

Documentation Technique

Sommaire

1. Pr�sentation de l'application	4
1.1 Objectif et fonctionnement	4
1.2 Fonctionnalit�s principales	4
1.2.1 Diagramme des Cas d'Utilisation	4
1.2.2 Description des fonctionnalit�s	4
2. Architecture de l'application	5
2.1 Architecture g�n�rale	5
2.1.1 Arborescence	5
<i>Arborescence du projet.</i>	5
<i>Description des r�pertoires et fichiers.</i>	5
2.1.2 Sous-syst�mes de l'application	6
<i>Application Java / JavaFX</i>	6
<i>Programme Python</i>	6
2.1.3 Fichiers utilis�s	6
<i>Fichier de configuration.</i>	6
<i>Fichiers de donn�es</i>	8
2.1.4 Interactions entre sous-syst�mes et fichiers	9
2.2 Ressources externes	10
2.2.1 API utilis�es	10
<i>JavaFX</i>	10
<i>OpenCSV</i>	10
2.2.2 Plugins utilis�s	10
<i>JavaFX Maven Plugin</i>	10
<i>Apache Maven Shade Plugin.</i>	10
<i>Apache Maven Javadoc Plugin.</i>	11
2.3 Structuration de l'application Java	11
2.3.1 Patterns mis en oeuvre	11
<i>Architecture MVC</i>	11
<i>Composants en Singleton.</i>	12
2.3.2 Packages	12
<i>Arborescence des packages.</i>	12
<i>Description des packages et de leur contenu</i>	12
2.4 Sp�cifications techniques	14
2.4.1 Conventions de nommage	14
<i>Nommage des classes</i>	14
<i>Nommage des variables.</i>	15

<i>Nommage des packages</i>	16
<i>Nommage des vues FXML</i>	16
2.4.2 Conventions de commentaire	16
Commentaire d'entête de classe	16
Commentaire d'entête de méthode	17
2.4.3 Structures récurrentes de classes	18
<i>Contrôleurs de dialogue</i>	18
<i>Contrôleurs de vue</i>	20
2.4.4 Threads utilisés	22
<i>Thread de récupération des données</i>	22
<i>Threads de mise à jour de l'affichage</i>	24
3. Conception et mise en oeuvre des fonctionnalités	25
3.1 Diagramme de Séquence Système	25
3.2 Implémentation des fonctionnalités	26
3.2.1 Menu principal	26
<i>Ouverture du menu principal</i>	26
3.2.2 Formulaire de paramétrage de la configuration	26
<i>Ouverture du formulaire</i>	26
<i>Enregistrement d'une configuration</i>	27
3.2.3 Tableau de bord	27
<i>Ouverture du tableau de bord</i>	27
<i>Mise à jour automatique des données affichées</i>	28
<i>Affichage d'un graphique de comparaison</i>	28
<i>Affichage d'un graphique d'évolution</i>	29
4. Procédures d'installation	30
4.1 Installation pour développement	30
4.1.1 Prérequis	30
4.1.2 Étapes d'installation	30
4.2 Installation pour utilisation	31
4.2.1 Prérequis	31
4.2.2 Étapes d'installation	31
4.2.3 Étapes de lancement	32

Étudiants
Nolhan Biblocque
Léo Guinvarc'h
Victor Jockin
Mathys Laguilliez
Mucahit Lekesiz

Enseignant
André Péninou

Formation
BUT Informatique
2^{ème} Année
Promotion 2024-2025

Établissement
IUT de Blagnac
Université Toulouse II Jean Jaurès (31)

1. Présentation de l'application

1.1 Objectif et fonctionnement

L'application développée a pour objectif de fournir un outil centralisé de visualisation et d'analyse des données issues de capteurs connectés. Elle repose sur une architecture modulaire combinant Java/JavaFX et Python.

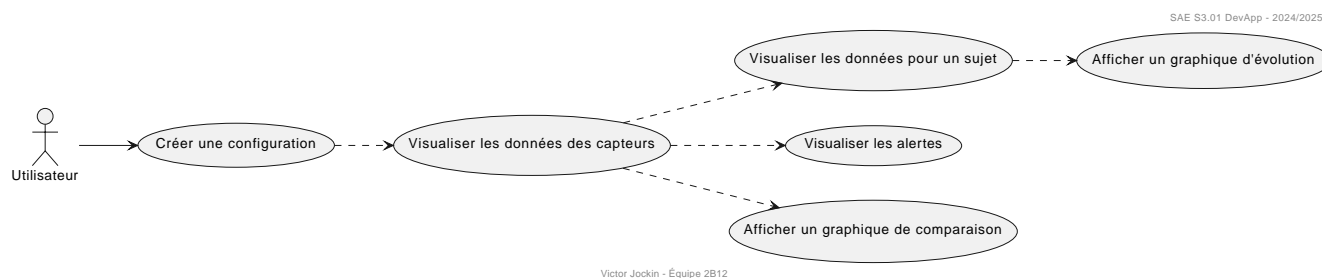
¥ JavaFX assure l'aspect "interface utilisateur" (IHM).

¥ Java assure le traitement des actions utilisateurs et plus généralement le fonctionnement interne de l'application.

¥ Python assure le processus de collecte des données des capteurs.

1.2 Fonctionnalités principales

1.2.1 Diagramme des Cas d'Utilisation



1.2.2 Description des fonctionnalités

¥ Création d'une configuration : Paramétrage des capteurs ^ consulter, des sujets ^ observer, des types de données ^ récupérer et des seuils d'alerte.

¥ Visualisation des données des capteurs : Affichage des relevés en temps réel dans un tableau de bord.

¥ Visualisation des alertes : Signalement dans le tableau de bord des dépassements de seuils d'alerte définis.

¥ Analyse des données : Représentation des données sous forme de graphiques d'évolution ou de comparaison.

2. Architecture de l'application

2.1 Architecture g n rale

2.1.1 Arborescence

Arborescence du projet

```
application_iot/  
! " " resources/  
# ! " " data/  
# # ! " " data.csv  
# # ! " " alert.csv  
# # $ " " ...  
# ! " " mqtt.py  
# $ " " configuration.ini  
! " " src/  
! " " target/  
# ! " " site/  
# # $ " " api docs/  
# # ! " " index.html  
# # $ " " ...  
# $ " " ...  
! " " dependency-reduced-pom.xml  
$ " " pom.xml
```

Description des r pertoires et fichiers

resources : Contient les fichiers et programmes externes n cessaires au fonctionnement de l'application Java / JavaFX.

¥ **data** : Contient les fichiers de donn es g n r s par le programme Python.

! **data.csv** : Le fichier de donn es global.

! **alert.csv** : Le fichier d'alertes.

! Les autres fichiers de donn es chacun associ    un sujet observ .

¥ **mqtt.py** : Le programme Python charg  de la collecte des donn es (client MQTT).

¥ **configuration.ini** : Le fichier de configuration du programme Python.

src : Contient le code source de l'application Java / JavaFX.

target : Contient les fichiers g n r s par le processus de compilation de l'application.

¥ **site/api docs/index.html** : Page principale de la documentation Javadoc de l'API.

pom.xml : Fichier g rant les d pendances et la configuration du projet Maven pour l'application Java / JavaFX.

`dependency-reduced-pom.xml` : Réduction du fichier `pom.xml`.

