[Titre ]

OCDM Base Stack

Version 0.1

# Historique du document

## Révisions du document

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Version applicative concernée | Description | Auteur |
| 0.1 | 03/00/2023 |  | - Version initiale | TGE |
|  |  |  |  |  |

## Validation du document

|  |  |  |
| --- | --- | --- |
| Validation du commanditaire | <Organisation>  <Nom / Prénom>  <Fonction> | Date et visa : |
| Validation du prestataire | <Organisation>  <Nom / Prénom>  <Fonction> | Date et visa : |
| Approbateurs : | <Nom / Prénom>, <Organisation>  <Nom / Prénom>, <Organisation> |  |

Table des matières

[Historique du document 2](#_Toc139550978)

[Révisions du document 2](#_Toc139550979)

[Validation du document 2](#_Toc139550980)

[1. Contexte 5](#_Toc139550981)

[1.1. Objet du document 5](#_Toc139550982)

[1.2. Enjeux projet 5](#_Toc139550983)

[1.3. Utilisateurs 5](#_Toc139550984)

[2. Fork du socle technique 6](#_Toc139550985)

[1.4. Fork du projet sur Github 6](#_Toc139550986)

[1.5. Publication d’une copie sur un dépôt externe 8](#_Toc139550987)

[3. Démarrer le projet 10](#_Toc139550988)

[1.6. Prérequis 10](#_Toc139550989)

[1.7. Installation 10](#_Toc139550990)

[1.8. Exécuter le projet avec une BDD H2 embedded (rien à installer) 10](#_Toc139550991)

[1.8.1. Exécuter le projet avec une BDD PostgreSQL 10](#_Toc139550992)

[1.9. Commandes du projet front end 11](#_Toc139550993)

[1.9.1. Installation 11](#_Toc139550994)

[1.9.2. Compilation et hot reload 11](#_Toc139550995)

[1.9.3. Compilation et minification pour la production 11](#_Toc139550996)

[1.9.4. Correction du code pour le linter 11](#_Toc139550997)

[1.10. Installation alternative avec docker compose 12](#_Toc139550998)

[1.11. Utilisateur 12](#_Toc139550999)

[4. Build / packaging / run 13](#_Toc139551000)

[1.12. Packaging 13](#_Toc139551001)

[1.13. Configuration de l’application 13](#_Toc139551002)

[1.14. Démarrage de l’application 14](#_Toc139551003)

[1.15. CI/CD 14](#_Toc139551004)

[5. Implémentation des fonctionnalités 18](#_Toc139551005)

[1.16. Renommer le projet 18](#_Toc139551006)

[1.17. Implémenter une fonctionnalité backend 18](#_Toc139551007)

[1.17.1. Modèle en couche avec Spring MVC 18](#_Toc139551008)

[1.18. Ajouter un écran sur le frontend 20](#_Toc139551009)

[1.18.1. Modèle MVVM avec Vue 20](#_Toc139551010)

# Contexte

## Objet du document

Le présent document est le guide d’implémentation du projet Base Stack. Il vise à guider les développeurs dans la démarche de mise en œuvre d’une application métier basée sur le socle Base Stack.

## Enjeux projet

Ce projet a pour objectif de fournir une base de démarrage uniforme et prête à l’emploi pour l’implémentation des projets.

L’objectif du déploiement de cette base commune pour nos futurs projets est multiple

* Gains de temps au démarrage des projets
* Industrialisation des projets facilitée (build, packaging, CI…)
* Amélioration du support de la DT et des autres projets
* Facilité pour intervenir sur différents projets
* Amélioration de la capacité à mettre à jour le socle

Ce socle projet n’implémente pas de logique métier, il propose un socle technique fonctionnel répondant aux problématiques transverses commune des projets

## Utilisateurs

Les utilisateurs de ce logiciel sont les équipes projet qui implémentent cette base pour la décliner comme un projet à destination des utilisateurs finaux.

# Fork du socle technique

La première étape pour décliner le socle sous forme d’une application est de créer une copie des sources mises à disposition pour y apporter les modifications nécessaires à l’implémentation des fonctions attendues par le client.

Il existe 2 manières de réaliser cette première étape :

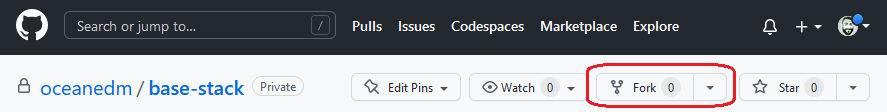
* Fork du projet sur Github
  + Méthode recommandée
* Clonage du projet puis publication sur un dépôt externe
  + Méthode alternative si le client a des contraintes spécifiques relatives au stockage du code

Dans les 2 cas le support de la DT est requis pour accéder au projet Base-Stack et/ou initier le dépôt du projet cible.

