous créerons une machine virtuelle (IP : 192.168.56.101) qui permettra de simuler l’architecture N-tiers en ayant l’application Backend lancé sur un autre serveur (FutureTracking-BackEnd).

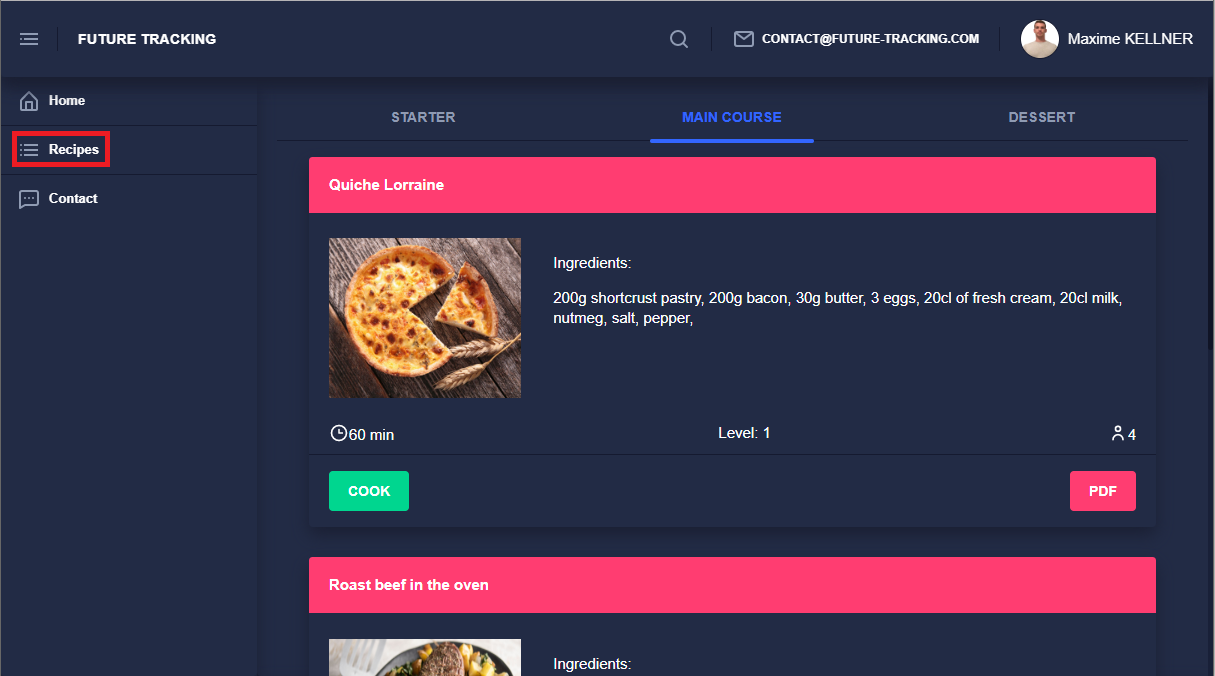
L’outil docker sera également installé sur la machine virtuelle dont l’os est Debian 9 et grâce auquel nous créerons les containeurs « Jenkins », « Sonarqube » et « MySql » ainsi que le volume « mySQLvolume » pour la persistance des données utilisées dans MySql.

**APERCU DE L’APPLICATION WEB FUTURE TRACKING (app client)**

* *Page d’accueil*

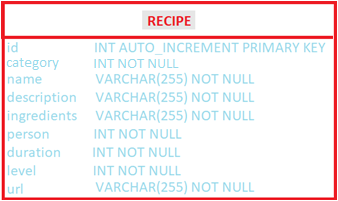
**

* *Page des recettes*



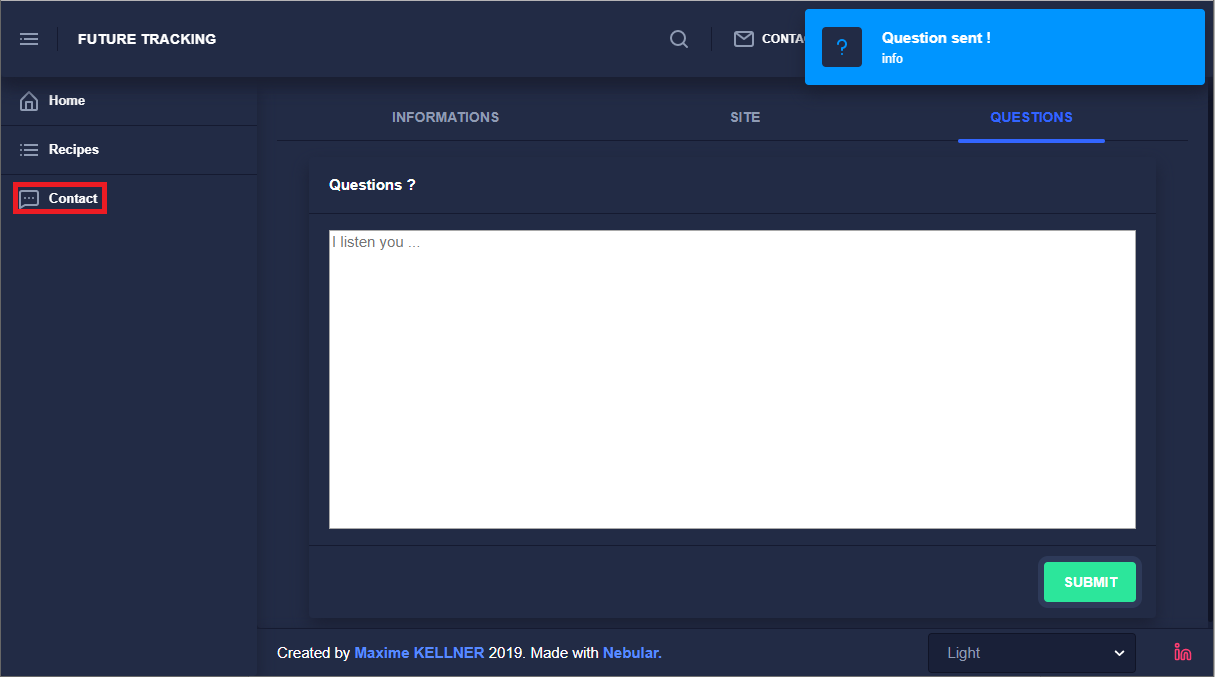
Lors de l’appel à l’onglet « Recipes » sera délivré toutes les recettes contenues dans la base de données à laquelle nous accèderons grâce aux webservices développer grâce à Springboot et Maven en langage JAVA dans l’application côté backend.

Une recette est de la forme :



Nous aurions pu avoir une table « INGREDIENT » et référencer les ingrédients via une clé étrangère dans RECIPE (en sachant qu’il aurait même fallu une table « RECIPE\_INGREDIENT » comme une recette possède plusieurs ingrédients). Cependant, ici, la modélisation de notre BDD n’est pas l’objectif et pour un gain de temps nous utiliserons cette table uniquement.

* *Page pour la prise de contact*

**

**INSTALLATION DU LOGICIEL DOCKER SUR DEBIAN (machine virtuelle)**

1 – Dans un premier temps, exécuter les commandes « apt-get update » et « apt-get upgrade » pour être mis à jour.

2 – Exécuter la commande « **curl https://releases.rancher.com/install-docker/18.03** »

**INSTALLATION DES CONTAINEURS ET VOLUMES**

Ce projet aura besoin de :

* JENKINS
* SONARQUBE
* MySQL
  + Un VOLUME pour la persistance de nos données

Les manipulations seront à effectuer en tant qu’administrateur, pour cela taper la commande « su » et saisissez votre mot de passe de root.

1. **JENKINS**

*Etape 1 :* Installation de Jenkins

Exécution de la commande :

***docker run -d -p 8080:8080 -p 50000:50000 -v jenkins\_home:/var/jenkins\_home jenkins/jenkins:lts***

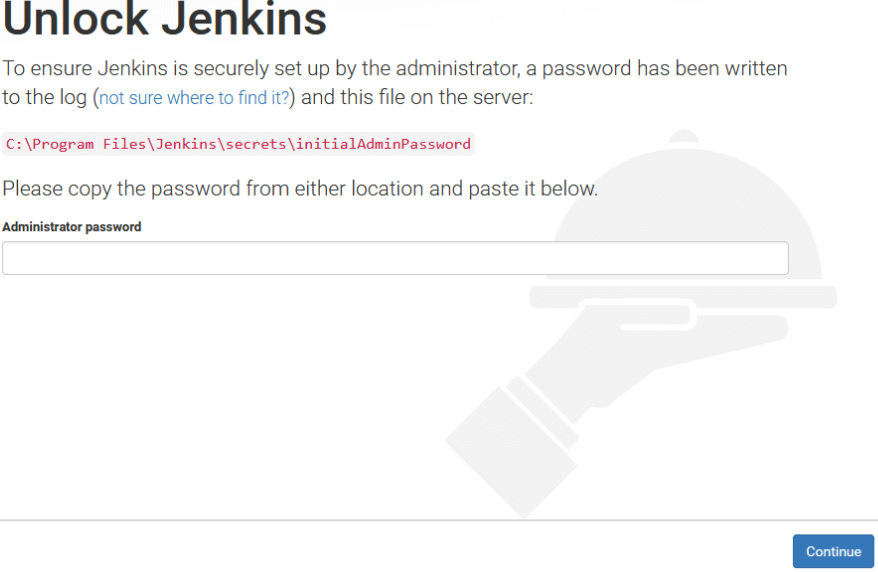
(-v jenkins\_home crée automatiquement le volume jenkins\_home associe au path qui suit)

Cette commande va induire le « pull » de l’image dans une première phase et ensuite lancer l’image avec la redirection sur le port 8080.

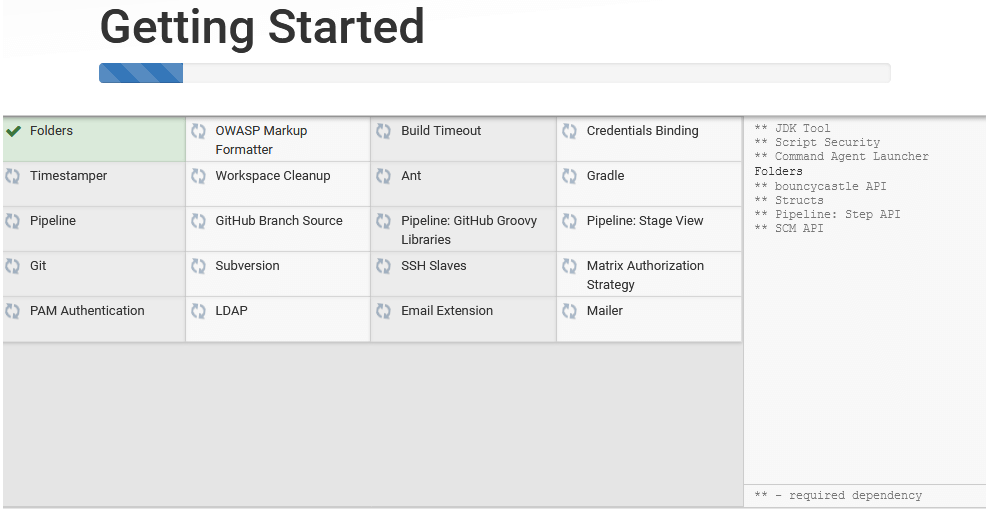
Par conséquent, nous pourront accéder à l’interface Jenkins depuis notre machine physique via l’adresse *192.168.56.101(adresse machine virtuelle) :8080*

*Etape 2 : Configuration de Jenkins*

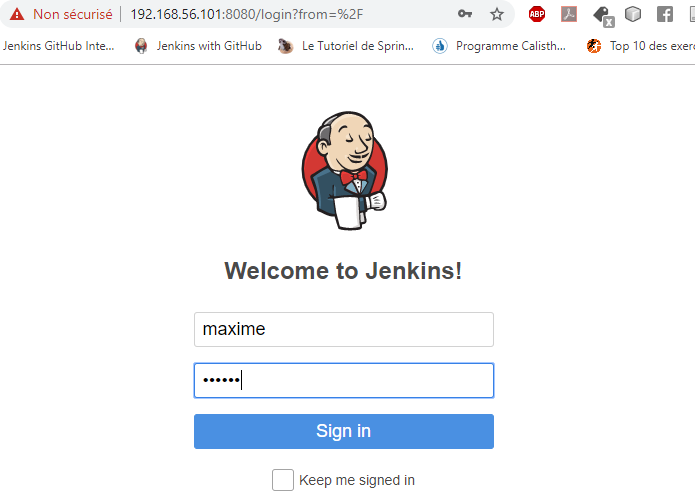
Tapez l’adresse 192.168.56.101:8080 dans la barre de navigation du navigateur voulu et il y aura une étape de configuration de Jenkins lors de la première connexion.