2.1.2 Sous-systèmes de l'application

Application Java / JavaFX

• Rôle : Gestion de l'interface graphique et mise en relation des différents sous-systèmes et fichiers.

• Tâches réalisées :

- ! Gestion d'une interface de paramétrage d'une configuration.
- ! Lancement et interruption du programme Python chargé de la collecte des données.
- ! Lecture des fichiers de données écrits par le programme Python.
- ! Gestion d'un tableau de bord permettant la visualisation des données des capteurs.

Programme Python

• Rôle : Collecte des données envoyées par les capteurs SOLAREEDGE et AM107.

• Tâches réalisées :

- ! Initialisation en fonction des paramètres définis dans le fichier de configuration.
- ! Réception des données envoyées par les capteurs.
- ! Écriture des données reçues dans des fichiers CSV.

2.1.3 Fichiers utilisés

Fichier de configuration

Le fichier de configuration `configuration.ini` situé sous le répertoire `resources` contient les paramètres de la configuration créés par l'utilisateur au travers de l'interface de l'application Java. Ce fichier est lu par le programme Python à son lancement qui adapte ainsi son comportement en fonction des paramètres spécifiés.

STRUCTURE DU FICHIER

```
[MQTT] ; [1]
broker=mqtt.iut-bagnac.fr
port=1883
topic={{ PRÉFIXE DES TOPIC MQTT }}

[SUBJECTS] ; [2]
subject1={{ SUJET 1 }}
subject2={{ SUJET 2 }}
...

[DATA_TYPE] ; [3]
dataType1={{ TYPE DE DONNÉES 1 }}
dataType2={{ TYPE DE DONNÉES 2 }}
```

```

dataType3={{ TYPE DE DONNÉES 3 }}
...

[THRESHOLD] ; [4]
{{ TYPE DE DONNÉES 1 }}={{ SEUIL }}
{{ TYPE DE DONNÉES 2 }}={{ SEUIL }}
{{ TYPE DE DONNÉES 3 }}={{ SEUIL }}
...

[PARAMS] ; [5]
frequency={{ FRÉQUENCE }}

```

[1] Paramètres de connexion MQTT

¥ **broker** : Adresse du broker MQTT (valeur fixe).

¥ **port** : Port utilisé pour la connexion au broker (port standard MQTT, valeur fixe).

¥ **topic** : Préfixe des topics auxquels le programme Python doit s'abonner.

! Pour accéder aux capteurs AM107, le préfixe correspondant est **AM107/by-room/**.

! Pour accéder aux capteurs SOLAREEDGE, le préfixe correspondant est **solaredge/blagnac/**.

[2] Liste des sujets à observer

¥ **subjectI** : Liste des sujets à observer.

! Pour les capteurs AM107, le nombre de sujets à observer peut aller jusqu'au nombre total de salles disponibles, soit 53.

! Pour les capteurs SOLAREEDGE, le nombre de sujets à observer se limite à 1 : **overview**.

[3] Liste des types de données à récupérer

¥ **dataTypeI** : Liste des types de données à récupérer pour le type de capteurs consulté.

[4] Liste des seuils d'alerte par type de données (capteurs AM107 uniquement)

¥ Cette section indique, pour chaque type de données listé dans la section **DATA_TYPE**, le seuil dont le dépassement déclenchera une alerte.

[5] Paramètres avancés

¥ **frequency** : Fréquence de lecture des données.

! À noter : La valeur pour ce paramètre n'a actuellement aucun impact sur le comportement du programme Python car non traité. La fréquence définie lors du paramétrage de la configuration est cependant prise en compte par le processus de lecture des données de l'application Java.

Fichiers de donn es

Les fichiers de donn es situ s sous le r pertoire `resources/data` sont des fichiers CSV permettant de stocker les donn es des capteurs. Ces fichiers sont cr  s et remplis par le programme Python et lus par l'application Java.

La premi re ligne de chaque fichier CSV contient les en-t tes d crivant la nature des donn es des lignes suivantes (lignes de donn es).

  noter : Dans les fichiers CSV manipul s, le s parateur de donn es utilis  est le point-virgule (;).

Fichier de donn es global

Le fichier `data.csv` correspond au fichier de donn es global. Il contient les derni res donn es re ues pour chaque sujet.

  Dans le cas des capteurs AM107, une ligne de donn es du fichier correspond aux derni res donn es re ues pour une salle.

  Dans le cas des capteurs SOLAREEDGE, la seule ligne de donn es pr sente dans le fichier correspond aux derni res donn es re ues pour le panneau solaire.

Ce fichier est utilis  par l'application Java afin d'afficher dans le tableau de bord les donn es en temps r el pour chaque sujet observ  ainsi que pour g n rer des diagrammes de comparaison des sujets sur un type de donn es.

STRUCTURE DU FICHIER

```
{{ TYPE DE SUJET }};{{ TYPE DE DONNEE 1 }};{{ TYPE_DE DONNEE 2 }}
{{ SUJET 1 }};{{ DERNI RE VALEUR MESUR E }};{{ DERNI RE VALEUR MESUR E }}
{{ SUJET 2 }};{{ DERNI RE VALEUR MESUR E }};{{ DERNI RE VALEUR MESUR E }}
{{ SUJET 3 }};{{ DERNI RE VALEUR MESUR E }};{{ DERNI RE VALEUR MESUR E }}
...
```

Fichier d'alertes (capteurs AM107 uniquement)

Le fichier `alert.csv` correspond au fichier d'alertes. Il contient l'ensemble des alertes d clench es par des d passements de seuils. Une ligne de donn es du fichier correspond donc   une alerte pour un type de donn es et pour une salle.

Ce fichier est utilis  par l'application Java afin d'afficher les alertes en temps r el dans le tableau de bord.

STRUCTURE DU FICHIER

```
room;dataType;threshold;measuredValue
{{ SALLE 1 }};{{ TYPE DE DONN ES }};{{ SEUIL }};{{ VALEUR MESUR E }}
{{ SALLE 2 }};{{ TYPE DE DONN ES }};{{ SEUIL }};{{ VALEUR MESUR E }}
...
```


Les fichiers dont le nom est de la forme **SUJET.csv** correspondent chacun à un fichier de données pour un sujet en particulier. Un fichier de ce type contient l'historique des données reçues pour un sujet.

¥ Dans le cas des capteurs AM107, autant de fichiers sont créés que de sujets sont observés. Les noms de ces fichiers correspondent aux noms des salles observées (exemple : **B101.csv**).

¥ Dans le cas des capteurs SOLAREEDGE, un seul fichier nommé **overview** est créé.

Ces fichiers sont exploités par l'application Java afin de construire des graphiques décrivant l'évolution des valeurs pour un type de données.

STRUCTURE DU FICHIER

```
{{ TYPE DE SUJET }};{{ TYPE DE DONNEE 1 }};{{ TYPE DE DONNEE 2 }}
{{ SUJET }};{{ VALEUR MESURE À L'INSTANT T0 }};{{ VALEUR MESURE À L'INSTANT T0 }}
{{ SUJET }};{{ VALEUR MESURE À L'INSTANT T1 }};{{ VALEUR MESURE À L'INSTANT T1 }}
{{ SUJET }};{{ VALEUR MESURE À L'INSTANT T2 }};{{ VALEUR MESURE À L'INSTANT T2 }}
...
```

2.1.4 Interactions entre sous-systèmes et fichiers

1. Écriture du fichier de configuration par l'application Java

! Après le paramétrage d'une configuration par l'utilisateur dans l'interface graphique, l'application Java crée un fichier **configuration.ini** sous le répertoire **resources** décrivant la configuration créée.