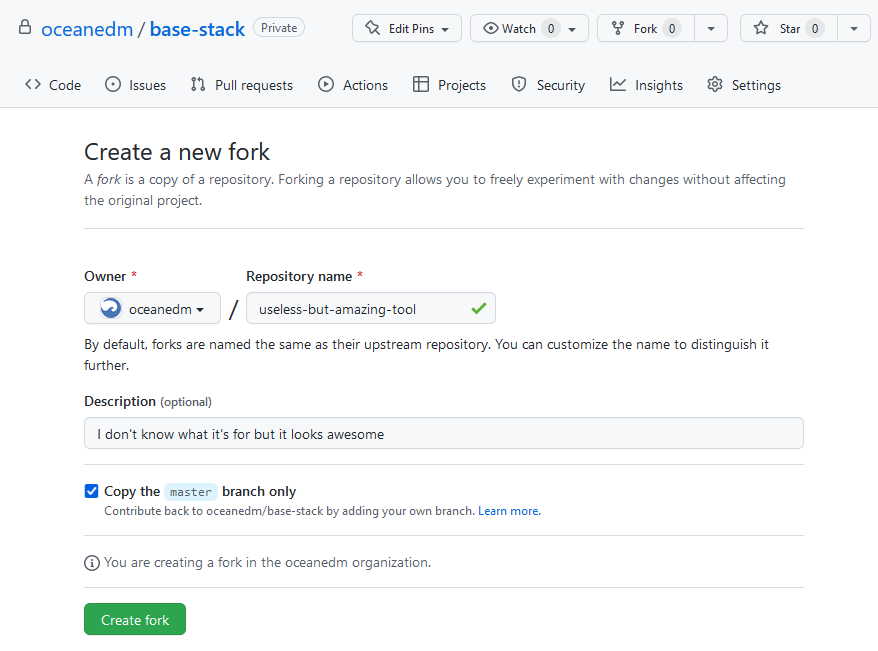
## Fork du projet sur Github

Cette méthode consiste à utiliser la fonction de Fork native de Github pour créer une copie du projet dans un nouveau dépôt.

Il suffit pour cela de cliquer sur le bouton « Fork » du projet « base-stack »



Puis de créer le fork dans l’espace « oceanedm » et de choisir un nom et une description appropriée



Une fois cette étape réalisée, le nouveau dépôt est prêt à être utilisé de manière classique (clone, branch, push, pull, PR…)

## Publication d’une copie sur un dépôt externe

Dans le cas où le client a exprimé des réserves à l’utilisation de la plateforme Github, notamment

* Si sa politique de sécurité exige que le code soit stocké sur le territoire Français
  + Utilisation de la plateforme Gitlab auto-hébergée par OCDM
* Si le client dispose de sa propre plateforme d’hébergement de code
  + Utilisation des services fournis par le client

Cette méthode consiste à cloner le projet sur son poste puis à publier la branche master sur le dépôt cible.

(La procédure est documentée en utilisant la ligne de commande pour être générique mais l’usage d’un [client Git graphique](https://git-scm.com/downloads/guis) est recommandé : IntelliJ IDEA, Eclipse EGit, Sublime Merge, SmartGit, GitKraken…)

1. Clonage du dépôt

|  |
| --- |
| git clone git@github.com:oceanedm/base-stack.git  cd base-stack |

1. Création du dépôt sur le serveur Git cible

Le dépôt doit être créé vide (pas de readme autogénéré, de .gitignore ou autre ficher autogénéré à la création)

S’adresser à la DT pour le serveur GitLab Oceane DM

Une image contenant texte

Description générée automatiquement

1. Configuration du dépôt distant cible

|  |
| --- |
| git remote add new-origin git@ocvgithub.oceanedm.local:integration/useless-but-amazing-tool.git |

1. Envoi du code sur le dépôt distant

|  |
| --- |
| git push -u new-origin master |

Le projet est maintenant disponible sur le nouveau dépôt, les membres de l’équipe peuvent le cloner, faire des pull et des push de manière habituelle.

|  |
| --- |
| git clone git@ocvgithub.oceanedm.local:integration/useless-but-amazing-tool.git |

/!\ Si le développeur qui a réalisé la procédure souhaite utiliser le dépôt créé dans le cadre de la procédure pour travailler, il est recommandé de supprimer le dépôt distant d’origine (git remote rm origin) et de configurer la branche de suivi sur le nouveau dépôt distant (git checkout --track new-origin/master) pour pouvoir simplement faire des git pull et git push sans préciser le dépôt « remote » à utiliser.