Récupérer alors le mot de passe (situé dans la console normale avec -ti ou depuis le path indiqué avec -ti) puis copié le et cliquez sur « Continue ».

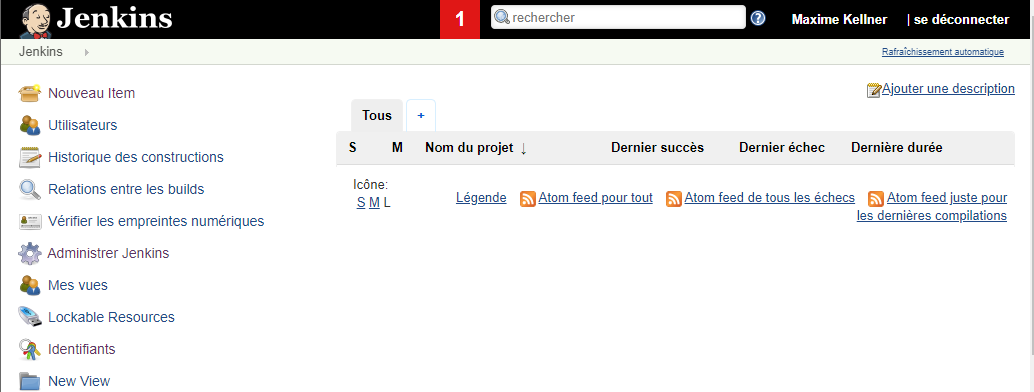


L’assistant de configuration vous proposera de télécharger tous les plugins recommandés, faite donc ce choix et patientez.



Vous pourrez, par-la-suite, saisir un nom d’utilisateur et mot de passe que vous utiliserez dans la suite du projet pour l’utilisation de Jenkins. Dans notre cas, il s’agira de « maxime / maxime » (pensez à cocher la case pour rester connecté).

Après connexion, vous arriverez donc à l’écran suivant :



*Etape 3: Installation des prérequis pour le client*

Lorsque nous allons réaliser notre build à l’aide de « pipeline », il nous faudra pouvoir exécuter certaines commandes depuis Jenkins.

Etant un projet « Angular », Jenkins aura par conséquent besoin de nodejs et @angular/cli mais aussi de Sonarqube-Scanner et de zip pour la création de l’artefact pour pouvoir réaliser la tâche d’après-build.

* Nous devons nous connecter en tant que root au Shell du containeur sur lequel nous avons Jenkins

Grâce à un « ***docker ps*** », nous savons que le nom de notre container est **«***fervent\_raman***»,** nous exécutons donc la commande suivante : ***docker exec -ti –user root fervent\_raman /bin/bash***

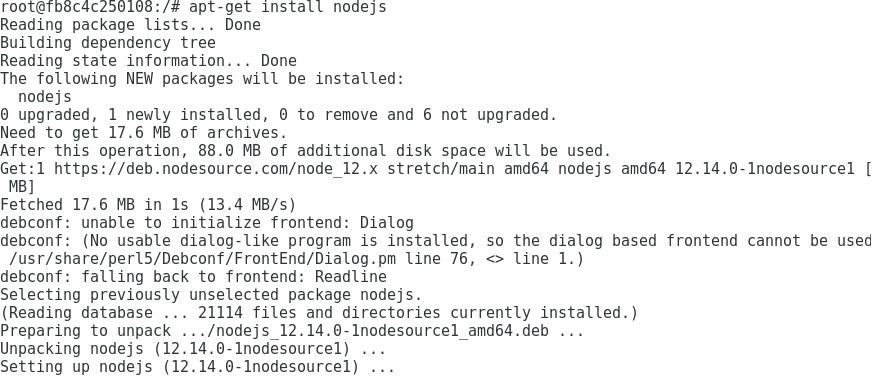
***NodeJS***

Nous avons besoin de NodeJS PPA avant de commencer, exécuter alors :

1. apt-get install curl software-properties-common
2. curl -sL https://deb.nodesource.com/setup\_12.x | bash

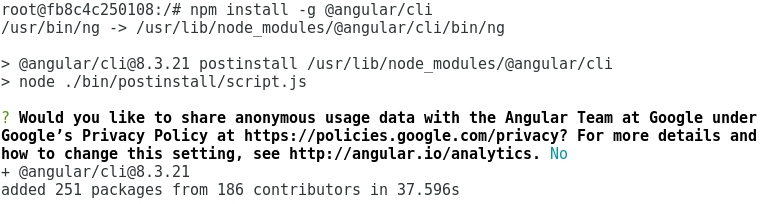
Avant d’exécuter :

1. ***apt-get install nodejs***

****

***@angular/cli***

Exécuter «**npm install -g @angular/cli**»

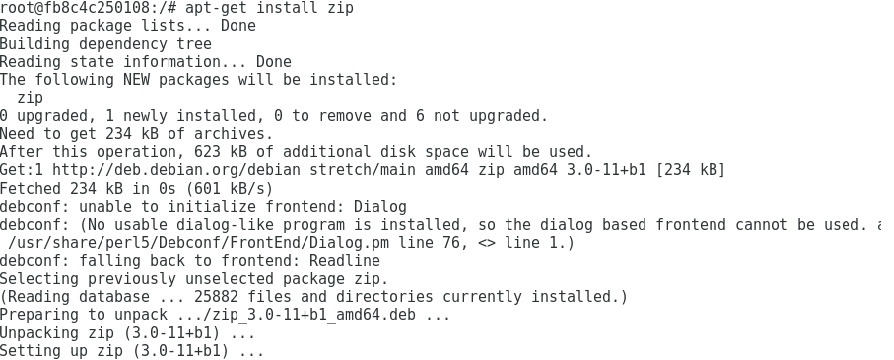


**Sonarqube-Scanner**

Exécuter « **npm install -g sonarqube-scanner** »

**ZIP**

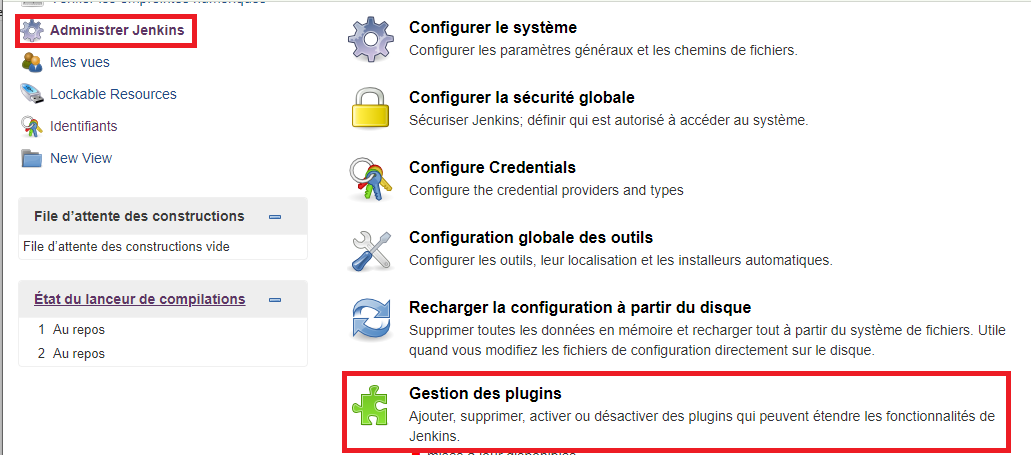
Exécuter « **apt-get install zip** »

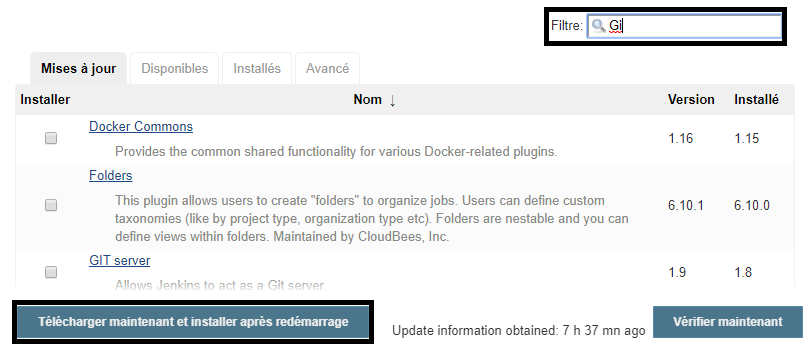


*Etape 4: Installation des prérequis pour l’application serveur*

Nous aurons besoin d’installer Maven pour cela, il suffira d’exécuter la commande « **apt-get install maven** ».

*Etape 5 : Paramétrage du credential pour pouvoir communiquer avec le serveur GitHub.*

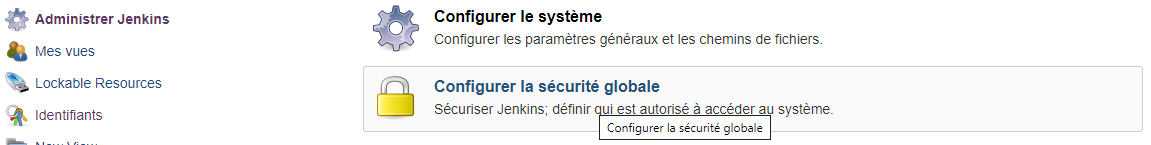
Afin de pouvoir se connecter et récupérer les ressources sur GitHub depuis Jenkins, il sera nécessaire de configurer une clé identifiant et d’installer le Plugin GitHub.

Il faudra recherche le plugin en saisissant son nom dans la barre de recherche :

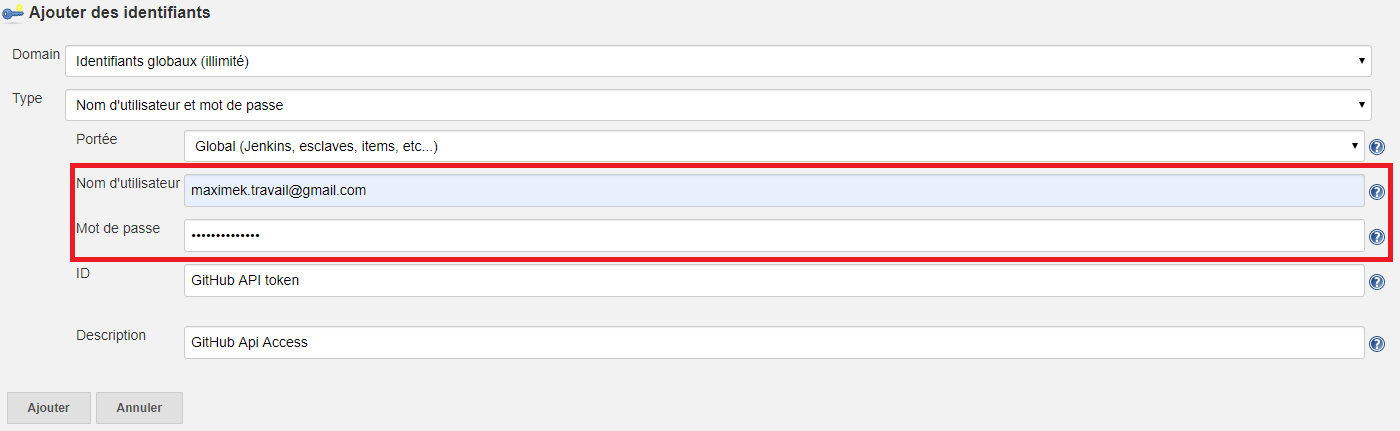
De sélectionner le plugin une fois affiché



Et de cliquer sur « Installer sans redémarrage » ou « Télécharger maintenant et installer après redémarrage » cela dépend de l’état de la machine Jenkins à cet instant.