! À noter : À cette étape, si un fichier de configuration existe déjà, celui-ci est remplacé par le fichier de configuration nouvellement créé. Aucun mécanisme d'historisation ou de sauvegarde des fichiers de configurations n'a été mis en place.

2. Lancement du programme Python par l'application Java

! Une fois le fichier de configuration créé, l'application Java démarre le processus de collecte des données en lançant en exécution le programme Python.

3. Collecte des données par le programme Python

! Au lancement, le programme Python lit le fichier de configuration définissant son comportement.

! Une fois lancé, il attend jusqu'à interruption les données envoyées par les sujets (capteurs).

! À chaque réception de données, celles-ci sont enregistrées dans les fichiers de données correspondants.

4. Lecture des fichiers de données par l'application Java

! En parallèle de l'exécution du programme Python, l'application Java lit à intervalle régulier (fréquence définie dans le fichier de configuration) les fichiers de données.

! Les données lues sont ensuite stockées dans des structures de données puis transmises au tableau de bord de l'application pour affichage.

5. Interruption du programme Python par l'application Java

! Lorsque le tableau de bord de l'application est fermé par l'utilisateur, le programme Python est automatiquement arrêté.

! À noter : Après arrêt du processus de collecte des données, le fichier de configuration ainsi que les fichiers de données écrits sont conservés. Ils seront recrasés lors de la prochaine exécution de l'application.

2.2 Ressources externes

2.2.1 API utilisées

JavaFX

• R  les :

! Conception de l'UI avec le module `javafx-fxml` (cr  ation d'interfaces utilisateur via des fichiers FXML).

! Prise en charge et gestion de l'interface graphique dans l'application.

• Version utilis  e : 17

• Site officiel de JavaFX : [JavaFX - Home](#)

• Documentation officielle : [Oracle - JavaFX Documentation](#)

OpenCSV

• R  le : Lecture des fichiers de donn  es au format `CSV` g  n  r  s par le programme python collecteur de donn  es.

• Version utilis  e : 5.5.2

• Site officiel de JavaFX : [OpenCSV - About / Opencsv Users Guide](#)

• Documentation officielle : [OpenCSV - About / Developer Documentation](#)

2.2.2 Plugins utilis  s

JavaFX Maven Plugin

• R  le : Packaging et ex  cution de l'application JavaFX.

• Version utilis  e : 0.0.8

• Site officiel de Maven Repository : [Maven Repository - JavaFX Maven Plugin Maven Mojo](#)

• Lien vers le d  p  t GitHub du plugin : [GitHub - Maven plugin for JavaFX](#)

Apache Maven Shade Plugin

• R  le : Cr  ation d'un ex  cutable au format `JAR` contenant toutes les d  pendances n  cessaires au fonctionnement de l'application.

• Version utilis  e : 3.4.1

¥ Site officiel d'Apache Maven : [Apache Maven Project - Apache Maven Shade Plugin](#)

Apache Maven Javadoc Plugin

¥ R  le : G  n  ration de la documentation du projet Java avec *Javadoc*.

¥ Version utilis  e : 3.4.1

¥ Site officiel d'Apache Maven : [Apache Maven Project - Apache Maven Javadoc Plugin](#)

2.3 Structuration de l'application Java

2.3.1 Patterns mis en oeuvre

Architecture MVC

L'application Java repose sur une architecture MVC (Mod  le-Vue-Contr  leur / Model-View-Controller) permettant la s  paration des couches de pr  sentation, de logique m  tier et de traitement des actions utilisateur.

Pr  sentation

¥ Composante MVC associ  e : Vue (*View*).

¥ R  le :

- ! Afficher les donn  es envoy  es par le Contr  leur.
- ! Permettre    l'utilisateur d'interagir avec l'interface graphique.

Logique m  tier

¥ Composante MVC associ  e : Mod  le (*Model*).

¥ R  le :

- ! Repr  senter les donn  es manipul  es par l'application.
- ! Appliquer des r  gles de gestion sur les donn  es.
- ! Fournir une interface permettant l'acc  s aux donn  es et leur mise    jour.
- ! Notifier le Contr  leur apr  s une mise    jour des donn  es.

Traitement des actions utilisateur

¥ Composante MVC associ  e : Contr  leur (*Controller*).

¥ R  le :

- ! Effectuer des op  rations sur le Mod  le en fonction des actions utilisateur.
- ! Mettre    jour la Vue afin de refl  ter les changements dans le Mod  le.

Configuration

La classe modélisant la configuration paramétrée par l'utilisateur (`Configuration.java`) est implémentée en *Singleton* en ce que l'application permet actuellement de définir une seule configuration à la fois. En d'autres termes, lorsqu'une nouvelle configuration est définie, celle-ci écrase automatiquement la configuration précédente.

Une implémentation selon le patron *Singleton* permet ainsi à cette classe de fournir une méthode donnant accès à l'unique instance de la configuration.



Cette implémentation serait susceptible d'évoluer si un mécanisme d'historisation ou de sauvegarde des différentes configurations définies par l'utilisateur était mis en place.

2.3.2 Packages

Arborescence des packages

Les packages de l'application Java sont situés sous le répertoire `src/main/java`.

```
application
! " " config
! " " control
! " " data
! " " enums
! " " model
! " " styles
! " " thread
! " " tools
$ " " view
```

Description des packages et de leur contenu

`application` : Package "racine" de l'application.

¥ `ApplicationMainFrame` : Contrôleur de dialogue du menu principal (fenêtre principale de l'application).

¥ `Main` : Classe principale de l'application.

`application.config` : Package des classes manipulant le fichier de configuration.

¥ `ConfigurationFileWriter` : Classe permettant d'écrire un fichier de configuration.

`application.control` : Package des contrôleurs de dialogue (Cf. [Architecture MVC](#)).

¥ `ConfirmationFileForm` : Contrôleur de dialogue du formulaire de paramétrage de la configuration.

¥ `DataVisualizationPane` : Contrôleur de dialogue du tableau de bord (fenêtre de visualisation des données).

¥ À noter : La classe `ApplicationMainFrame` située dans le package `application` pourrait être déplacée dans ce package en ce qu'il s'agit d'un contrôleur de dialogue.

`application.data` : Package des classes relatives aux données.

¥ `DataCollector` : Classe de gestion du processus de collecte des données.

¥ `DataLoader` : Classe d'accès aux fichiers de données.

¥ `DataTypeUtilities` : Classe utilitaire fournissant des méthodes relatives aux types de données (ex : formatage de noms).

`application.enums` : Package des énumérations.

¥ `Room` : Classe d'énumération des salles existantes.

¥ `RoomDataType` : Classe d'énumération des types de données des salles.

¥ `Sensor` : Classe d'énumération des types de capteurs (`AM107` / `SOLAREEDGE`).

¥ `SolarPanelDataType` : Classe d'énumération des types de données des panneaux solaires.

`application.model` : Package des classes modèles (Cf. [Architecture MVC](#)).

¥ `Configuration` : Classe modèle représentant une configuration.

¥ `DataRow` : Classe modèle représentant une ligne de données (Cf. [Fichiers de données](#)).

`application.styles` : Package des classes de stylisation de l'interface graphique.