# Démarrer le projet

*Les informations de ce chapitre sont disponibles dans le readme du projet.*

## Prérequis

* JAVA 17
* Maven 3
* Node 18.14.0 (optionnel pour hot reload)
* NPM 9.3.1 (optionnel pour hot reload)
* PostgreSQL (optionnel pour lancer le projet avec une BDD PostgreSQL)
* Docker compose (optionnel pour lancer le projet dans des conteneurs dockers avec une BDD PostreSQL)

## Installation

Télécharger les dépendances avec Maven

|  |
| --- |
| mvn clean install |

## Exécuter le projet avec une BDD H2 embedded (rien à installer)

Créer une configuration par défaut dans votre IDE favoris pour lancer l'application spring-boot via le main de la classe BaseStackApplication avec les profiles suivants :

* dev
* h2

Ou lancer le projet avec Maven :

|  |
| --- |
| mvn spring-boot:run -Dspring-boot.run.profiles=dev,h2 |

Par défaut le serveur se lance sur le port 1987. Le front end est servi de manière statique à la racine de l'url du backend.

### Exécuter le projet avec une BDD PostgreSQL

#### Prérequis :

* un serveur PostgreSQL (testé avec PostgreSQL v15.1 et 15.2)
  + créer un utilisateur BDD accessible via mot de passe
  + créer un schéma vide
  + mettre à jour application-dev.yml avec les informations de connexion à la BDD

Suivre la même procédure que pour lancer sous H2 mais avec le profil dev uniquement (au lieu de dev et h2)

## Commandes du projet front end

Il est possible de lancer le frontend avec npm sur un port spécifique. Cela nécessite que le backend soit déjà lancé et permet de visualiser les modifications apportées au code Vue.js sans avoir besoin de rebuilder et de relancer le serveur (hot reload)

Les commandes suivantes sont à lancer depuis le répertoire *src/frontend*

### Installation

|  |
| --- |
| npm install |

### Compilation et hot reload

Par défaut le front sera disponible sur le port 1504.

|  |
| --- |
| npm run serve |

### Compilation et minification pour la production

|  |
| --- |
| npm run build |

### Correction du code pour le linter

|  |
| --- |
| npm run lint |

## Installation alternative avec docker compose

Lancer un environnement de dev

|  |
| --- |
| docker compose -f docker-compose-dev.yml up |

Stopper l'environnement

|  |
| --- |
| docker compose -f docker-compose-dev.yml down |

Cet environnement utilise une base de données PostgreSQL v15.1 avec persistance des données dans le répertoire 'pgdata' à la racine du projet.

## Utilisateur

Un utilisateur admin est créé automatique au lancement de l'application :

Email : *admin@admin.com*

Mot de passe : *admin*

# Build / packaging / run

## Packaging

Pour packager le livrable, il suffit de faire un build maven, au moins jusqu’à la phase *package* du cycle de vie Maven, le tout sans les profiles *dev* ni *h2.*

*Seuls Java et Maven sont requis pour le build et le packaging, les versions appropriées de Node et NPM sont automatiquement téléchargées et utilisées par Maven.*

Exemples valides

|  |
| --- |
| mvn clean install  mvn clean deploy  mvn package  … |

Une fois le build réussi, 2 fichiers war sont créés dans le répertoire *target* du projet :

* Un fichier .war auto exécutable intégrant une version d’apache Tomcat embedded dans lequel l’application est déjà déployée.

C’est le comportement par défaut de Spring Boot, ce mode de fonctionnement réduit les besoins de gestion des serveurs et permet de livrer des mises à jour du serveur d’application en même temps que les mises à jour de l’application elle-même

* Un fichier .war.original qui est un war « classique » pouvant etre déployé dans un serveur d’application Java installé sur le serveur

## Configuration de l’application

Pour que l’application fonctionne, il faut que les éléments de configurations présents dans le fichier application.yml soient valides (datasource, port, cors…)

Le fichier du projet src/main/resources/application.yml est embarqué dans le war, il est utilisé par défaut dans le cas où aucun fichier de configuration externalisé n’est présent mais ce fonctionnement n’est pas recommandé.

Il est préférable de déposer une copie de ce fichier qui pourra être édité pour correspondre à l’environnement ou est exécuté l’application, les éventuels fichiers application.yml seront chargés dans les emplacements suivants :

* A l’intérieur du war
* A la racine du CLASSPATH
* Dans un sous-dossier /config du CLASSPATH
* Dans le dossier courant
* Dans un sous-dossier /config du répertoire courant
* Dans les sous-dossiers du répertoire /config du dossier courant

Si plusieurs fichiers configuration.yml sont trouvés dans ces emplacements, ils seront tous chargés dans l’ordre de la liste, les valeurs communes des derniers écraseront les valeurs des fichiers présent plus haut dans la liste.