Maintenant, nous devons à nouveau revenir dans « Administrer Jenkins », puis cliquer sur « Configurer Credentials » pour ajouter une clé d’authentification sur GitHub

En cliquant sur « Ajouter », nous aurons la fenêtre suivante, dans laquelle il faudra rentrer vos identifiants de connexions à GitHub dans la zone rouge puis nous cliquons sur « Ajouter »



Pour ce qui est des autres champs, tout doit être exactement comme sur la capture d’écran !

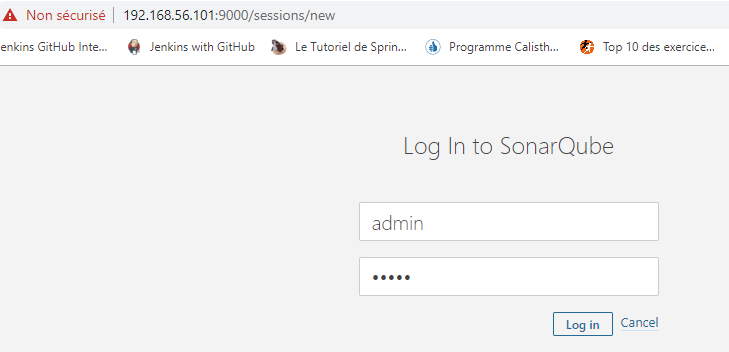
Voilà ! JENKINS EST PRÊT !

1. **SONARQUBE**

Pour installer Sonarqube, nous n’aurons besoin d’exécuter que la commande suivante :

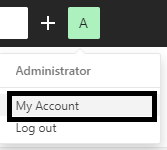
«***docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube***»

Ainsi, vous aurez accès à Sonarqube sur l’adresse 192.168.56.101:9000 grâce à la redirection de port et pourrez-vous connecter par défaut avec le couple admin/admin (login/motdepasse)

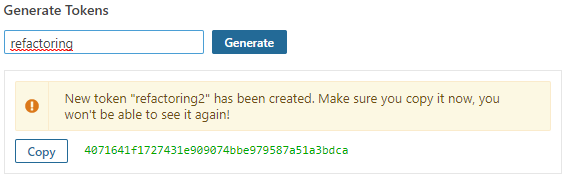


Ensuite, nous créons un Token afin de pouvoir se connecter à SonarQube depuis nos application (token à bien conserver)

1. Cliquez sur « MyAccount »

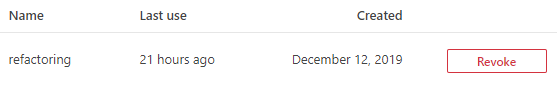


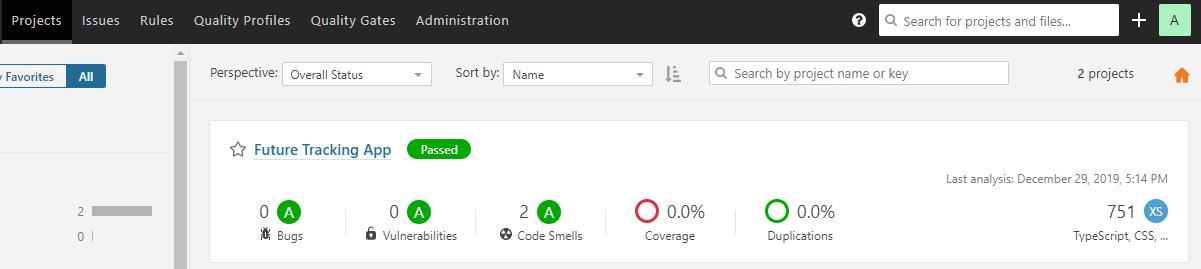
1. Taper « refactoring ou autre chose » puis cliquez sur « Générer »



Il sera recommandé de précieusement conserver la clé générée car celle-ci n’est pas récupérable après actualisation de la page !

Et voila, a tout moment vous pourrez consulter vos token, et les supprimer si necéssaire :





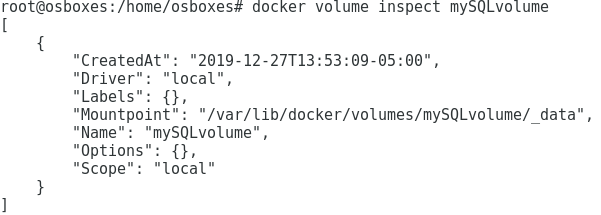
SONARQUBE EST PRÊT !

1. **Volume pour MySQL (impératif de le faire avant de créer le containeur de MySQL)**

Exécuter la commande « ***docker volume create --name mySQLvolume*** » pour créer le volume que nous utiliserons pour MySQL.

« ***docker volume ls*** » pour vérifier la présence du volume



***« docker volume inspect mySQLvolume*** » pour vérifier la disponibilité du volume

LE VOLUME EST PRÊT !

1. **MySQL**

*Etape 1 : Installation du containeur MySQL*

Exécuter la commande suivante :

***« docker run --name mysql\_futureTracking -e MYSQL\_ROOT\_PASSWORD=root -d -p 3306:3306 -v mySQLvolume:/var/lib/mysql mysql »***

Pour savoir à quel path associer un volume il suffit de vérifier le DockerFile de mysql avec le tag latest dans notre cas et de voir que le path ***:/var/lib/mysql*** est le path de stockage.

On fait un « ***docker ps*** » pour vérifier l’exécution de notre containeur :



MySQL EST PRÊT !



**INSTALLATION ET CONFIGURATION DE L’OUTIL « SQLDEVELOPPER »**

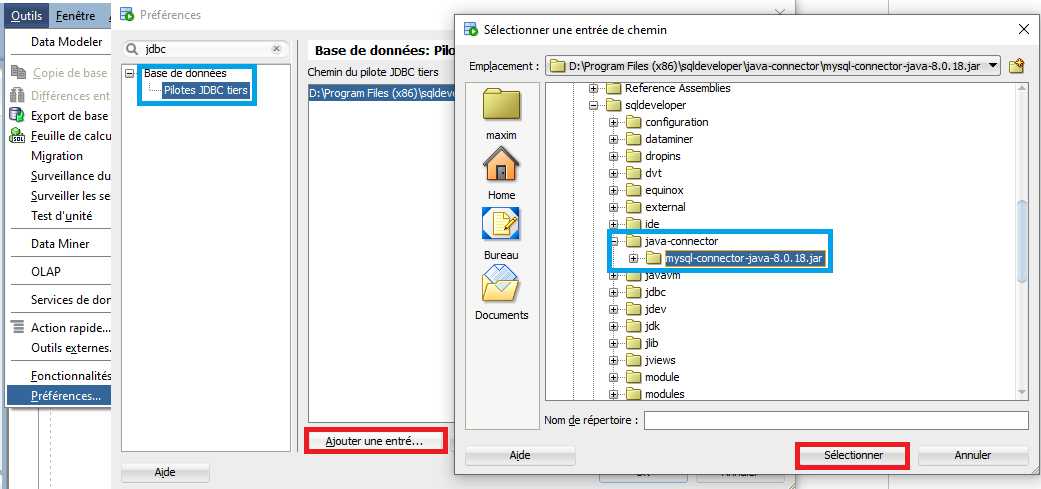
*Etape 1 : Installation du pilote de gestion des connections MySQL*

MySQL n’a pas d’interface web pour gérer la BDD, il faut passer par un outil comme SQLDevelopper (sur lequel on ajoute la library jdbc-connector pour pouvoir prendre en charge les connexions vers une BDD MySQL)

Lien de SQLdevelopper : <https://www.oracle.com/fr/tools/downloads/sqldev-v192-downloads.html>

Lien de jdbc-connector : <https://dev.mysql.com/downloads/connector/j/> (plateform independant)

Puis ouvrez SQL > Onglet « Outils » > Sélectionnez « Préférences » > paramètre « Pilotes JDBC tiers » puis il suffit d’ajouter le .jar du jdbc-connector comme indiqué sur la capture ci-dessous !



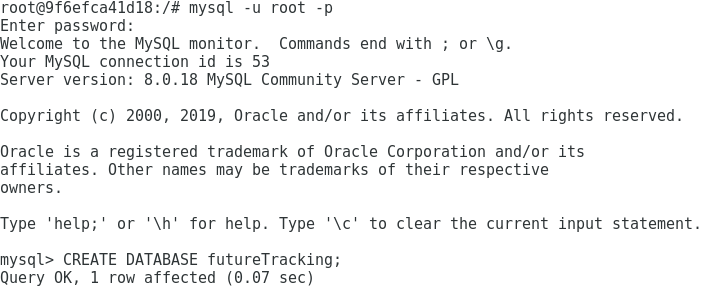
*Etape 2 : Etablissement d’une « instance » de base de données et création d’un utilisateur pour le projet FutureTracking*

1. Se connecter au Shell du docker MySQL : ***« docker exec -it mysql\_futureTracking /bin/bash »***



1. Créer une BDD dédiée au projet FutureTracking

Pour cela, nous aurons besoin de nous connecter à MySql en tant que root avec la commande **« mysql -u root -p »** et nous créerons ensuite la BDD avec la commande **« CREATE DATABASE futureTracking ; »**



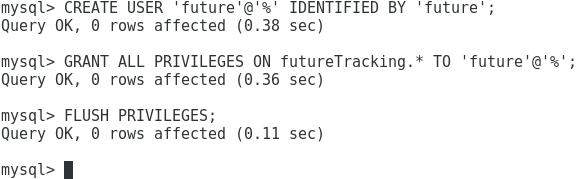
Base de données CREEE !

1. Créer un utilisateur et lui donner tous droits sur la BDD « futureTracking »

Toujours en étant connecté en tant que root au service MySQL, il suffit de créer un utilisateur avec la commande « **CREATE USER ‘future’@’%’ IDENTIFIED BY ‘future’** », et de lui donner tous les droits sur tout le contenu de la BDD grâce à la commande « **GRANT ALL PRIVILEGES ON ‘futureTracking. \*’ TO ‘future’@’%’ ;** »

Ensuite on enregistre la configuration des droits données à notre nouvel utilisateur :

« **FLUSH PRIVILEGES ;** »

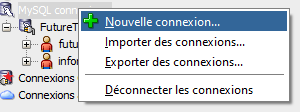


A cet instant, nous avons un utilisateur « future » attribué pour le projet « Future Tracking » et une BDD qui lui est propre !

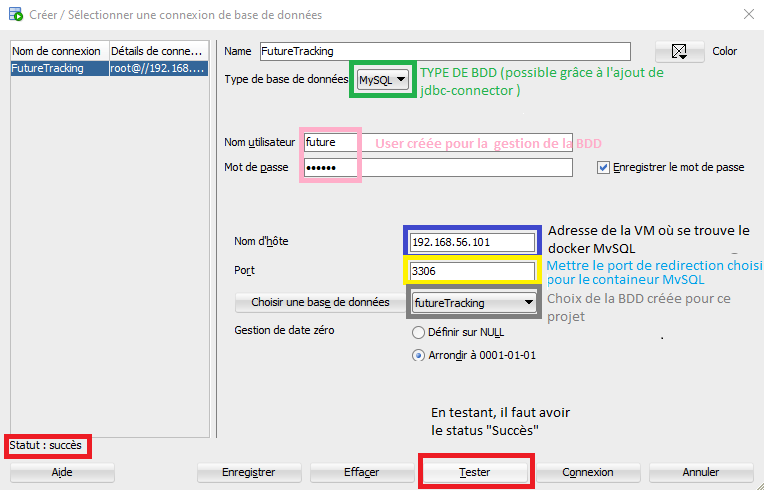
Il ne manque plus qu’à se connecter à cette BDD à l’aide de SQLdevelopper que nous avons déjà configuré pour accepter une connexion à MySQL.