¥ `FontLoader` : Classe d'accès aux typographiques (fonts) utilisés dans l'interface graphique.

`application.thread` : Package des threads.

¥ `CsvReaderTask` : Thread chargé de lire le [fichier de données global](#) (`data.csv`) et le [fichier d'alertes](#) (`alert.csv`).

¥ `UpdateAlertDisplayTask` : Thread chargé de la mise à jour de l'affichage des alertes dans le tableau de bord.

¥ `UpdateDataDisplayTask` : Thread chargé de la mise à jour de l'affichage des données dans le tableau de bord.

`application.tools` : Package des classes utilitaires.

¥ `DataFileReading` : Classe utilitaire fournissant des méthodes de lecture de fichiers de données.

¥ `GraphGenerator` : Classe utilitaire fournissant des méthodes de génération de graphiques.

¥ `TextUtilities` : Classe utilitaire fournissant des méthodes relatives à des éléments textuels (NON UTILISÉ).

`application.view` : Package des contrôleurs de vue (Cf. [Architecture MVC](#)).

¥ `ApplicationMainFrameViewController` : Contrôleur de vue du menu principal.

¥ `ConfigurationFileFormViewController` : Contr^mleur de vue du formulaire de param^ẽtrage de la configuration.

¥ `DataVisualisationPaneViewController` : Contr^mleur de vue du tableau de bord.

2.4 Sp^ẽcifications techniques

2.4.1 Conventions de nommage

!

L'anglais est utilis^ẽ pour tous les noms de classes, de variables, de packages et de vues.

Nommage des classes

Les noms de classes Java sont format^ẽs en Upper Camel Case.

Contr^mleurs de dialogue

¥ R[•]gle : Nom de la vue FXML en Upper Camel Case.

¥ Exemple : Contr^mleur de dialogue associ^ẽ ^ la vue `configurationFileForm.fxml` "
`ConfigurationFileForm`.

Contr^mleurs de vue

¥ R[•]gle : Nom de la vue FXML en Upper Camel Case + `ViewController`.

¥ Exemple : Contr^mleur de la vue `configurationFileForm.fxml` "
`ConfigurationFileFormViewController`.

Classes utilitaires

¥ R[•]gle : Objet manipul^ẽ + `Utilities`.

¥ Exemple : Classe utilitaire fournissant des m^ẽthodes relatives aux types donn^ẽes "
`DataTypeUtilities`.

Classes mod[•]le

¥ R[•]gle : Nom de l'objet repr^ẽsent^ẽ.

¥ Exemple : Classe mod[•]le repr^ẽsentant une configuration " `Configuration`.

Classes d^ẽnum^ẽration

¥ R[•]gle : Nom du type d'objet ^ẽnum^ẽrer^ẽ.

¥ Exemple : d^ẽnum^ẽration des salles " `Room`.

!

Les classes d^ẽnum^ẽration, pouvant avoir des noms identiques ^ ceux de classes mod[•]les, ont ^ẽt^ẽ plac^ẽes dans un package `enums` d^ẽdi^ẽ afin d^ẽviter toute confusion.

Threads

¥ R gle : Nom de la t che r ali e par le thread + Task.

¥ Exemple : Thread charg  de la mise   jours de l affichage des donn es dans le tableau de bord
" UpdateDataDisplayTask.

Classes avec m thodes statiques

¥ R gle : Objet concern  + Verbe d action.

¥ Exemple : Classe d acc s aux donn es " DataLoader.

Nommage des variables

Les noms de variables Java sont format s en Lower Camel Case.

Variables  ph m res

S applique aux variables de type indice ou aux compteurs de boucles.

¥ R gle : Nom "court".

¥ Exemples :

! Compteur de boucle for " i.

! Entr e couremment trait  dans une boucle de type for-each parcourant le contenu d un dictionnaire " m.

Remarque : Les noms de ces variables peuvent  tre plus explicites si besoin.

Variables r currentes

S applique aux variables et aux collections utilis es   plusieurs endroits dans une classe.

¥ R gle : Nom explicite.

¥ Exemples :

! Cha ne de caract res d crivant le pr fixe d un topic MQTT " topicPrefix.

! Liste de types de donn es " dataTypeList.

Variables de composants graphiques

S applique uniquement   une variable d un contr leur de vue correspondant   un  l ment graphique de la vue associ e.

¥ R gle : R le du composant graphique + fventuellement type du composant.

¥ Exemples :

! Composant graphique de type Bouton (Button) " button.

! Conteneur de graphiques de type VBox " graphContainerVBox ou graphContainer.

Paramètres

S'applique uniquement aux paramètres de fonctions et de méthodes.

Œ Règle : **p** + Nom explicite en Upper Camel Case.

Œ Exemples :

! Liste de types de données passés en paramètre d'une fonction / méthode " **pDataTypeList** (Cf. [Variables récurrentes](#)).

! Composant graphique de type Bouton (**Button**) passé en paramètre d'une fonction / méthode " **pButton** (Cf. [Variables de composants graphiques](#)).

Nommage des packages

Œ Règle : Nom court décrivant le type des classes contenues par le package en Lower Camel Case.

Œ Exemple : Package des classes utilitaires " **tools**.

Nommage des vues FXML

Œ Règle : Nom de la vue en Lower Camel Case.

Œ Exemple : Vue du formulaire de paramétrage du fichier de configuration " **configurationFileForm.fxml**.

2.4.2 Conventions de commentaire



Les commentaires des classes Java sont entièrement rédigés en français.

Commentaire d'en-tête de classe

Modèle

```
/**
 * [ TYPE DE CLASSE + RÔLE ]
 *
 * Date de dernière modification :
 * - [ DATE ] -
 *
 * @author [ DÉVELOPPEUR ]
 * ...
 * - [ NOM DE L'ÉQUIPE DE DÉVELOPPEMENT ] -
 */
```


Exemple

Classe : `ConfigurationFileForm`

```
/**
 * Contrôleur de dialogue du formulaire de paramétrage
 * d'un fichier de configuration.
 *
 *
 * Date de dernière modification :
 * - Mardi 10 décembre 2024 -
 *
 * @author Victor Jockin
 * - équipe G2B12 -
 */
```

Commentaire d'entête de méthode

Modèle

```
/**
 * [ RÔLE DE LA MÉTHODE ]
 * @param pParam1 [ DESCRIPTION DU PARAMÈTRE ]
 * @param pParam2 [ DESCRIPTION DU PARAMÈTRE ]
 * ...
 * @return [ DESCRIPTION DE LA VALEUR RETOURNÉE ]
 * @throws EXCEPTION [ CONDITION DE LEVÉE DE L'EXCEPTION ]
 */
```

Exemple

Méthode : `ConfigurationFileForm.isThresholdValid(RoomDataType, double)`

```
/**
 * Indique si un seuil d'alerte pour un type de données de salle est valide.
 * @param pRoomDataType un type de données de salle
 * @param pThreshold un seuil d'alerte
 * @return true si le seuil d'alerte est valide, false sinon
 */
```

2.4.3 Structures r  currentes de classes

Dans cette section, on suppose avoir l'arborescence suivante :

```
src/  
! " " main/  
# ! " " java/  
# # $ " " application/  
# # ! " " control/  
# # # ! " " Example.java [1]  
# # # $ " " ...  
# # ! " " view/  
# # # ! " " ExampleViewController.java [2]  
# # # $ " " ...  
# # $ " " ...  
# $ " " resources/  
# $ " " application/  
# ! " " view/  
# # ! " " example.fxml [3]  
# # $ " " ...  
# $ " " ...  
$ " " ...
```