Il est également possible de surcharger le nom ou le chemin du fichier de configuration directement via les paramètres de démarrage de l’application.

Plus d’informations :

<https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config.files>

## Démarrage de l’application

Pour lancer l’application, il suffit de taper la commande suivante dans le répertoire ou est déposé le war :

|  |
| --- |
| java -jar app-name-X.Y.Z.war |

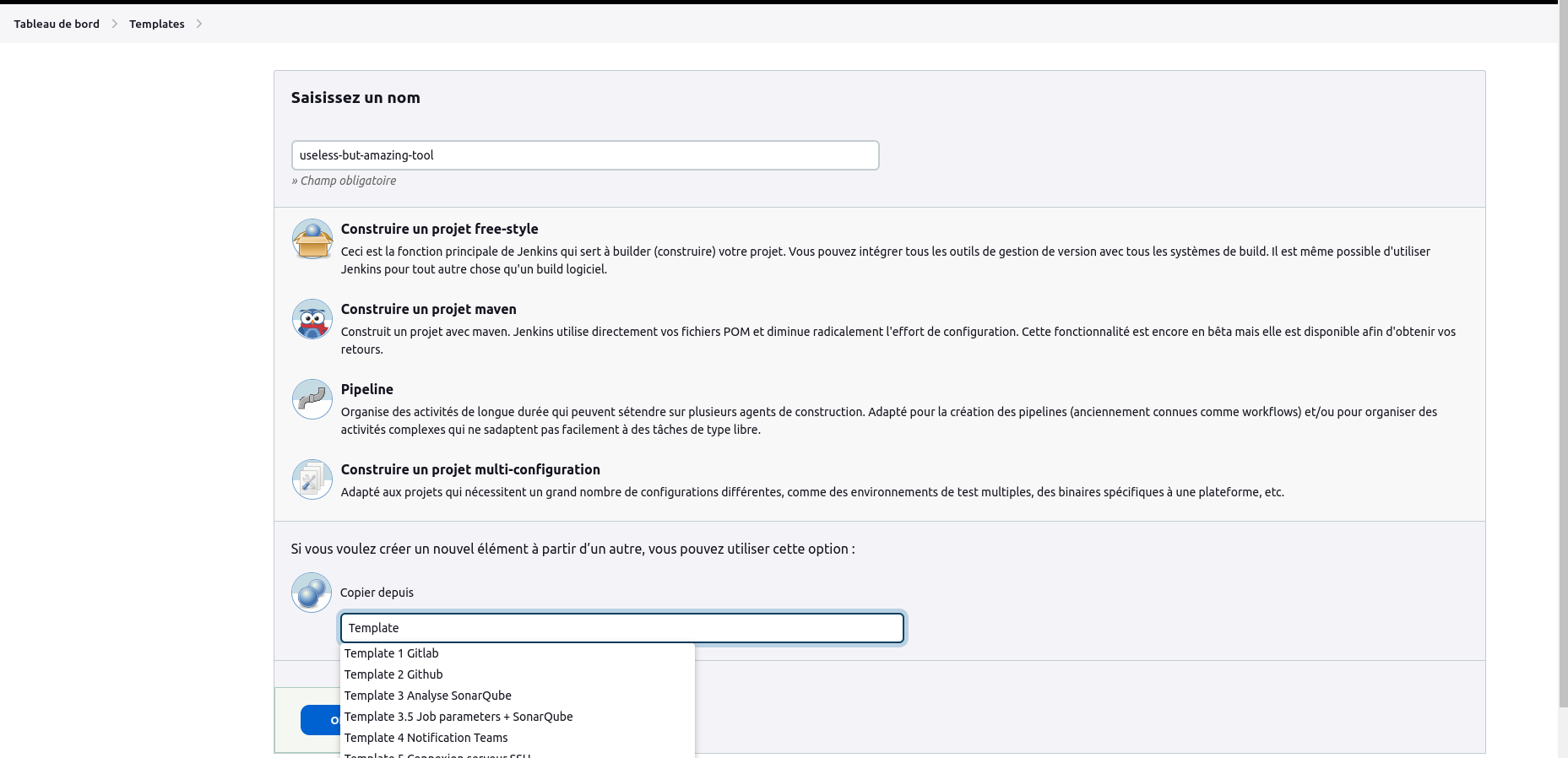
## CI/CD

En fonction de la typologie du projet et des besoins, vous serez amenés à intégrer le projet dans la CI/CD. Dans ce chapitre, nous allons intégrer le projet fraîchement créé dans l’orchestrateur Jenkins.