*Etape 3 : Connexion à la BDD MySQL via SQLdevelopper*

Pour créer une connexion, clique-droit sur MySQL connexion puis de faire « Nouvelle connexion »

**

Une nouvelle fenêtre s’affiche alors et dans laquelle, après avoir choisis « MySQL » dans le type de BDD voulue, nous rentrerons les informations suivantes :



Et enfin, cliquez sur « **Connexion** » !

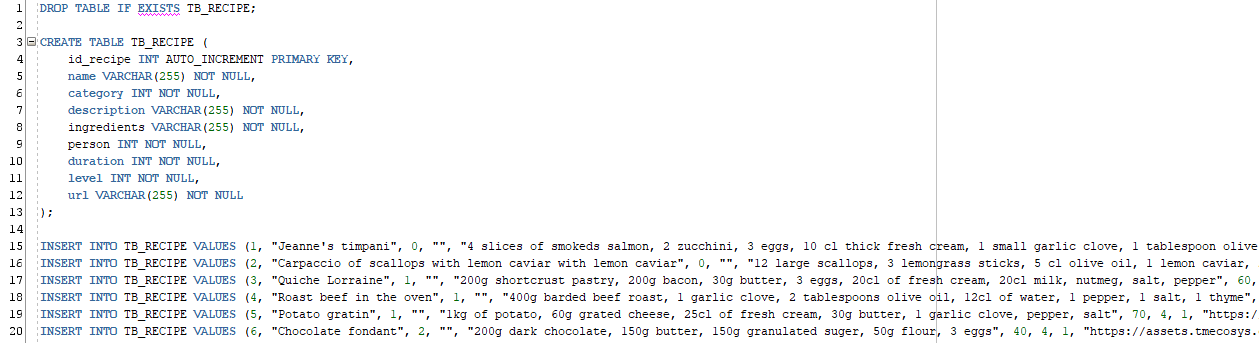
Nous sommes enfin prêts à utiliser pleinement MySQL !

**CREATION DE LA TABLE RECIPE via SQLdevelopper**

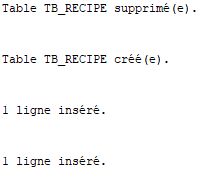
En effectuant un double-clique (gauche) sur la base de données futureTracking qui sera affichée une fois la connexion établie, il vous sera possible d’accéder à un éditeur de texte qui permettra l’exécution des commandes SQL.

Ce script sera très simple et nous permettra de réaliser le stockage des données de notre projet, voici donc le script que nous insérerons pour la création de la table et l’insertion des données

Script :





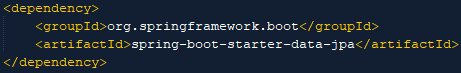
Exécution du script avec la commande « **F5** » sur votre clavier ou en cliquant sur et si tout se passe bien vous devrez avoir ceci :

**PROJET « FutureTracking-BackEnd »**

1. **Création du projet**

Je souhaite donc créer une application me permettant d’appeler des webservices qui me fournirait les données en BDD et retournerai un JSON, etc…

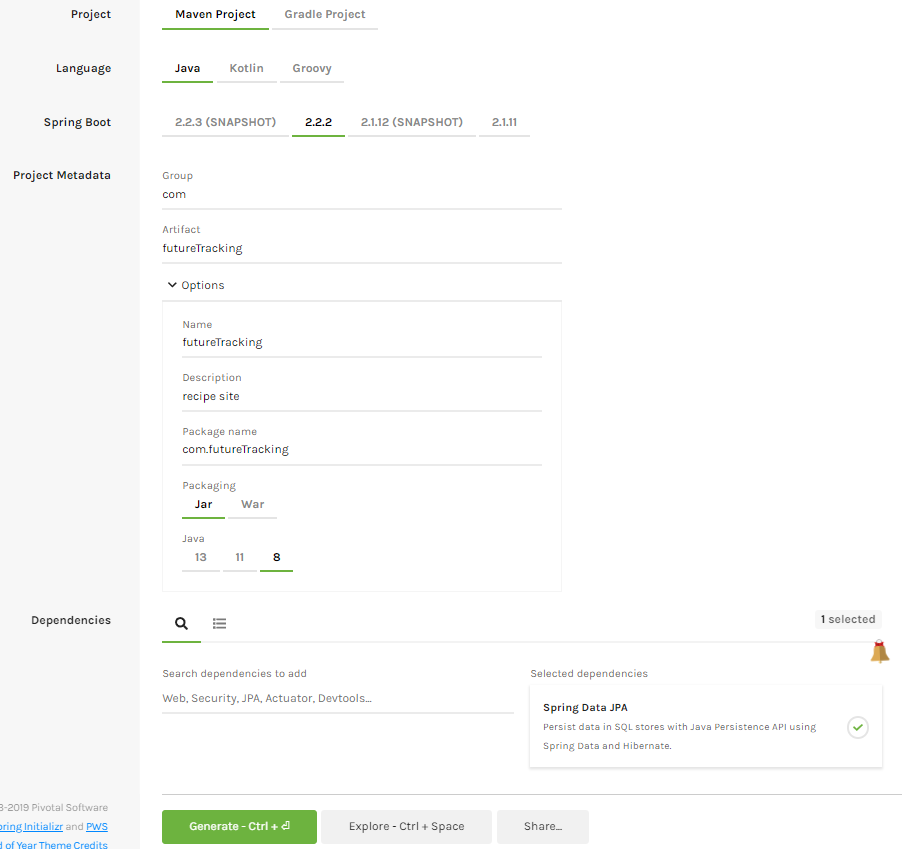
Nous allons donc générer un projet Springboot Maven utilisant JAVA et le plugin pour utiliser Hibernate:



Mais il est préférable de passer par le site <https://start.spring.io/> qui initialisera un projet voulu (Maven / Gradle) avec la version de java voulue, etc…

Après avoir rempli le formulaire, il vous sera possible de générer un zip du projet initialisé et d’ouvrir ce projet par exemple avec NetBeans dans notre cas

Voici comment remplir le formulaire en n’omettant pas d’ajouter le plugin pour Hibernate :



1. **Gestion de version (GITHUB)**

Il nous faudra par la suite, ajouter le projet à GitHub afin d’avoir un gestionnaire de version et de pouvoir bénéficier du Team collaboration.

Pour cela, l’outil « Git Bash » sera nécessaire ! Pour le télécharger : <https://gitforwindows.org/>

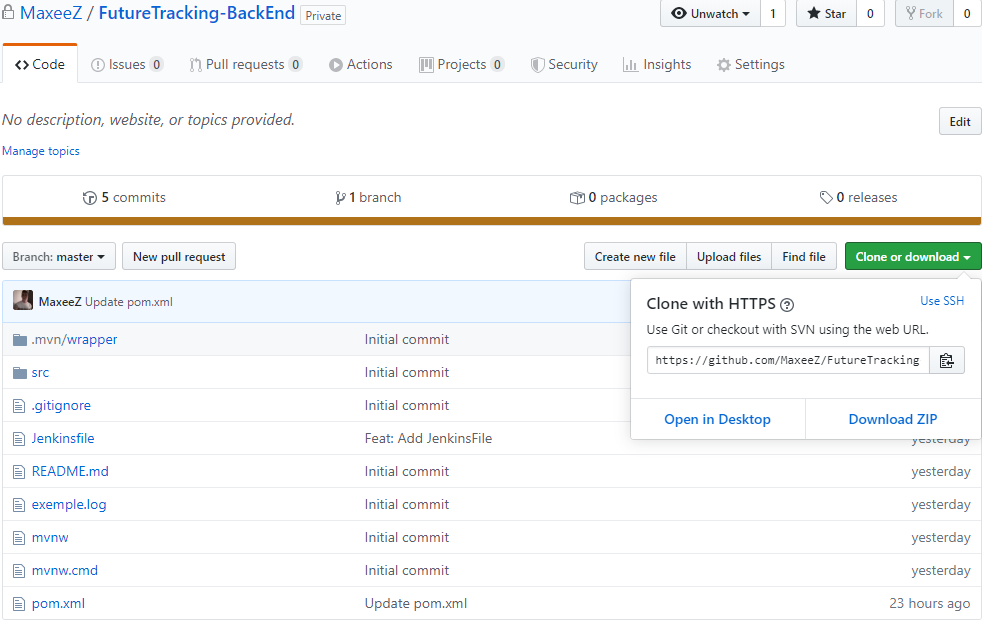
(Exécutez le fichier et suivez les étapes)

Une fois installé, rendez-vous dans le répertoire où se trouve le projet « FutureTracking-BackEnd » et réaliser dans l’ordre les commandes suivantes :

1. Aller sur GitHub, créer un nouveau projet « FutureTracking-BackEnd » vide, puis récupérer le lien de clone <https://github.com/MaxeeZ/FutureTracking-BackEnd.git>
2. Dans le terminal Git Bash, à votre première utilisation, réaliser les commandes suivantes :
   1. git config --global user.name "nom GitHub" (MaxeeZ)
   2. git config --global user.email email@deGitHub (maximek.travail@gmail.com)
3. Dans le terminal Git Bash, à la racine du répertoire, effectuer la commande « **git init** »
4. Effectuer la commande :

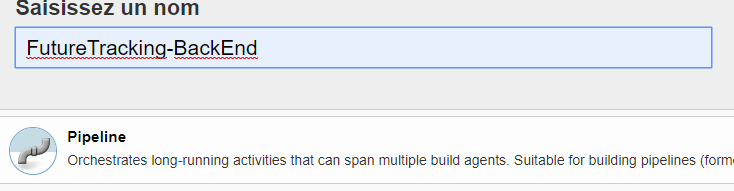
**« git remote add origin** [**https://github.com/MaxeeZ/FutureTracking-BackEnd.git**](https://github.com/MaxeeZ/FutureTracking-BackEnd.git) »

1. Effectuer la commande : « **git add .** »
2. Effectuer la commande : **« git commit -m « Initial Commit** » // pour initialiser le répertoire avec toutes nos ressources
3. Effectuer la commande : « **git push -u origin master** »
4. **Vérifier** que le répertoire GitHub est bien initialisé !