1. Contr  leur de dialogue d'exemple.
2. Contr  leur de vue d'exemple.
3. Vue FXML d'exemple.

Contr  leurs de dialogue

```
package application.control ;  
  
import application.view.ExampleViewController ;  
  
...  
  
public class Example  
{  
    // d  claration des constantes  
    // -----  
  
    private static final double MIN_WINDOW_WIDTH    = ... ;    // largeur minimale de  
    la fen  tre  
    private static final double MIN_WINDOW_HEIGHT  = ... ;    // hauteur minimale de  
    la fen  tre  
    ...  
  
    // d  claration des attributs
```

```

Ê // -----

Ê // attributs relatifs au contr leur de dialogue
Ê private Stage eStage ;
Ê private ExampleViewController eViewController ;

Ê // attributs relatifs au Module
Ê ...

Ê /**
Ê  * Constructeur : charge la fen tre d'exemple.
Ê  */
Ê public Example(Stage _parentStage)
Ê {
Ê     try
Ê     {
Ê         // initialisation des attributs relatifs au Module
Ê         ...

Ê         // initialisation d'un nouveau stage pour la fen tre d'exemple
Ê         this.eStage = new Stage() ;
Ê         this.eStage.initOwner(_parentStage) ;
Ê         this.eStage.initModality(Modality.WINDOW_MODAL) ;

Ê         // chargement de la vue FXML de la fen tre d'exemple
Ê         FXMLLoader fxmLoader = new
FXMLLoader(ExampleViewController.class.getResource("exemple.fxml")) ;

Ê         // initialisation de la sc ne
Ê         Scene scene = new Scene(fxmLoader.load(), MIN_WINDOW_WIDTH,
MIN_WINDOW_HEIGHT) ;
Ê         this.eStage.setScene(scene) ;
Ê         this.eStage.setTitle("Exemple") ;

Ê         // initialisation du contr leur de vue
Ê         this.eViewController = fxmLoader.getController() ;
Ê         this.eViewController.setStage(this.eStage) ;
Ê         this.eViewController.setCffDialogController(this) ;
Ê         this.eViewController.initializeView() ;

Ê         // application des styles   la sc ne
Ê
this.eStage.getScene().getStylesheets().add(getClass().getResource("/application/style
/e.css").toExternalForm()) ;
Ê     }
Ê     catch (Exception e)
Ê     {
Ê         e.printStackTrace() ;
Ê     }
Ê }

```

```

Ê // accesseurs
Ê // -----

Ê ...

Ê // méthodes publiques de gestion du dialogue
Ê // -----

Ê /**
Ê  * Effectue le dialogue d'exemple.
Ê  */
Ê public void doExampleDialog() { this.eViewController.displayDialog() ; }

Ê ...

Ê // méthodes privées
Ê // -----

Ê ...
}

```

Contr^mleurs de vue

```

package application.view ;

import application.control.Example ;

...

public class ExampleViewController
{
Ê // déclaration des constantes
Ê // -----

Ê ...

Ê // déclaration des attributs
Ê // -----

Ê // attributs relatifs au contrmleur de vue
Ê private Stage stage ;
Ê private Example eDialogController ;

Ê // attributs relatifs au Mod•le
Ê ...
}

```

```

Ê // Éléments graphiques de la vue FXML (ordonnés par ordre d'apparition)
Ê // -----

Ê @FXML private ... ;
Ê ...

Ê // méthodes d'initialisation du contrôleur de vue
Ê // -----

Ê /**
Ê  * Définit le stage de la vue.
Ê  * @param _stage    un stage
Ê  */
Ê public void setStage(Stage _stage)
Ê {
Ê     this.stage = _stage ;
Ê }

Ê /**
Ê  * Définit le contrôleur de dialogue de la vue.
Ê  * @param _eDialogController un contrôleur de dialogue
Ê  */
Ê public void setEDialogController(Example _eDialogController)
Ê {
Ê     this.eDialogController = _eDialogController ;
Ê }

Ê /**
Ê  * Initialise la vue.
Ê  */
Ê public void initializeView()
Ê {
Ê     // initialisation des éléments graphiques de la vue
Ê     ...
Ê }

Ê /**
Ê  * Affiche la fenêtre.
Ê  */
Ê public void displayDialog()
Ê {
Ê     this.stage.showAndWait() ;
Ê }

Ê /**
Ê  * Gère la fermeture de la fenêtre.
Ê  * @param e un Événement de fenêtre
Ê  */

```

```

Ê private void closeWindow(WindowEvent e)
Ê {
Ê     this.doClose() ;
Ê     e.consume() ;
Ê }

Ê // mŁthodes de traitement des actions utilisateur
Ê // -----

Ê /**
Ê  * Ferme la fenŁtre.
Ê  */
Ê @FXML
Ê private void doClose()
Ê {
Ê     this.stage.close() ;
Ê }

Ê // mŁthodes privŁes
Ê // -----

Ê ...
Ê }

```

2.4.4 Threads utilisŁs

Thread de rŁcupŁration des donnŁes

Le thread `CsvReaderTask` lis, Ā intervalle de temps rŁgulier (cf. frŁquence dŁfinie dans la configuration), le fichier de donnŁes global (`data.csv`) et le fichier dŁalertes (`alert.csv`). Il notifie ensuite le contrŁleur de dialogue du tableau de bord `DataVisualisationPane` que les donnŁes ont ŁtŁ actualisŁes.

MŁthode de lecture du fichier de donnŁes global

```

Ê /**
Ê * Lis le fichier de donnŁes.
Ê */
private void readDataFile()
{
Ê     Map<String, Map<String, String>> dataMap = new HashMap<>() ;
Ê     try (CSVReader csvReader = new CSVReaderBuilder(new
FileReader(DataLoader.getAllRoomDataFile()))
Ê         .withCSVParser(new CSVParserBuilder().withSeparator(delimiter).build())
Ê         .build()
Ê     ) {
Ê         String[] header = csvReader.readNext() ;

```

```

Ê      String[] values = null ;
Ê      while ((values = csvReader.readNext()) != null && !values[0].equals(""))
Ê      {
Ê          dataMap.put(values[0], new HashMap<String, String>()) ;
Ê          for (int i = 1; i < values.length; i++)
Ê          {
Ê              dataMap.get(values[0]).put(header[i], values[i]) ;
Ê          }
Ê      }
Ê      this.dvpDialogController.setDataMap(dataMap) ; // LE CONTRiLEUR DE DIALOGUE
EST NOTIFIÉ DE LA MISE À JOUR
Ê  }
Ê  catch (FileNotFoundException e) { throw new RuntimeException(e) ; }
Ê  catch (IOException e) { throw new RuntimeException(e) ; }
Ê  catch (CsvValidationException e) { throw new RuntimeException(e) ; }
Ê  }