Tout d’abord, on se connecte à l’interface web de Jenkins :

* https://dm-forge.oceaneconsulting.com/jenkins

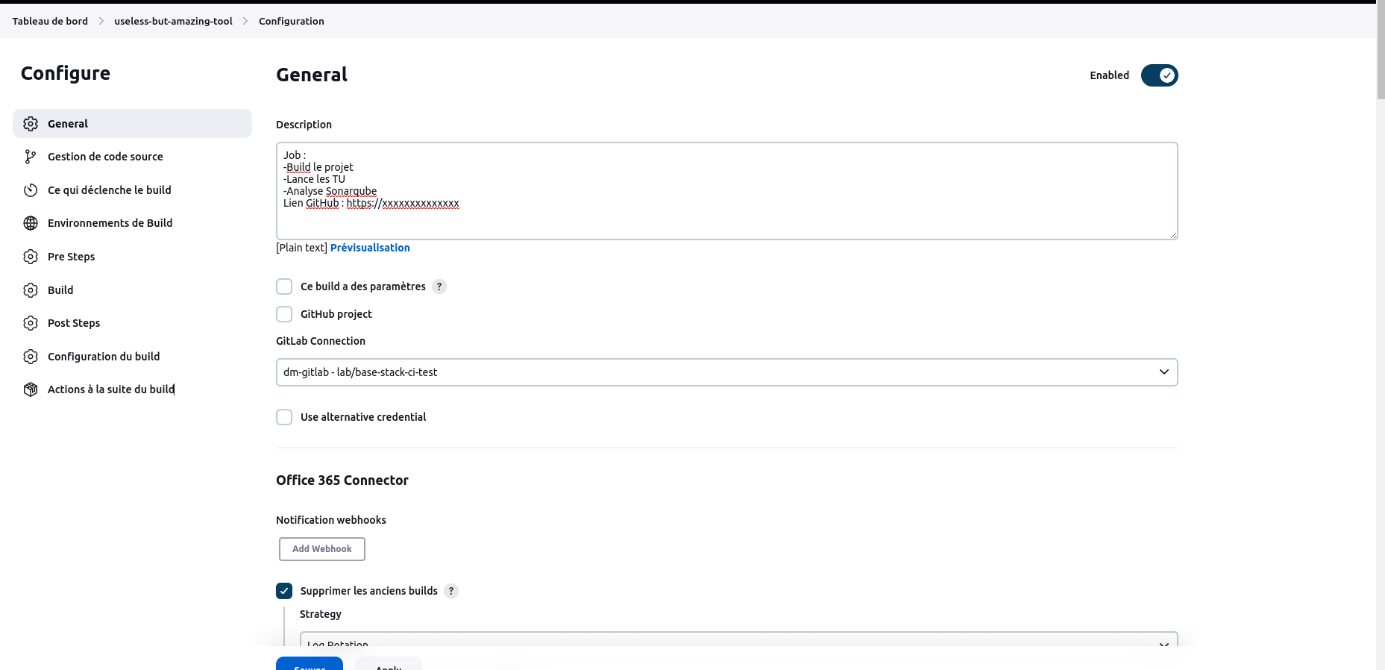
Une fois authentifié, depuis le tableau de bord, cliquez sur le bouton « Nouveau Item » pour créer un nouveau job :



Pour gagner du temps, initier la configuration du nouveau job à partir d’un existant. Les templates sont fait pour ça ☺

L’écran de configuration du nouveau job apparaît, Mettre à jour certaines parties, à minima :

* La description
* La version du JDK
* L’URL Git de clone
* Propriétés de l’analyseur Sonar
* Les notifications