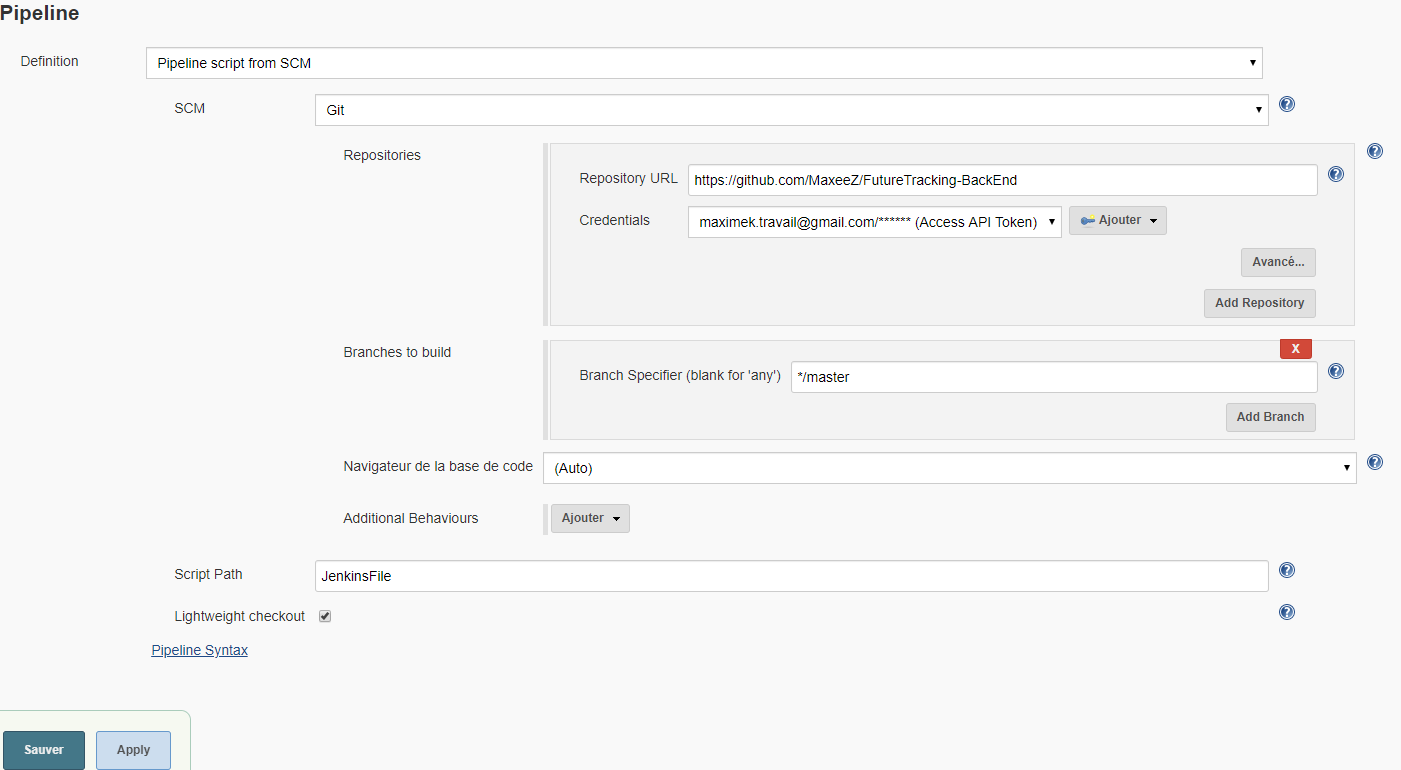
1. **Intégration du projet dans Jenkins depuis GitHub**
2. Ajouter un projet dans jenkins



Cliquez sur puis tapez nom « FutureTracking-BackEnd » et sélectionner « Pipeline » pour le nouvel item créé.

Après avoir cliqué sur « OK », il faudra se rendre à la section « Pipeline », puis indiqué qu’il s’agira de la construction d’une build à partir d’un JenkinsFile qui sera récupéré depuis le gestionnaire de version GitHub, pour cela il faudra renseigner l’adresse du projet GitHub que nous avons initialisé tout à l’heure !

Pour pouvoir accéder à l’API GitHub en rapport avec notre compte, nous utiliserons l’identifiant qui est lié à notre compte GitHub et enfin il n’y aura plus qu’à valider la configuration.



Le résultat :



La build est pour l’instant au rouge car il recherche un fichier de test non construit encore car il n’y a pour l’instant aucun test.

1. **JENKINSFILE**

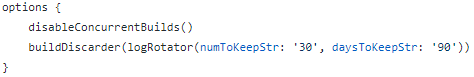
Nous utiliserons la machine « maître » par défaut et aucun slave pour l’exécution de nos builds :



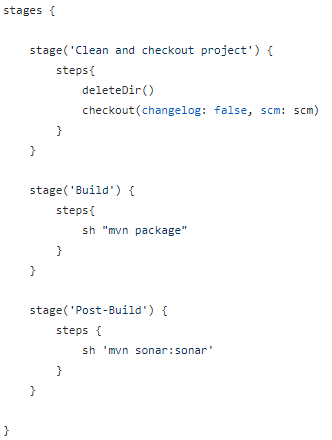
Nous vérifierons tous les 5 minutes le répertoires GitHub afin de réaliser de nouvelles builds en cas d’évolution / modification sur le répertoire :



Nous paramétrons une durée de vie de 90 jours à la build avant suppression :



Durant le build :

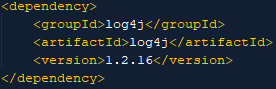


Nous aurons une première étape de nettoyage du répertoire de build de Jenkins.

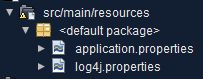
Ensuite nous effectuerons la build grâce à la commande **« mvn package »** qui permettra également d’exécuter tous les tests.

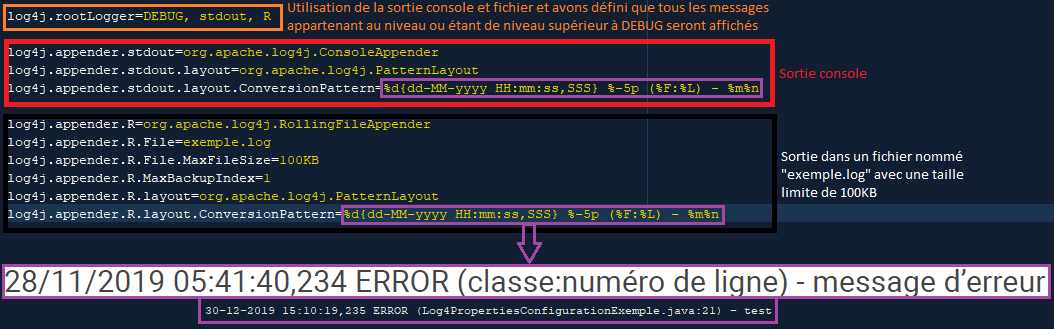
Après la build, nous enverrons le code sur Sonarqube

1. **LOG4J**

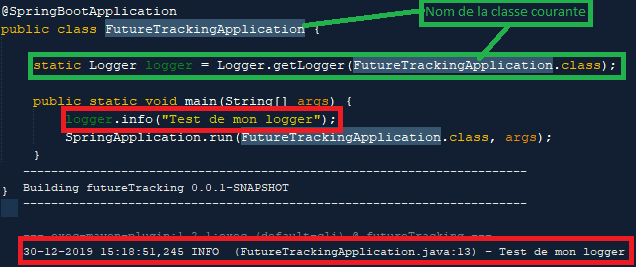
Pour ajouter Log4J, il faudra ajouter dans les dépendances, celle de log4j :

Ensuite, pour configurer Log4J, nous aurons besoin de créer un fichier nommer « log4j.properties » (fichier à créer dans src/main/resources) dans lequel nous inscrirons les propriétés suivantes :

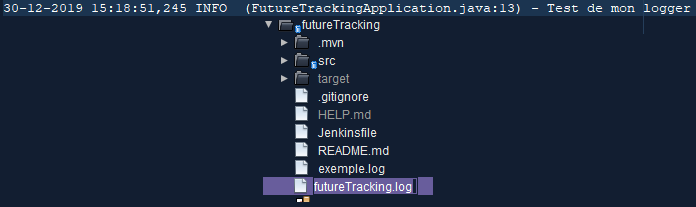




Pour chaque utilisation dans une classe, il faudra créer un membre static « logger » qui sera propre à la classe où il sera utilisé !

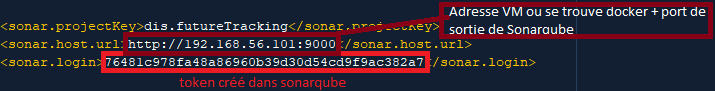


Et résultat avec le fichier.log :



1. **SONARQUBE**

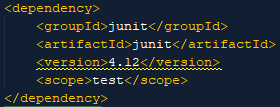
Afin de pouvoir utiliser la commande « mvn sonar : sonar » il faudra paramétrer SonarQube dans le projet Maven, pour cela nous nous servirons du fichier ***« pom.xml »*** et plus précisément de la section**« properties »**

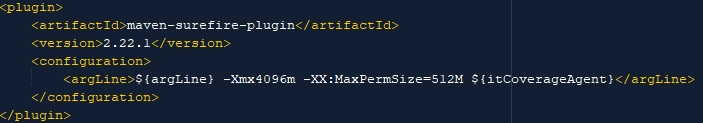


Localisation des fichiers tests :

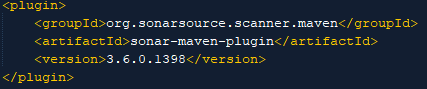


Ici pour les tests je souhaiterai utiliser **junit** et **maven-surefire ainsi que d’autre outils en complément (utilisation du clean de mvn, etc…)**, il va falloir inclure d’autres plugins et dépendances pour cela :

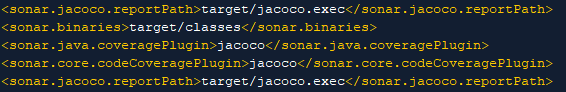




Utilisation du scanner de sonarqube :

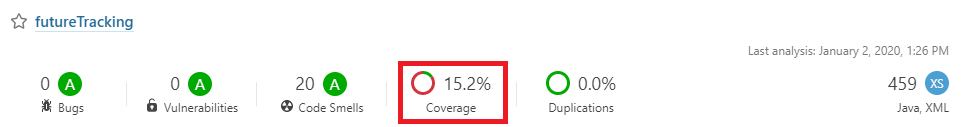


Je voudrai également avoir la couverture de code, pour cela il sera utile d’utiliser « **jacoco** » qui permet le calcul du rapport de couverture de code que Sonarqube ne calcule pas !



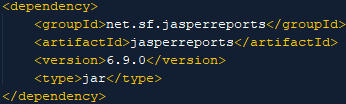


Dans l’installation du plugin maven-surefire, il ne faut pas oublier la ligne « **argLine** », car cela peut complètement gêner le fonctionnement de jacoco (je n’ai pas encore trouvé la réponse) il me semble qu’il s’agit sans doute de la manière dont sont écris les résultats !

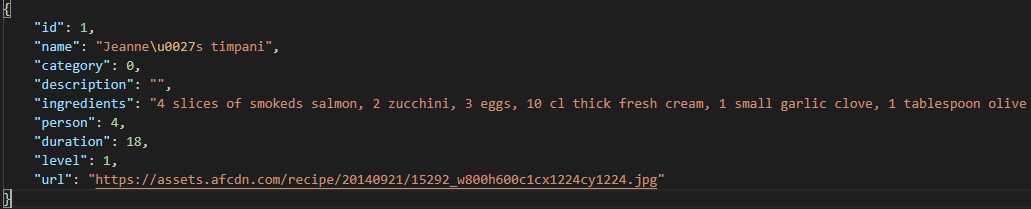


1. **JASPER**

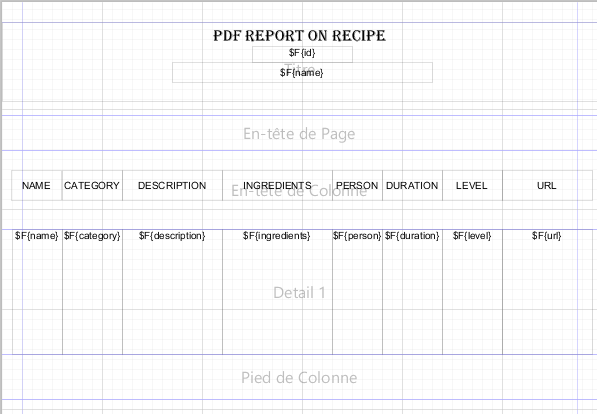
Pour installer le plugin, il suffira d’installer dans les dépendances le code suivant :

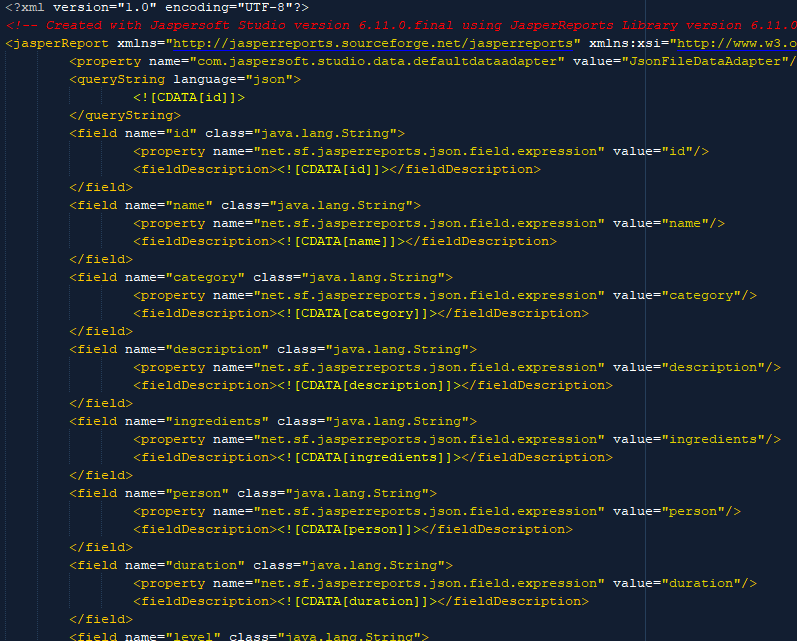


Il faudra dans un premier temps, déterminer les données que nous voulons utiliser lors de la génération d’un fichier PDF pour extraire une recette.

