Documentation Technique de l'application Java de l'entreprise SubOne

Réalisé par Enzo Mancini

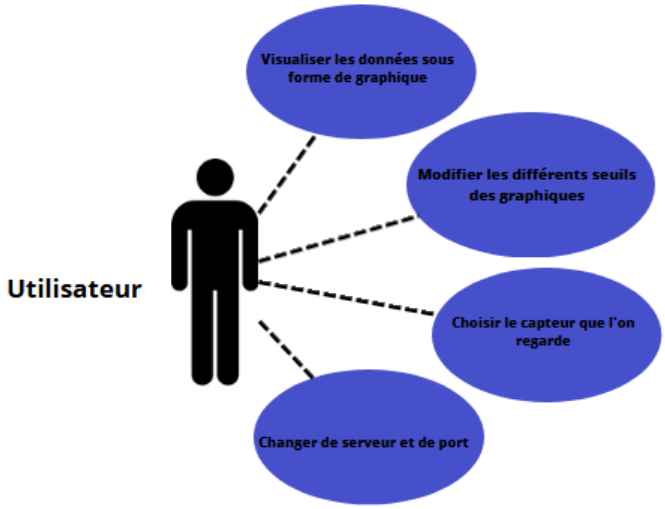
Table des matières :

I- Présentation rapide de l'application

L'entreprise SubOne souhaite pouvoir garder un œil sur leurs entrepôts de stockage et en particulier sur les conditions de stockage. Des capteurs sont présents dans l'entrepôt et chacun d'entre eux renvoie la température, le taux d'humidité et le taux de co2, il est nécessaire d'avoir une plateforme unique où voir les données de chaque capteur.

Notre application devra récupérer les différentes données envoyées par les capteurs et les afficher dans une IHM qu'il sera possible de paramétrer.

A- Explication du use case global

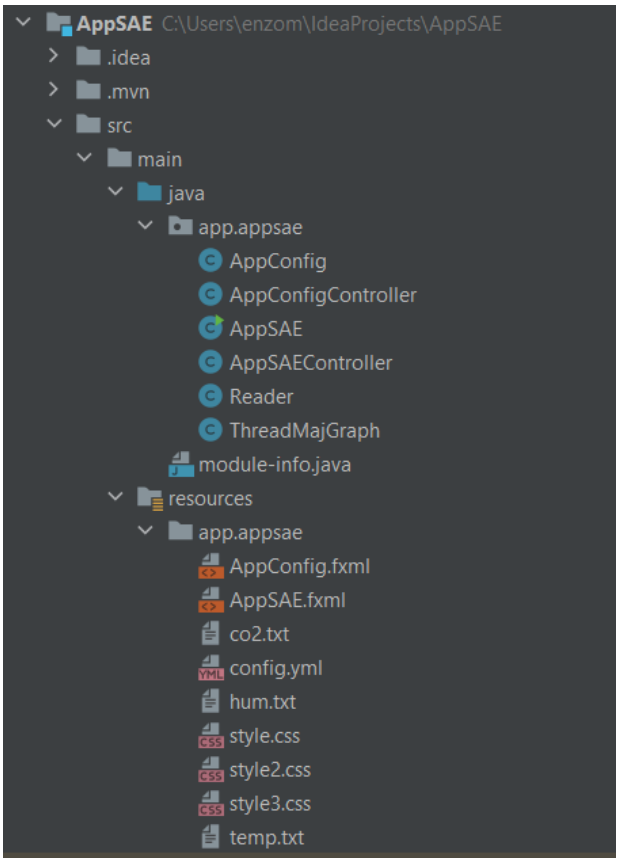


Ce use case représente un utilisateur (SubOne ou quelqu'un d'autre ayant l'application) et ce qu'il peut faire avec cette application.
Il pourra donc voir les données des capteurs sous formes de graphiques, modifier ces derniers (max, min, seuil de danger et le pas du graphique), choisir le capteur dont on veut voir les informations et enfin changer le nom du serveur et du port pour les différents entrepôts.

II- Architecture

A- Architecture générale

En terme de visuel, l'application a été faite pour être le plus clair possible et simple d'utilisation.
Dans le projet, les fichiers sont rangés en deux groupes, l'un contient les fichiers sources de l'application pour ouvrir les pages et que l'IHM fonctionne. L'autre contient les ressources que va utiliser l'application, on y retrouve les documents textes dans lesquels les données des capteurs sont stockées, le fichier de configuration yml pour les paramètres des graphiques ainsi que du serveur, les fichiers .fxml et enfin des fichiers de style facultatif.