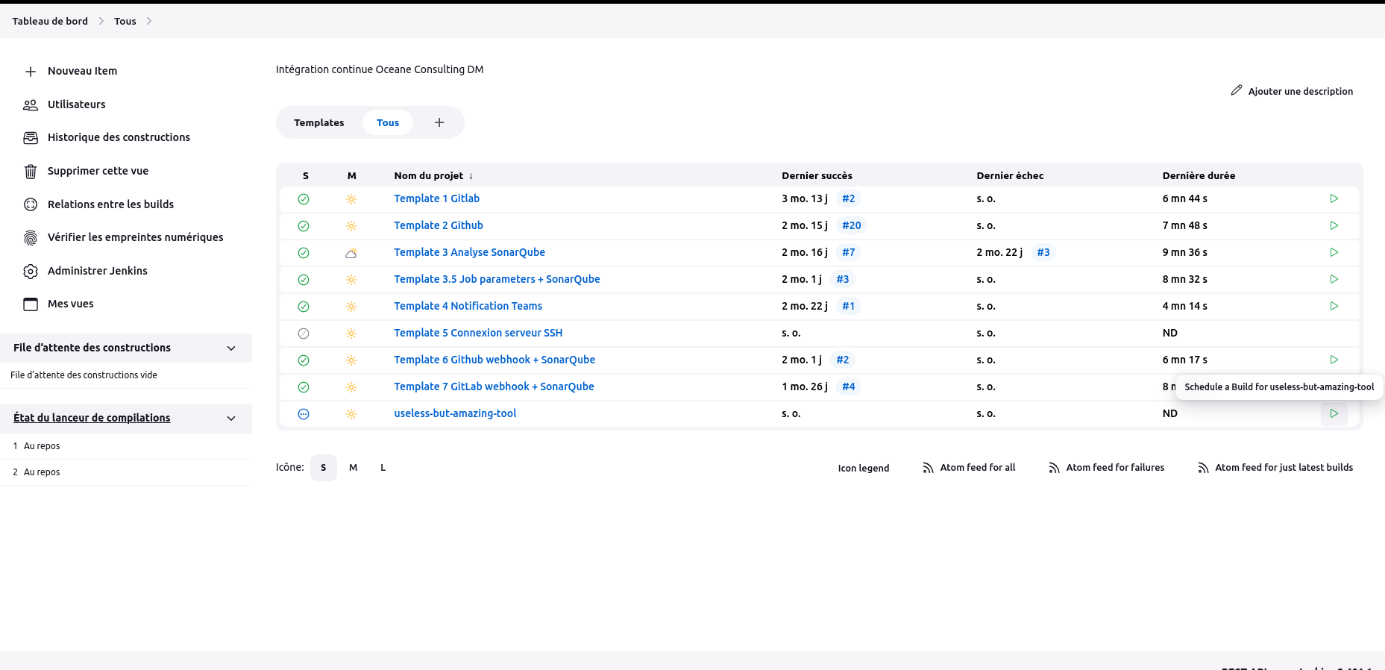


Pour les propriétés de l’analyseur Sonar :

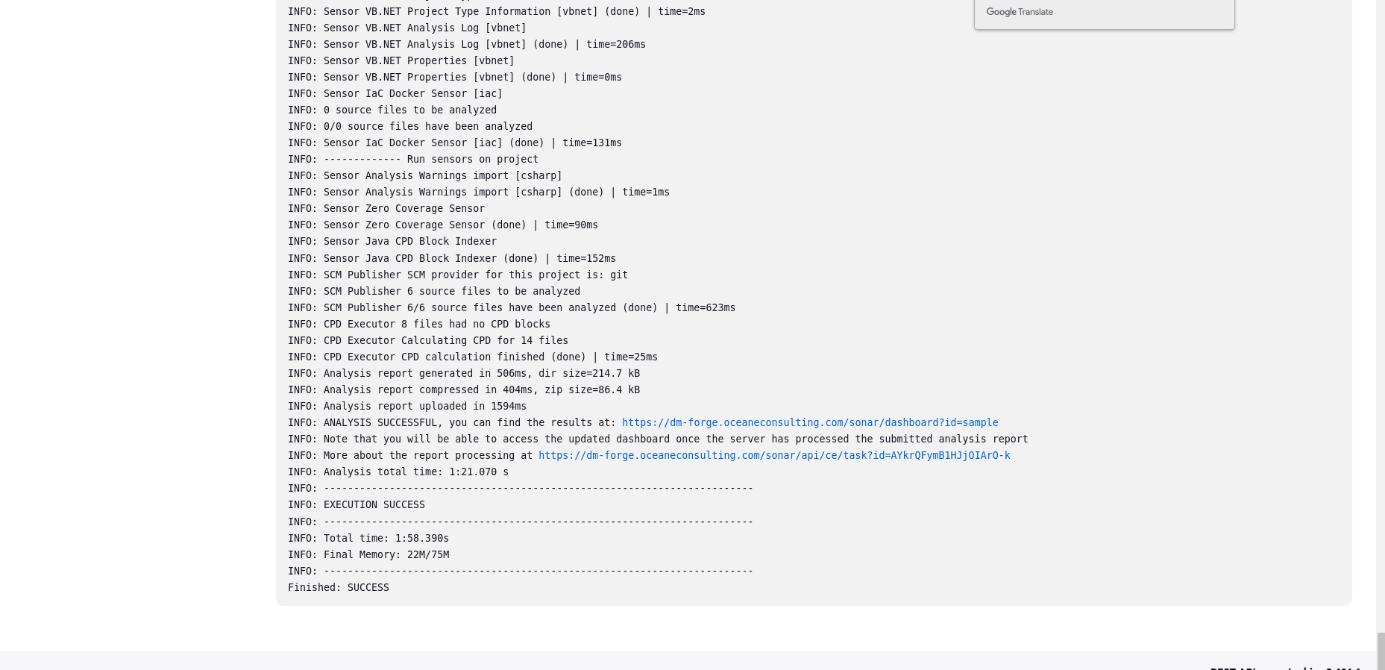
|  |
| --- |
| # unique project identifier  sonar.projectKey=useless-but-amazing-tool  # path to Java project compiled classes  sonar.java.binaries=target  # uncomment to force Jenkins to fail if sonarqube is unhappy  #sonar.qualitygate.wait=true |

Adapter la valeur de « sonar.projectKey » avec un nom unique afin de le distinguer des autres projets dans Sonarqube.