Voici un fichier JSON qui représente l’architecture de l’objet d’une recette

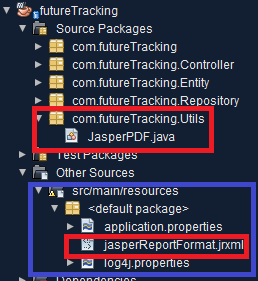
En suivant ce tutoriel : <https://www.youtube.com/watch?v=UG1RU393FtE>, il sera possible de générer un fichier que nous nommerons « jasperReportFormat.**jrxml »** qui contiendra les lois propres à la manière de rédiger le rapport effectué par Jasper y compris le design du fichier PDF depuis le fichier JSON donné ici en exemple.

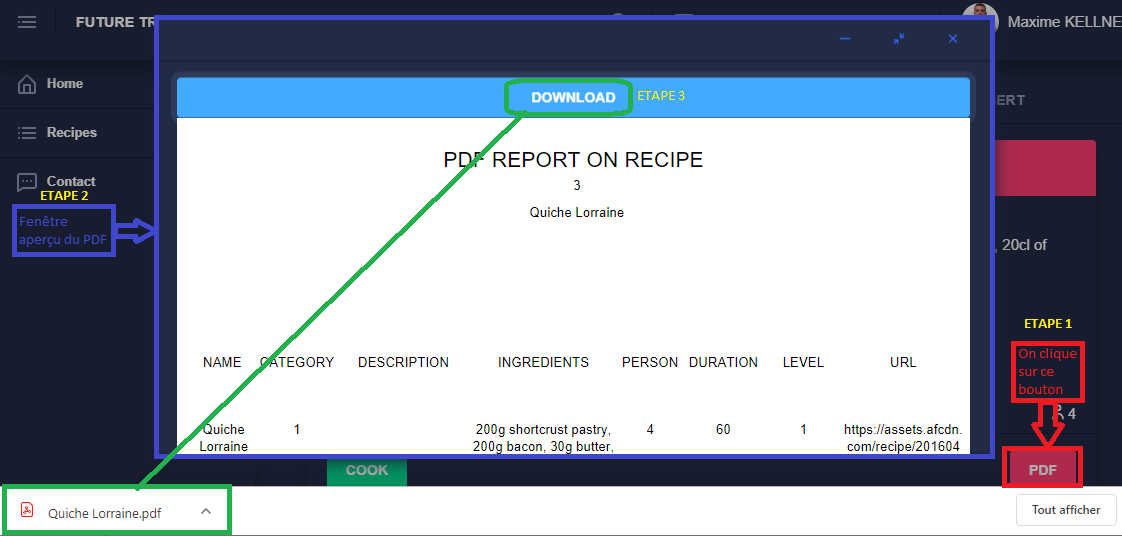
Ainsi, en suivant ce tutoriel, nous devrions obtenir le design suivant :

Et qui une fois compilé, donnera le fichier « **\*.jrxml** » (extrait du fichier capturé) suivant :

Ce fichier contient une première section qui présente les données que nous allons fournir et dans la deuxième section il s’agira du formatage (design, police, etc…) du fichier PDF.

Le format de notre fichier « jasperReportFormat.jrxml » sera stocké en tant que ressource du projet afin d’être utilisé :

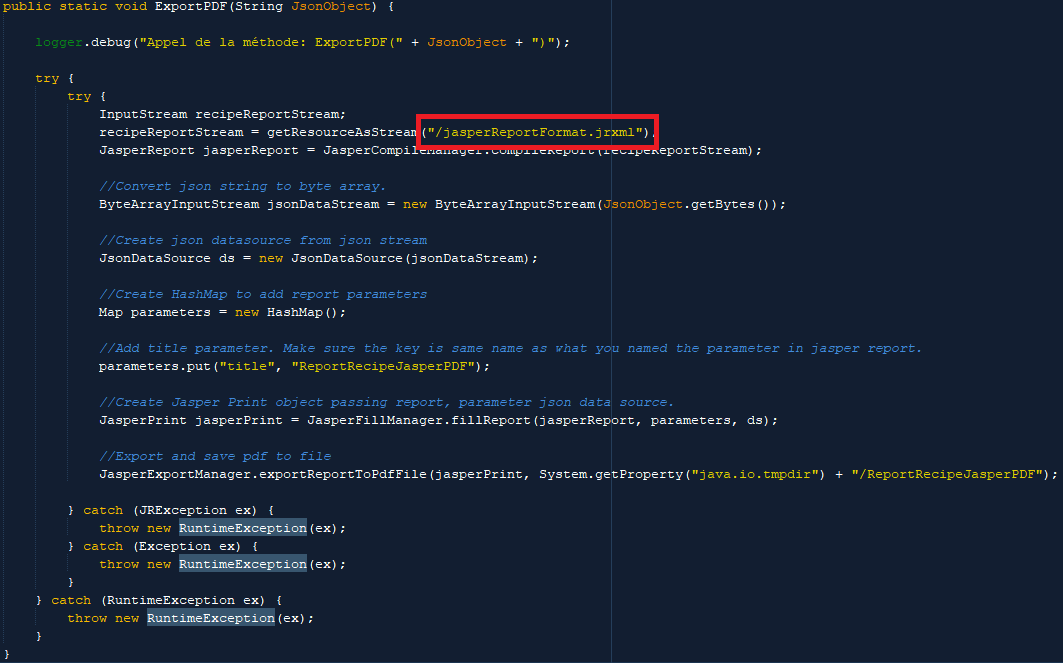


Par conséquent, nous obtiendrons en appuyant sur le bouton ci-dessous :

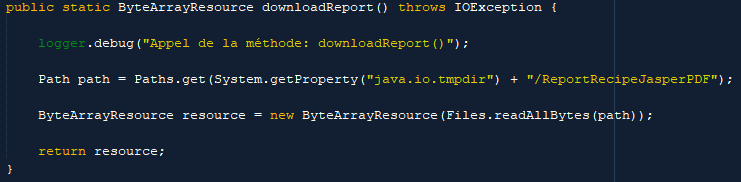
L’application client fait en effet appel à l’adresse : **« 192.168.56.101 :8181/getRecipes/ID\_Recipe\_current (ici c’est 3) »**

Grâce au mapping nous pourrons par la suite appeler la méthode **RecipeIdExportToPDF**(idRecipe : 3) située dans les webservices qui nous permettra de :

1. **Récupérer les informations** de la recette **3**
2. **Construire un rapport PDF** via jasper (JasperPrint) depuis les informations récupérées et le fichier « jasperReportFormat.jrxml » pour la construction du PDF (appel de la méthode ExportPDF(String JSONObject // informations recettes 3).

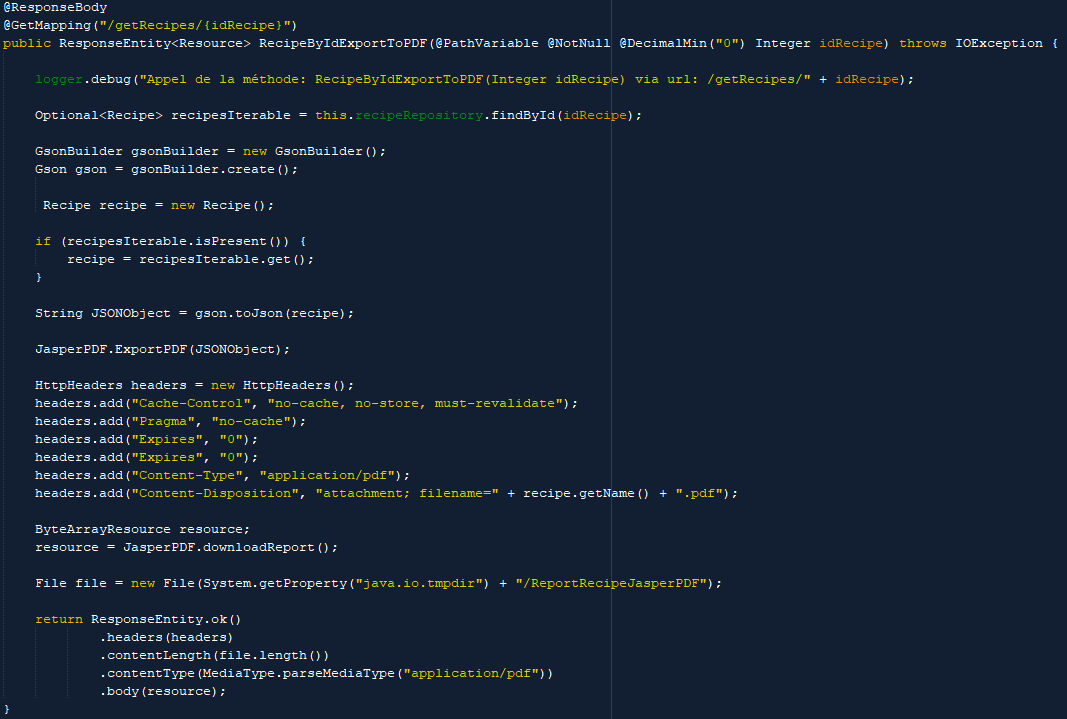
**ExportPDF (String JSONObject)** :

1. **Créer une ressource qui peut être transmise** dans une réponse à requête http GET (appel de la méthode downloadPDF())

**downloadPDF()**

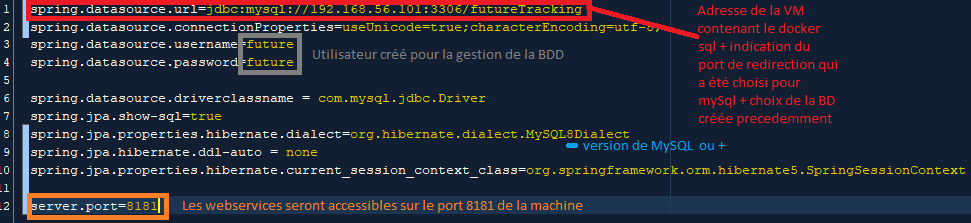
1. Ensuite, nous allons **construire un objet ResponseEntity** qui permettra la transmission du fichier PDF en fournissant directement le type des ressources qui sont transmises (on crée un header pour cela)

Vous trouverez ci-dessous la méthode **RecipeIdExportToPDF :**

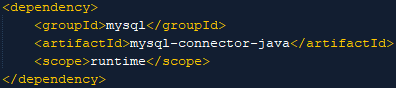


JASPER REPORT EST PRÊT !

1. **HIBERNATE**

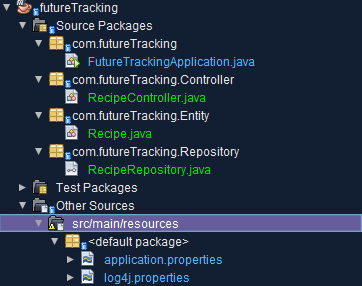
Comme précédemment expliqué, la configuration de Hibernate s’est réalisé dans le fichier ***application.properties et persistence.xml*** qui se trouve dans le répertoire ressource du projet.