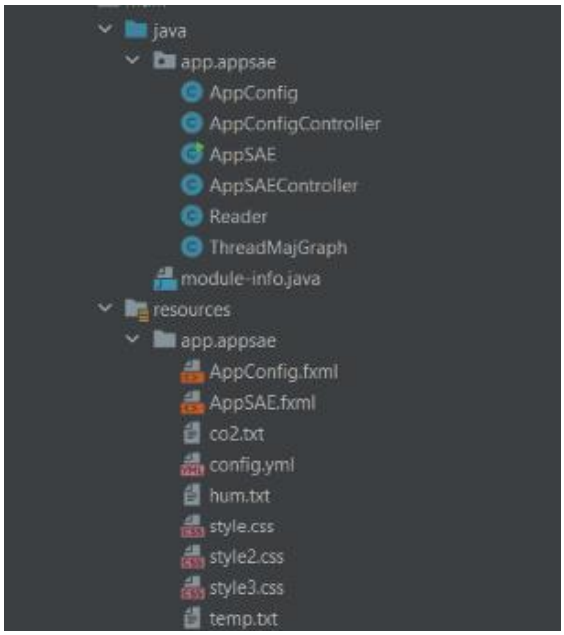


B- Ressources externes et rôles

Vu que le langage de l'application en Java, nous utiliserons l'extension .jar, celui-ci étant le plus pratique et avantageux en terme de stockage des classes.

Il faudra cependant un JRE 1.8 au minimum car l'application utilise JavaFX. L'application à aussi utilisé FXML et SceneBuilder pour son aspect pratique.

C- Structuration en packages de l'application



Notre application n'utilise que deux packages qui sont Java pour le code JAVA et Resources qui contient tous les fichiers utilisés hors java.

D- Éléments à connaître

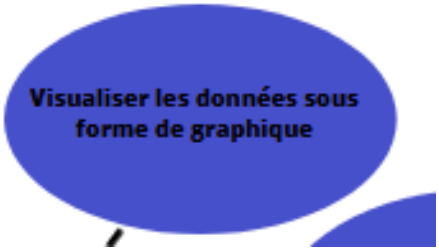
Éléments nécessaire au développement :

Si quelqu'un a besoin des fichiers sources de l'application pour vérifier le fonctionnement de celle-ci ou simplement pour de la maintenance, il devra utiliser une machine avec un JRE 1.8 au minimum.
Il lui faudra aussi un IDE pour visualiser le code, comme Eclipse ou IntelliJ IDEA qui est celui qui a été utilisé pour coder l'application.
Le workspace utilisé devra être paramétré pour JavaFx et le logiciel SceneBuilder est vivement conseillé pour manipuler les fichiers FXML.

III- Explication des fonctionnalités

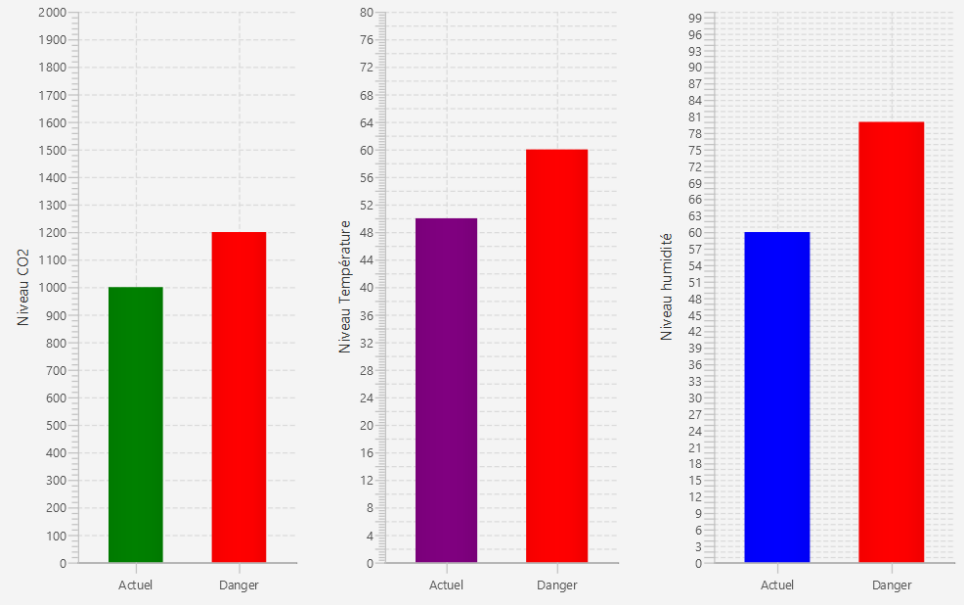
A- Classes impliquées

Première fonctionnalité :