```

Méthode de lecture du fichier d'alertes

```

/**
Ê* Lis le fichier d'alertes.
Ê*/
private void readAlertFile()
{
Ê  Map<String, Map<String, String>> alertMap = new HashMap<>() ;
Ê  try (CSVReader csvReader = new CSVReaderBuilder(new
FileReader(DataLoader.getAlertFile()))
Ê      .withCSVParser(new CSVParserBuilder().withSeparator(delimiter).build())
Ê      .build()
Ê  ) {
Ê      String[] header = csvReader.readNext() ;
Ê      String[] values = null ;
Ê      while ((values = csvReader.readNext()) != null && !values[0].equals(""))
Ê      {
Ê          alertMap.put(values[0], new HashMap<String, String>()) ;
Ê          for (int i = 1; i < values.length; i++)
Ê          {
Ê              alertMap.get(values[0]).put(header[i], values[i]) ;
Ê          }
Ê      }
Ê      this.dvpDialogController.setAlertMap(alertMap) ; // LE CONTRiLEUR DE DIALOGUE
EST NOTIFIÉ DE LA MISE À JOUR
Ê  }
Ê  catch (FileNotFoundException e) { throw new RuntimeException(e) ; }
Ê  catch (IOException e) { throw new RuntimeException(e) ; }
Ê  catch (CsvValidationException e) { throw new RuntimeException(e) ; }
Ê  }

```

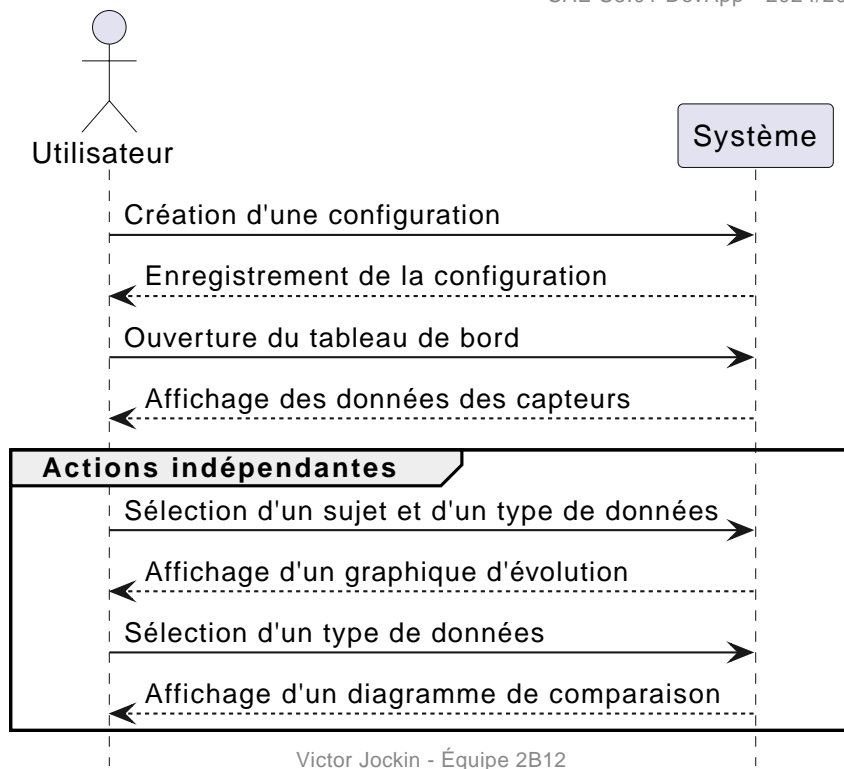
Threads de mise à jour de l'affichage

Les threads `UpdateDataDisplayTask` et `UpdateAlertDisplayTask` sont appelés à chaque notification d'actualisation des données de la part du thread de récupération des données (`CsvReaderTask`). Ils sont respectivement chargés de la mise à jour de l'affichage des données et des alertes dans le tableau de bord.