La dépendance nécessaire pour la gestion de la classe driver :



Maintenant que nous avons tous les outils, il nous faut faire du ORM avec Hibernate pour récupérer par exemple toutes les recettes stockées en BDD.

Voici l’architecture de notre application :



Dans le package « Entity » se trouve la classe « Recipe » qui prend pour architecture celle de l’objet qui le représente en BDD.

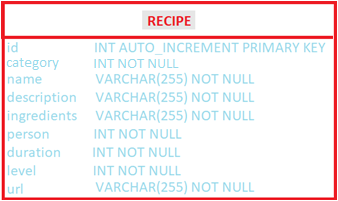
Dans le package « Repository », nous allons développer une interface « Repository » pour l’objet « Recipe » qui nous permettra grâce à l’héritage de la classe « CrudRepository<Recipe, Integer> » de récupérer les méthodes CRUD génériques (Create, Read, Update, Delete).

Dans le package « Controller », nous créerons une classe qui nous permettra de faire le mapping sur l’appel d’une méthode qui se servira de l’interface « RecipeRepository » afin d’utiliser les méthodes génériques.

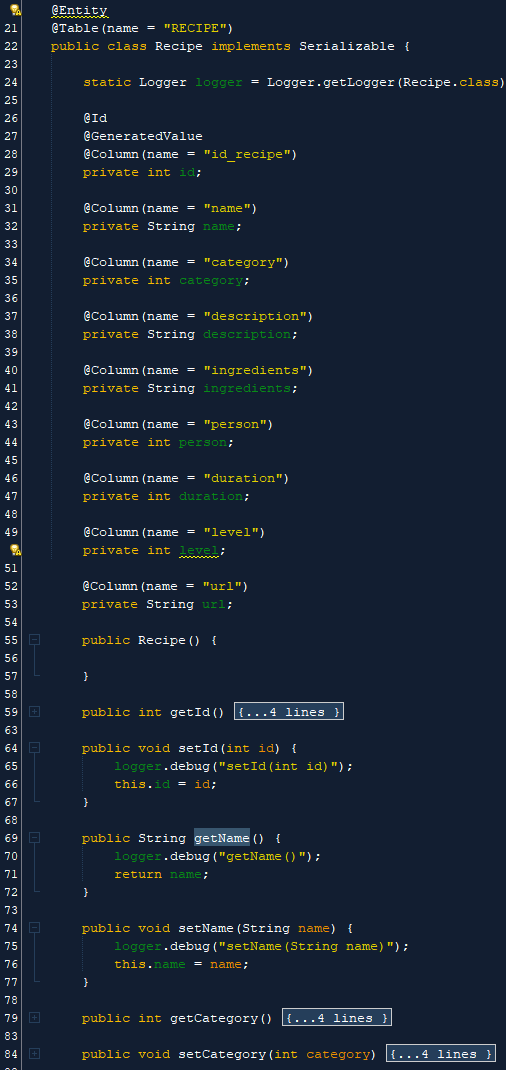
Pour la classe Entity, il ne faudra pas oublier les annotations qui permettent de spécifier ce qu’une classe représente.

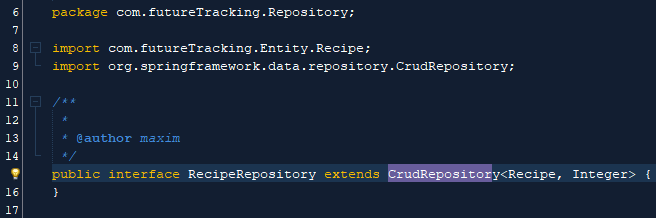
Est-ce un Repository (@Repository) ? Est-ce une classe lié à une table (@Entity / @Table(name = « «RECIPE ») / @Id et @GeneratedValue / @Column(name= « nomcolonne » / pas de cardinalités ici donc pas besoin de @OneToMany, etc…)

Nous avons en **BDD** :

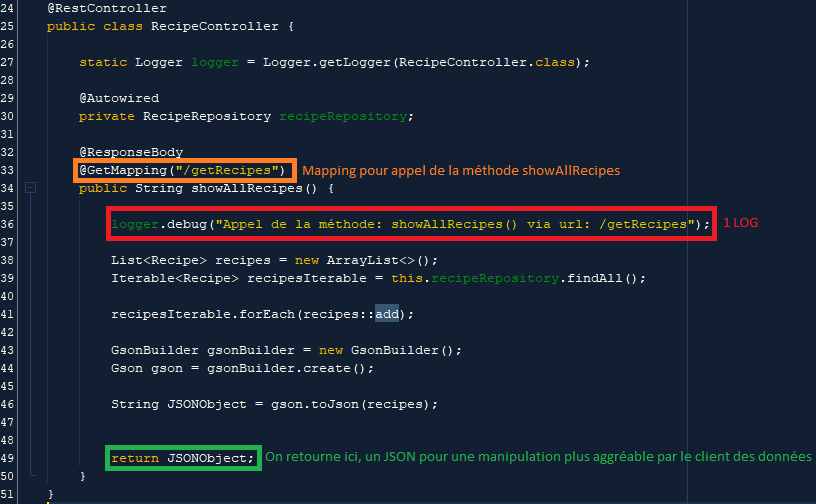


Nous avons dans **ENTITY** :

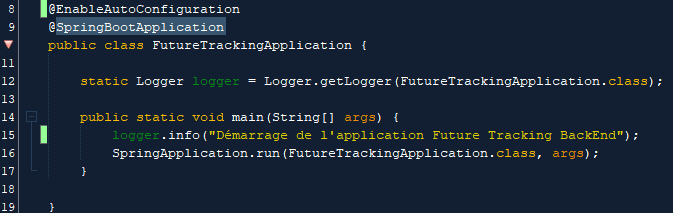


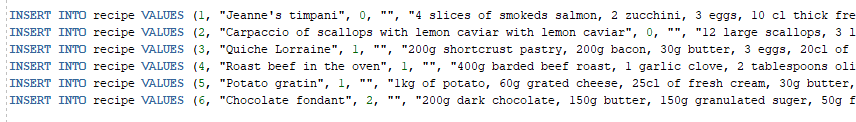
**Nous avons dans **REPOSITORY** :

Nous avons dans **CONTROLLER** :

**

Et enfin, nous avons dans le package par défaut, la main :

**

**Script des données contenu en BDD :

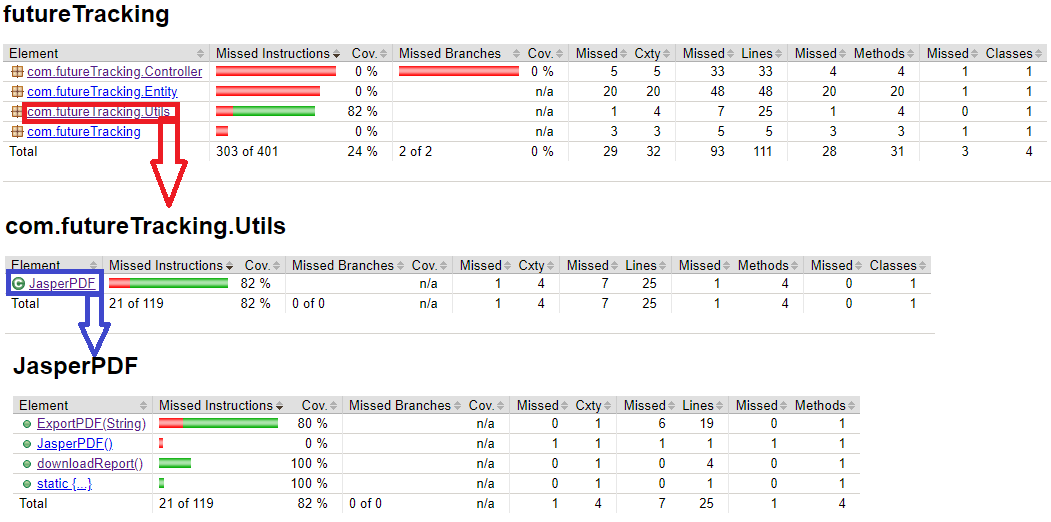
Résultat après appel du webservice :

**HTML TEST**

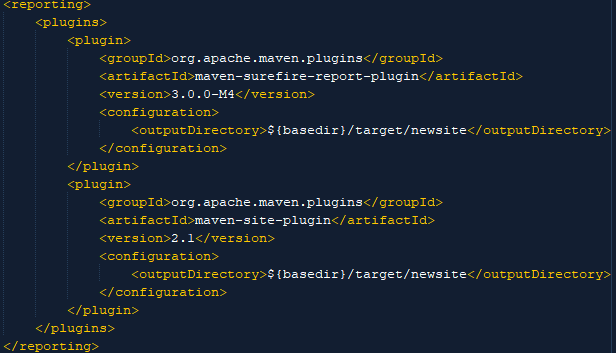
Le petit bonus avec jacoco, c’est que celui-ci réalise déjà des rapports de test en HTML, tout comme maven-surefire-report-plugin qui permet également d’avoir des rapports HTML dès que dans sa configuration nous spécifions un fichier de sortie. La différence étant que sur jacoco, nous avons surtout des détails sur la couverture de code effectué et à la fois qui est passé au vert dans les tests.

Ainsi, en se rendant à la racine du projet, après avoir exécuter une première fois les tests, vous trouverez dans le répertoire suivant : « ***futureTracking\target\site\jacoco*** » , un fichier **index.html** que je vous invite à ouvrir à l’aide de votre navigateur et qui vous permettra d’avoir un rapport sur tous les tests réalisés y compris le rapport de couverture de code.