Dès le lancement de l'application, nous pouvons voir 3 graphiques qui nous donnent le niveau actuel et le seuil de danger à ne pas dépasser des trois données: CO2, Température et Humidité.
Un Thread actualise ces graphiques toute les X min afin que les données affichées soient les dernières que les capteurs ont envoyées.

Rendu :



Classes impliquées :

Coté Java

→ *AppSAE.java*
→ *AppSAEController.java*
→ *Reader.java*

→ *ThreadMajGraph.java*

Coté Ressources :

→ *config.yml*
→ *AppSAE.fxml*
→*co2.txt*
→*hum.txt*
→*temp.txt*
→*style.css*
→*style2.css*
→*style3.css*

Éléments à connaître :

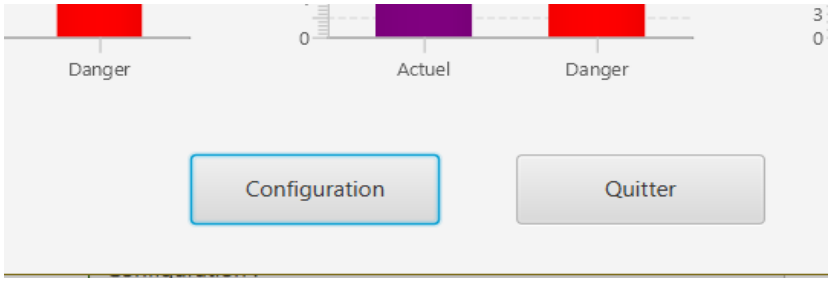
La classe *AppSAE.java* est notre classe application, c'est elle qui se lance en premier, son controller (*AppSAEController.java*) initialise d'abord les graphiques avec les paramètres stockés dans *config.yml*.

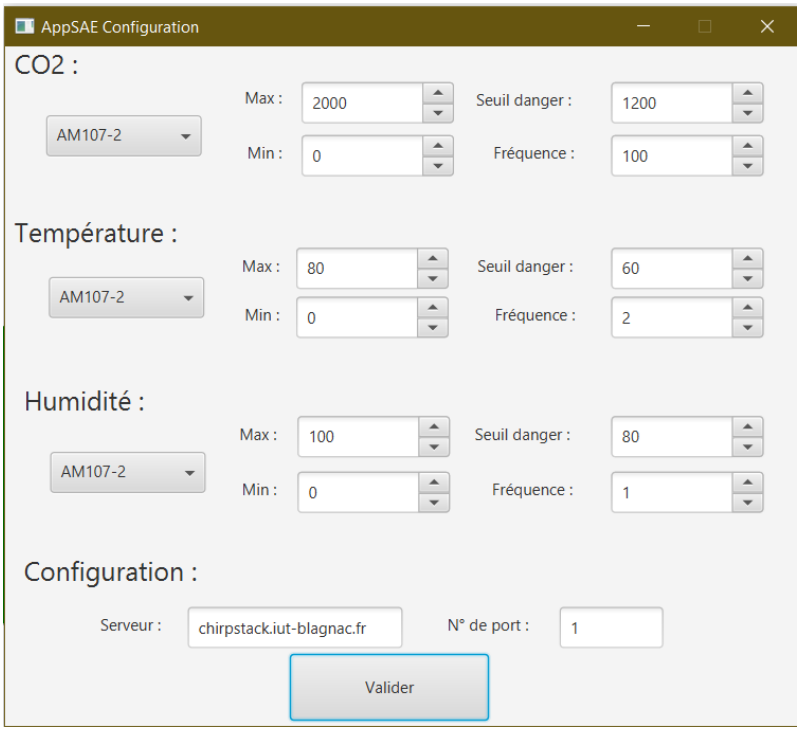
Toute les X min le Thread dans *ThreadMajGraph.java* lance trois méthodes (*majChartCo2, majChartTemp, majCharthum*) contenus dans le controller qui vont créer les barres et les mettre à jour en récupérant les données dans les fichiers txt (co2, température et humidité).