3. Conception et mise en oeuvre des fonctionnalités

3.1 Diagramme de Séquence Système

SAE S3.01 DevApp - 2024/2025



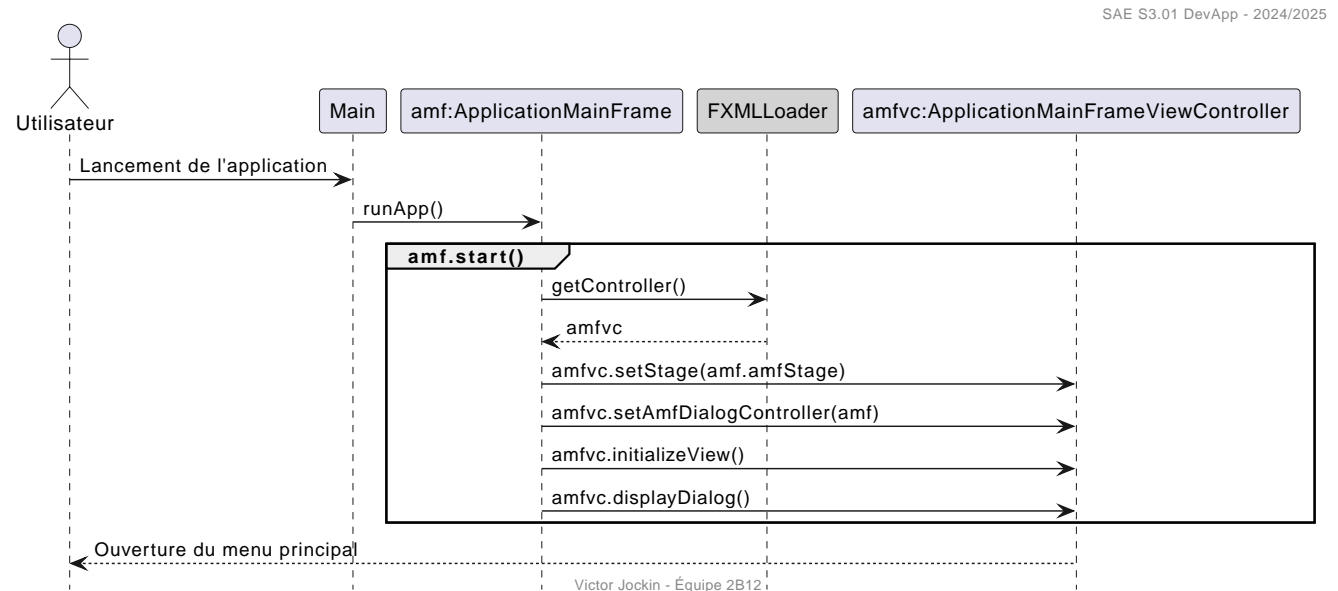
Victor Jockin - Équipe 2B12

3.2 Implémentation des fonctionnalités

3.2.1 Menu principal

Ouverture du menu principal

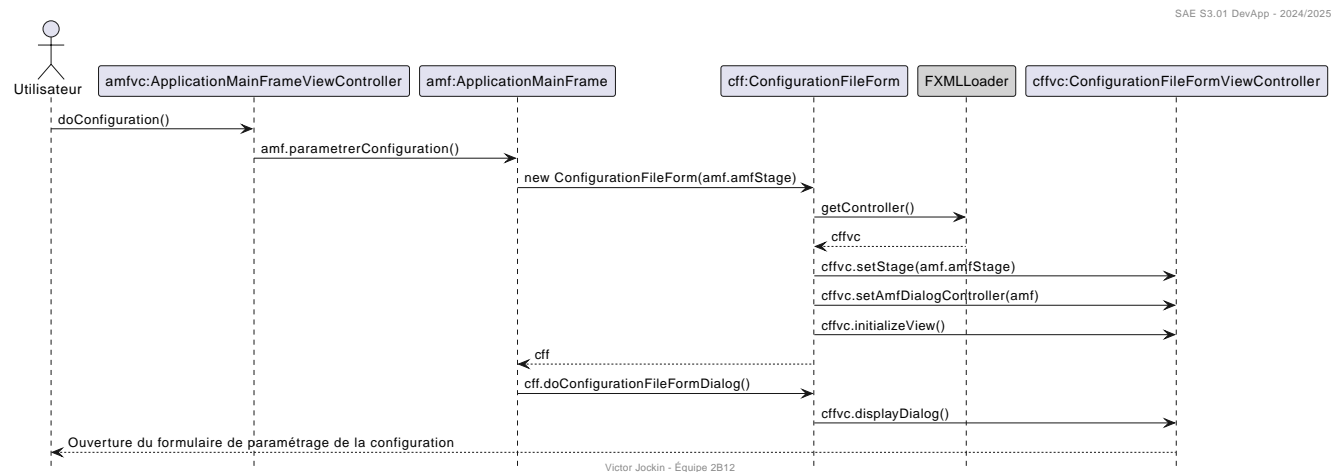
Diagramme de Séquence



3.2.2 Formulaire de paramétrage de la configuration

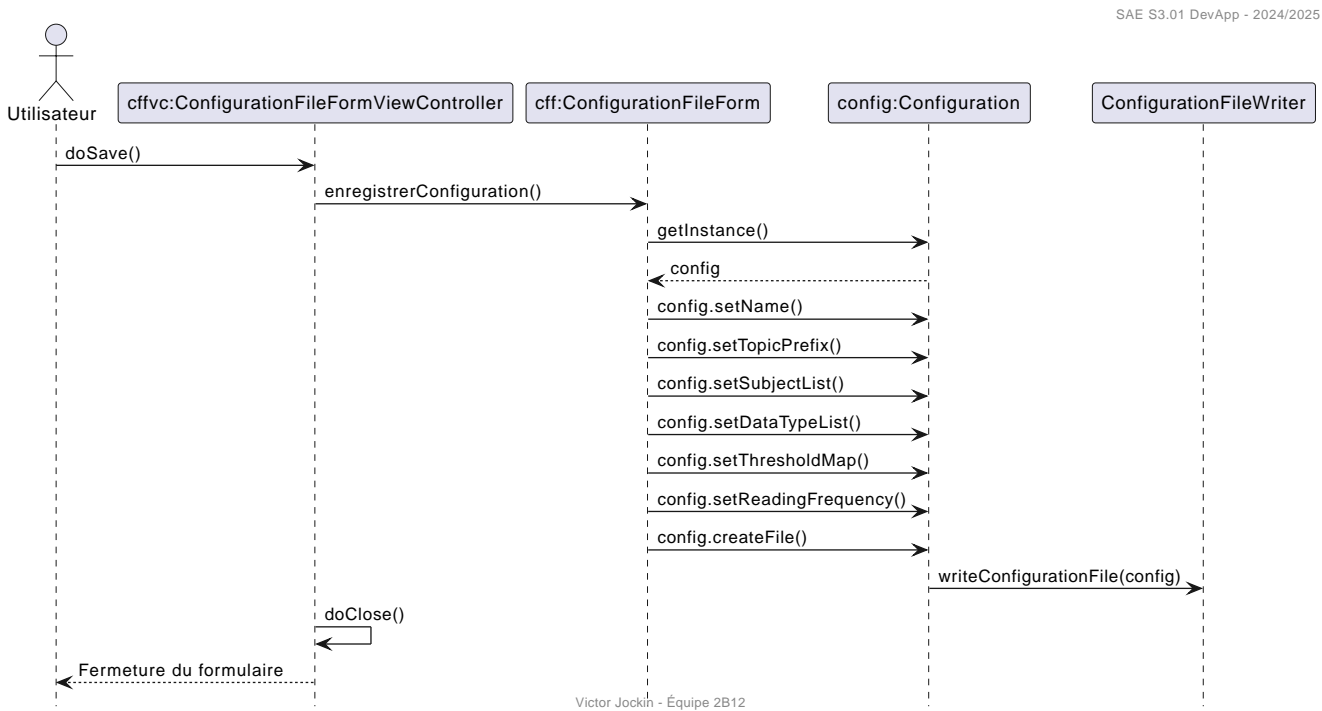
Ouverture du formulaire

Diagramme de Séquence



Enregistrement d'une configuration

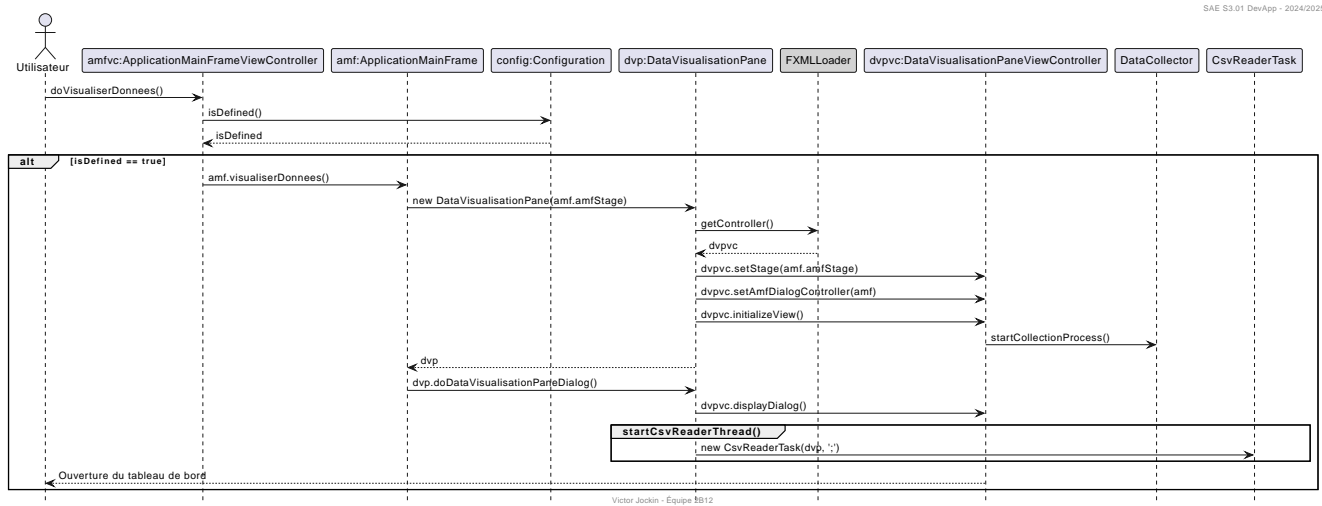
Diagramme de S quence



3.2.3 Tableau de bord

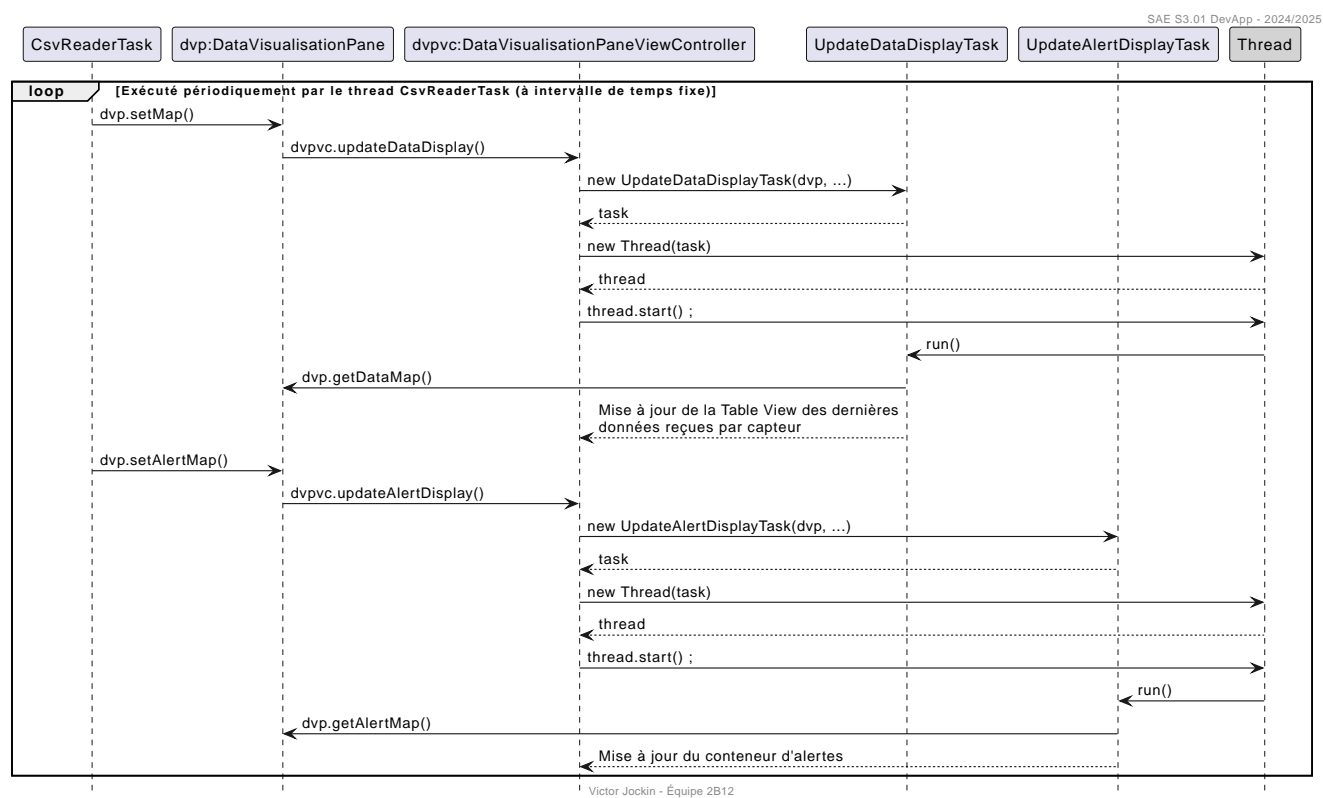
Ouverture du tableau de bord

Diagramme de S quence



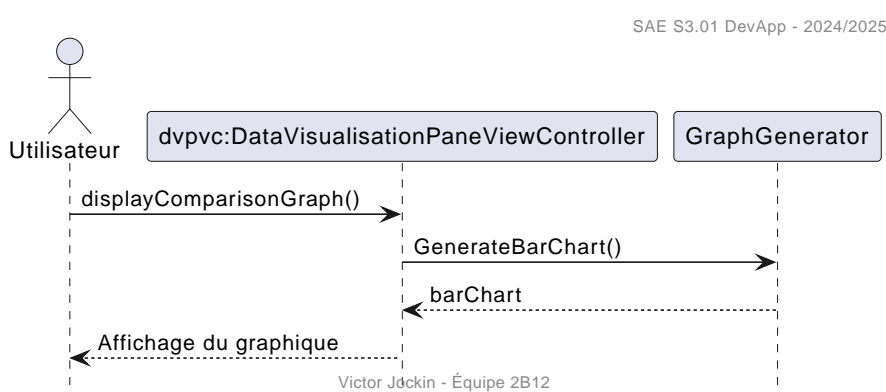
Mise à jour automatique des données affichées

Diagramme de Séquence



Affichage d'un graphique de comparaison

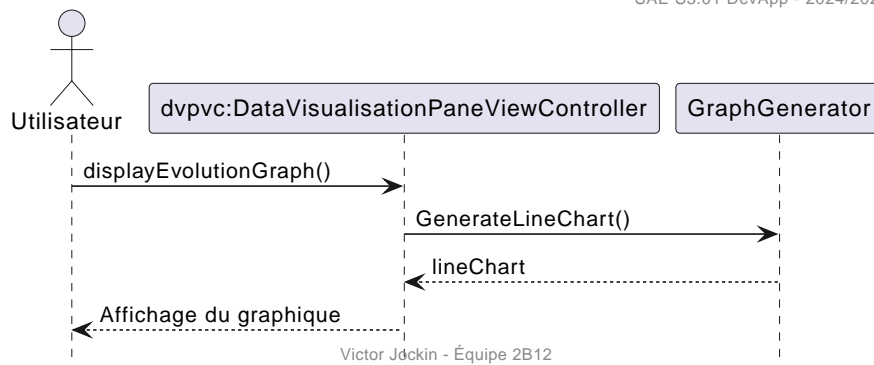
Diagramme de Séquence



Affichage d'un graphique d'évolution

Diagramme de Séquence

SAE S3.01 DevApp - 2024/2025



4. Procédures d'installation

4.1 Installation pour développement

4.1.1 Prérequis

1. Installer l'environnement de développement Java

- ! Télécharger le JDK 17 (ou version compatible) depuis le site officiel d'Oracle : [Oracle - Java Downloads](#).
- ! Installer le JDK en suivant les instructions indiquées par l'installateur.
- ! Si nécessaire, ajouter le chemin vers le JDK à la variable d'environnement `PATH`.