Valider que tout est OK. Lancer manuellement le job en cliquant sur l’icône « Run » à droite dans le tableau :



N’hésitez pas à consulter la sortie console :



# Implémentation des fonctionnalités

## Renommer le projet

La première chose à faire est de renommer le projet, pour cela il suffit de modifier le fichier pom.xml à a la racine du projet, notemment les valeurs

* /project/groupId
* /project/artifactId
* /project/name
* /project/description

## Implémenter une fonctionnalité backend

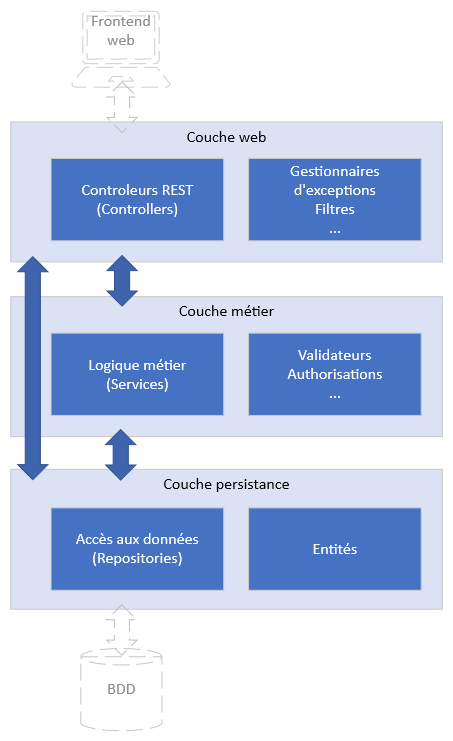
### Modèle en couche avec Spring MVC

Le modèle d’architecture utilisé côté backend est le modèle en couche de Spring MVC (Three tier)

Il consiste à faire « traverser » au flux d’exécution 3 couches composées principalement de

1. La couche web : des contrôleurs REST pouvant être appelés directement depuis l’extérieur, notemment par le frontend Vue.js s’exécutant dans le navigateur de l’utilisateur
2. La couche métier : des services servant à implémenter la logique et les règles métier
3. La couche persistance : des repository permettant d’accéder et de stocker des objets en BDD de manière transparente

Dans le cas d’opération de CRUD simple sans opération métier ou de validation, il est possible pour la couche web d’accéder directement à la couche persistance.



Les grandes étapes permettant d’ajouter une fonction dans le backend consistent donc à

1. Créer l’entité (objet Java servant à stocker des données) manipulée pour réaliser l’action, dans le package « model »
2. Créer un nouveau contrôleur et/ou opération pour exposer la fonction désirée (CRUD, action…)
   1. Cette opération prend en paramètre les données saisies/sélectionnées dans le Frontend et retourne les données désirées, généralement l’entité créée à l’étape 1 est utilisée en paramètre et/ou en retour de l’opération
3. Créer un service qui expose les méthodes publiques et privées permettant de réaliser les opérations métier demandées (contrôle des données, règles de gestions, appel de service externe, génération de document, envoi de mail…)
4. Créer un repository permettant de stocker et récupérer les données à manipuler, généralement l’entité créée à l’étape 1
5. Gérer les droits nécessaires pour appeler les opérations, soit via les annotations, soit dans le fichier SecurityConfig.java

Une fonction complexe pourra nécessiter de créer plusieurs opérations, contrôleurs, services ou « repositories ».