Deuxième fonctionnalité :

En appuyant sur le bouton 'Configuration' de la première page, celui-ci ouvre une nouvelle fenêtre dans laquelle nous pouvons modifier les seuils maximum, minimum, de danger et le pas des graphiques, ainsi que le capteur sélectionné et modifier le serveur et le port des capteurs.

Rendu :





Classes impliquées :

Coté Java

→ *AppConfig.java*
→ *AppConfigController.java*
→ *Reader.java*

Coté Ressources :

→ *config.yml*
→ *AppConfig.fxml*

Éléments à connaître :

La valeur de base de chaque case est la dernière valeur enregistrée dans le fichier de configuration. Pour que les nouvelles valeurs enregistrées soit affichées, il faut redémarrer l'application.

Quand le formulaire est validé celui-ci écrase le contenu de config.yml et écrit les nouvelles données saisies à la place. Pour cela on utilise PrinterWriter et Yaml avec une Map que l'on a créé et dans lequel on y a mis nos nouvelles données.

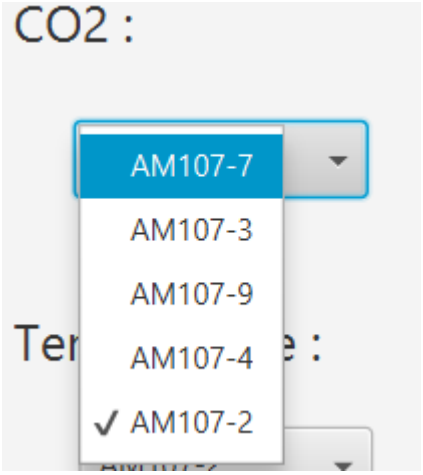
```
try {
    //une fois que toutes les données sont dans map on créait un PrintWriter vers le fichier config.yml pour ecra
    PrintWriter writer = new PrintWriter(new File( pathname: "src/main/resources/app/appsae/config.yml"));
    //on créait un nouveau yaml
    Yaml yaml = new Yaml();
    //on dump dans le yaml les données de map dans le fichier config.yml
    yaml.dump(map, writer);
}catch (Exception e){
```

```java
//on charge le fichier de config dans la Map data
data = reader.chargerYaml();
//on créait un nouveau Map pour stocker les nouvelles données
Map<String, String> map = new HashMap<>();
//on insère dans map les valeurs de la page de configuration
map.put("server",this.textserveur.getText());
map.put("appId",this.textport.getText());
//ici on récupère l'ancienne valeur (présente dans data) car on ne le changera jamais
map.put("deviceId",data.get("deviceId"));
map.put("tauxMaxCO2",this.maxCO2.getValue().toString());
map.put("tauxMinCO2",this.minCO2.getValue().toString());
map.put("seuilCO2",this.seuilCO2.getValue().toString());
map.put("frequenceCO2",this.freqCO2.getValue().toString());
map.put("tauxMaxTemp",this.maxTemp.getValue().toString());
map.put("tauxMinTemp",this.minTemp.getValue().toString());
map.put("seuilTemp",this.seuilTemp.getValue().toString());
map.put("frequenceTemp",this.freqTemp.getValue().toString());
map.put("tauxMaxHum",this.maxHum.getValue().toString());
map.put("tauxMinHum",this.minHum.getValue().toString());
map.put("seuilHum",this.seuilHum.getValue().toString());
map.put("frequenceHum",this.freqHum.getValue().toString());
map.put("capteurCO2",this.capteurCO2.getValue());
map.put("capteurTemp",this.capteurTemp.getValue());
map.put("capteurHum",this.capteurHum.getValue());
try {
```

Data ici récupère les données de config.yml sous forme d'une autre Map via la méthode chargerYaml que j'ai créé dans Reader.java.

Troisième fonctionnalité :

Comme indiqué ici, l'utilisateur peut choisir le capteur sur lequel il souhaite récupérer des données, il est possible de choisir trois capteurs différents pour les trois graphiques.