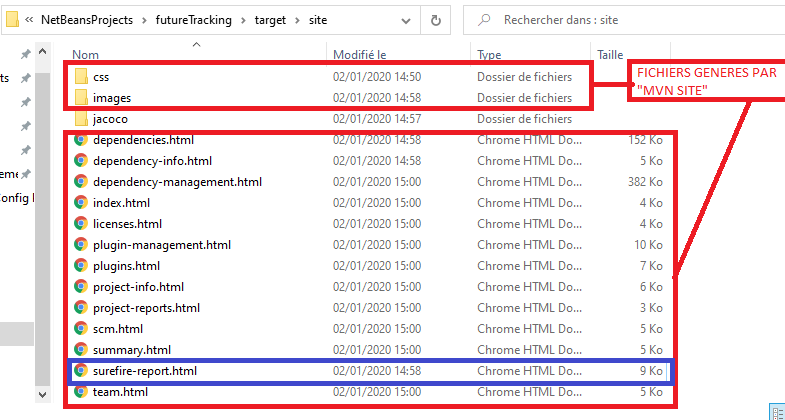
1 - Jacoco



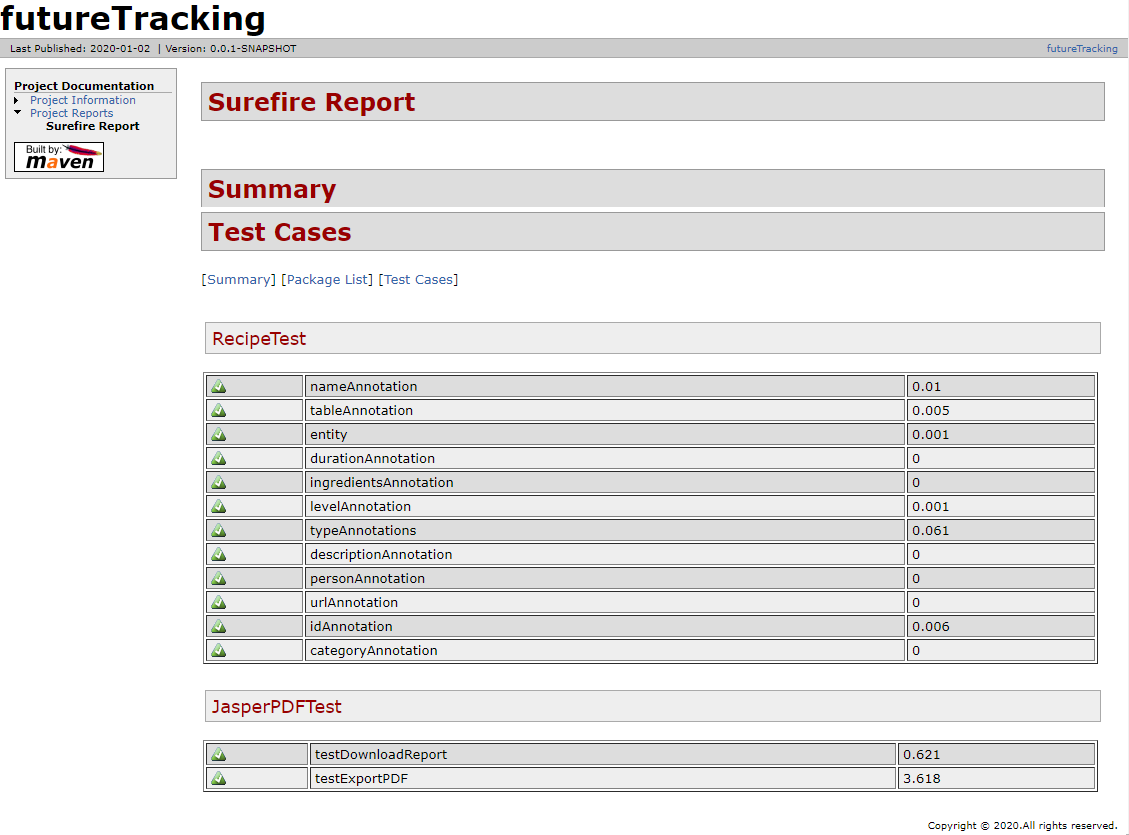
2 – Surefire-report

****Pom.xml :

En ajoutant ce plugin, lors de l’exécution de la commande mvn site, nous allons avoir accès au rapport HTML suivant :



Et en exécutant surefire-report.html :

****

Pour résumer, afin d’avoir l’effet BackEnd avec une architecture N-tiers, le .jar générer par le projet FutureTracking-BackEnd » a été exécuté avec la commande « java -jar futureTracking-0.0.1-SNAPSHOT.jar » sur la machine virtuelle.

**PROJET « FutureTracking-Client » (BRANCHE « MASTER » UNIQUEMENT)**

1. **APACHE TOMCAT – Migration du client Angular sur TOMCAT**

*Etape 1 : Télécharger et installer TOMCAT*

Suivre le lien : <https://o7planning.org/fr/11583/installation-et-configuration-de-tomcat-server>

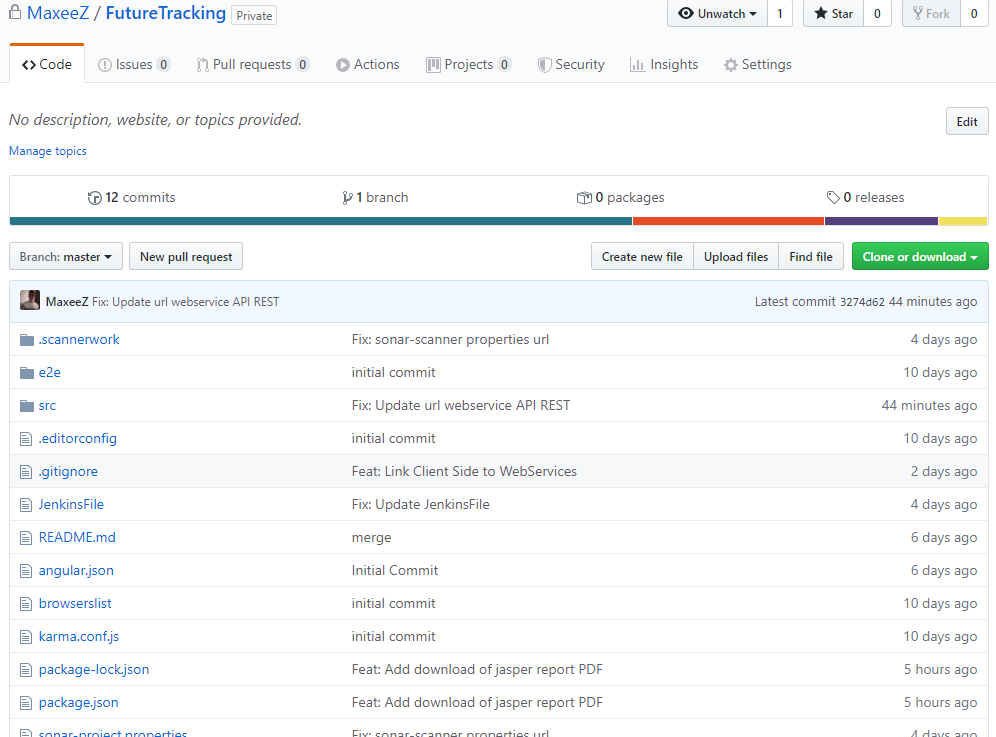
*Etape 2 : Mettre notre client Angular sur TOMCAT*

Il suffit tout simplement de réaliser un build de notre client avec la commande « ng build » et de copier toute la sortie du build dans le fichier www, le fichier de build est par défaut stocker dans le fichier « dist » qui se trouve à la racine du projet Angular.

Etape 3 : Accéder à votre projet

Pour cela, il suffit d’accéder via l’url : « ***localhost : portTomcat/repertoireDeNotreSiteAngular*** ».

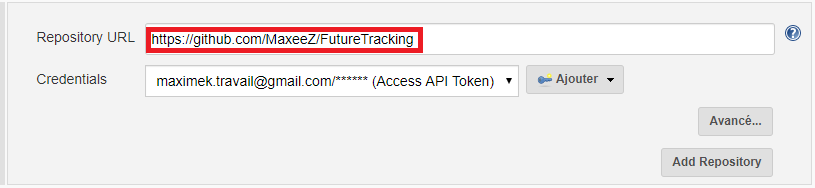
1. **Gestion de version (GITHUB)**
2. Créer un répertoire « FutureTracking-Client »
3. Récupérer le lien de partage (dans mon cas : <https://github.com/MaxeeZ/FutureTracking>)
4. Réaliser exactement les mêmes manipulations que pour le projet « FutureTracking-BackEnd »



1. **Intégration du projet dans Jenkins depuis GitHub**

L’intégration du projet client s’effectue exactement de la même manière que pour le projet « FutureTracking-BackEnd ».

Il faudra cependant changer le nom du pipeline par « FutureTracking-Client » et remplacer lien d’accès au fichier JenkinsFile par le lien suivant : « <https://github.com/MaxeeZ/FutureTracking> »



1. **JENKINSFILE**

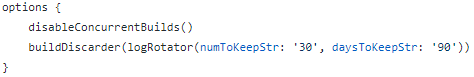
Le Nous utiliserons la machine « maître » par défaut et aucun slave pour l’exécution de nos builds :



Nous vérifierons tous les 5 minutes le répertoires GitHub afin de réaliser de nouvelles builds en cas d’évolution / modification sur le répertoire :



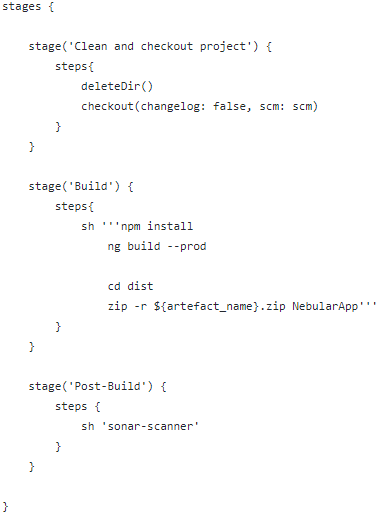
Nous paramétrons une durée de vie de 90 jours à la build avant suppression :



Cette fois-ci, j’ai émis la possibilité de configurer le nom de l’artefact qui sera générée à chaque lancement de build :



Durant le build :



Nous aurons une première étape de nettoyage du répertoire de build de Jenkins.

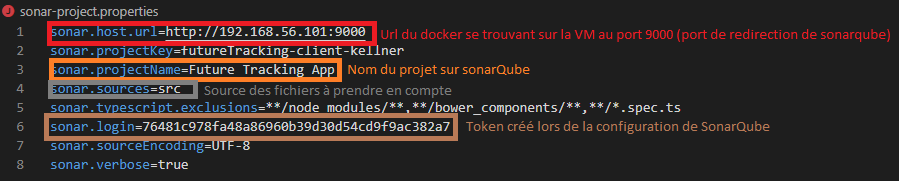
Ensuite nous effectuerons la build grâce à la commande **« npm build »** après avoir installé tous les packages nécessaires avec la commande **« npm install »**.

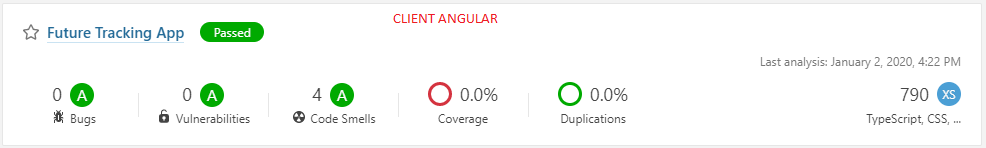
Ensuite, nous irons dans le fichier de sortie du build **« cd dist »** avant de construire l’artefact dont le nom aura été passé en paramètre !

Après la build, nous enverrons le code sur Sonarqube

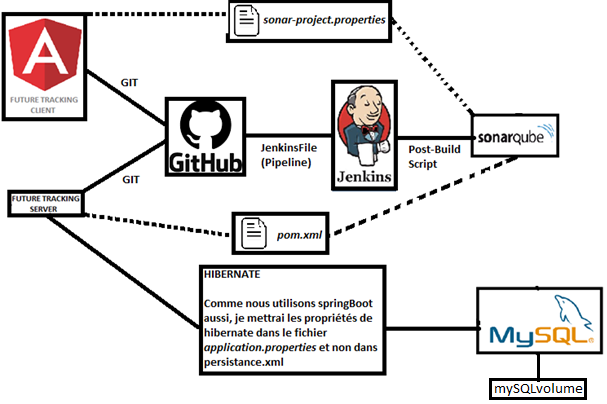
**E – SONARQUBE**

Afin de pouvoir utiliser la commande « sonar-scanner » il faudra paramétrer SonarQube dans le projet Angular, pour cela nous créerons un fichier ***« sonar-project.properties »*** que nous placerons à la racine du projet Angular***.***

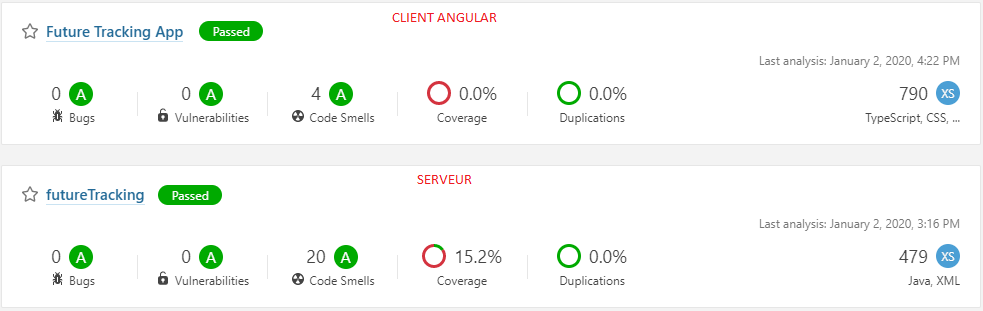


****

**RESUME DE LA CONFIGURATION**

**





**GESTION DE PROJET**