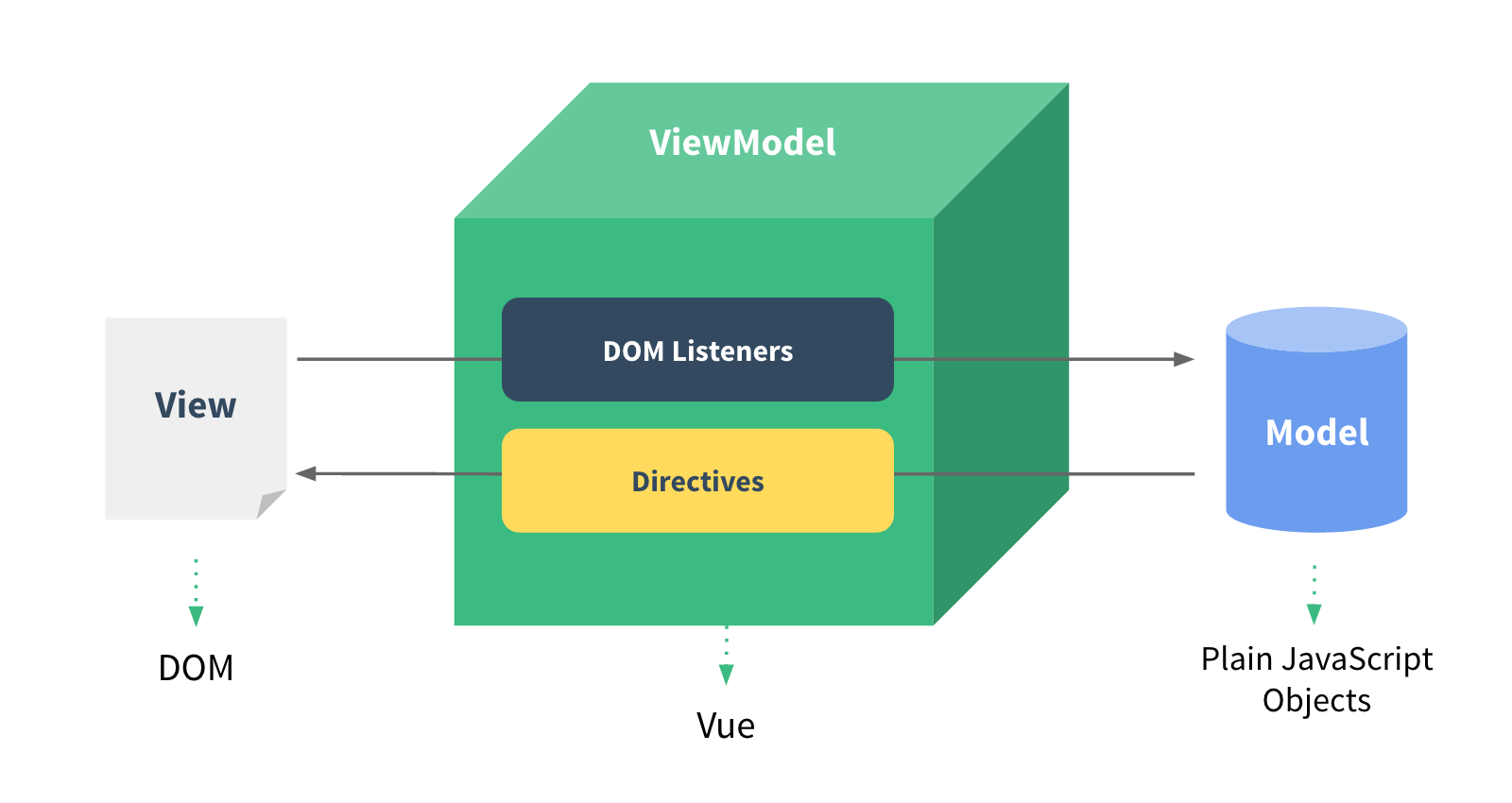
## Ajouter un écran sur le frontend

### Modèle MVVM avec Vue

Vue.js est un framework web component, il permet de découper les vues en un ensemble de composants réutilisables.

Le modèle d’architecture utilisé est le modèle MVVM, ce modèle consiste à proposer un composant « view model » (Vue.js) qui accède aux données du modèle et qui est lié à la vue (le DOM) par un système de data-binding :

* Les modifications de données coté vue (saisie utilisateur) ou coté « view model » (rafraichissement des données du model) sont notifiées et propagées en temps réel

[](https://012.vuejs.org/guide/#Concepts_Overview)

*https://012.vuejs.org/guide/#Concepts\_Overview*

Les grandes étapes permettant d’ajouter un écran dans le frontend Vue.js consistent à

1. Créer les composants nécessaires à l’écran s’ils ne sont pas déjà disponibles dans le package components
   1. Chaque composant embarque son template (la vue html)
   2. Son « view model » en JS
   3. Son style si besoin
2. Créer son écran, ce dernier est un composant comme les autres, il a simplement le rôle d’écran dans l’application, il contient généralement des composant du package components
3. Créer un service permettant d’appeler les API mises a disposition par le backend pour l’envoi et la réception des données
4. Définir la route\* permettant de définir le mapping entre l’écran créé et son url dans le navigateur.

*\* Le frontend est une SPA (Single Page Application), cela signifie que l’application est contenue dans unique fichier HTML. Afin de pouvoir naviguer de manière classique dans l’application (liens uniques, fonctions précèdent/suivant…) nous utilisons le module vue-router qui permet de mapper les écrans de l’application sur des urls.*