Classes impliquées :

Coté Java
- → *AppConfig.java*
- → *AppConfigController.java*
- → *Reader.java*

Coté Ressources :
- → *config.yml*
- → AppConfig.fxml

Éléments à connaître :

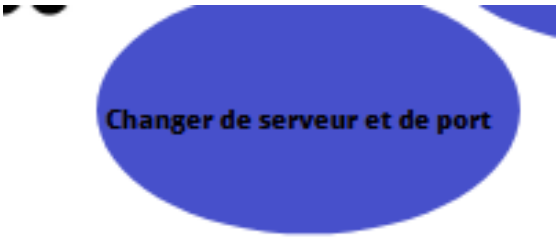
J'ai seulement mis 5 capteurs même s'il y en a beaucoup plus normalement, j'ai fait cela car je ne savais pas du tout combien de capteurs il y avait et je ne voulais pas créer de problème dans l'application. Pour en rajouter c'est très simple, il suffit juste de les écrire à la suite dans cette ligne :

```
//on remet les différents capteur (en brut..)
this.capteurCO2.getItems().addAll( ...es: "AM107-7","AM107-3","AM107-9","AM107-4","AM107-2");
```

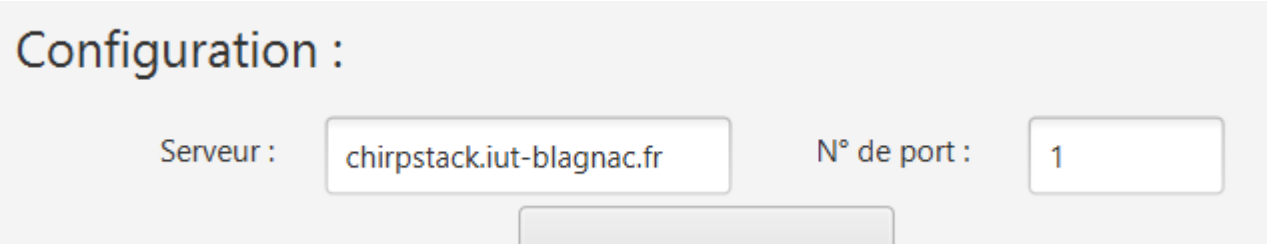
Dans la méthode setConfig() de la classe AppConfigController.java.

A noter qu'il y a trois menus déroulants différents, donc il faut modifier les trois lignes de la même manière.

Dernière fonctionnalité :



Afin de se connecter au bon réseau de capteur, il faut changer le nom de serveur et le port connecté auquel le Python se connectera. C'est le réseau de l'IUT par défaut.



Classes impliquées :

Coté Java
- → *AppConfig.java*
- → *AppConfigController.java*
- → *Reader.java*

Coté Ressources :
- → *config.yml*
- → AppConfig.fxml

Éléments à connaître :

Comme expliqué plus haut, à la validation du formulaire l'ancien fichier de configuration est remplacé par un nouveau avec les nouvelles données, dont le serveur et le port. Le code python récupérera ces deux informations afin de s'y connecter. Il faudra sûrement redémarrer le code Python après que les informations aient été validées.

IV- Procédures d'installation

A- Pour le développement

Sur notre dépôt se trouvent nos fichiers sources et ressources de l'application, pour les récupérer et les ouvrir dans un IDE il faut tout d'abord les télécharger dans un dossier. Ensuite, l'ouvrir dans votre IDE que vous voulez ; attention il y aura sûrement une erreur de *source root* mais pas d'inquiétude, c'est facilement réglable.

Sur IntelliJ par exemple il suffit de faire :

File→ Project Structure→Modules→ src→ Application→ Sources→ Apply

Il faut aussi être sûr d'avoir un JRE 1.8 au minimum et normalement tout devrait être bon.

Si ce n'est pas le cas, essayez de voir si toute les librairies sont installées, si ce n'est pas le cas faites :

File→ Project Structure→ Librairies et le petit + pour en ajouter.

Pour le code Python, voir sa documentation.

B- Pour un utilisateur

Afin d'utiliser l'application, il faudra premièrement avoir un JRE 1.8 d'installé sur votre machine, télécharger le .jar sur le dépôt GitHub et le projet Python. Il vous faudra lancer et bien régler un projet Python sur PyCharm par exemple (voir documentation du code Python) et enfin lancer les deux codes.