- ! Dans un terminal, vérifier l'installation avec la commande `java -version` ou `java --version`.

2. Installer Apache Maven

- ! Télécharger Maven (archive ZIP) depuis le site officiel d'Apache Maven : [Apache Maven Project - Downloading Apache Maven](#).
- # Pour une installation sur Linux ou Mac OS, télécharger la *Binary tar.gz archive*.
- # Pour une installation sur Windows, télécharger la *Binary zip archive*.
- ! Ajouter le chemin vers Maven à la variable d'environnement `PATH`.
- ! Dans un terminal, vérifier l'installation avec la commande `mvn -version`, `mvn --version` ou `mvn -v`.

3. Configurer un IDE

- ! Si nécessaire, installer des plugins de prise en charge de Maven et JavaFX dans l'IDE utilisé pour le développement.

4.1.2 étapes d'installation

1. Cloner le dépôt du projet

- ! Accéder au dépôt GitHub du projet : [GitHub - SAE S3.01 DevApp](#)
- ! Cloner le dépôt du projet via la commande :

```
git clone https://github.com/IUT-Blagnac/sae-3-01-devapp-2024-2025-g2b12.git
```

- ! Accéder au répertoire du projet Java situé sous `solution\iot\application_iot` via la commande :

```
cd solution\iot\application_iot
```

2. Construire le projet avec Maven

- ! Supprimer les fichiers et ressources précédemment compilés avec la commande `mvn clean`

puis compiler le projet Java via la commande `mvn install`. Il est également possible d'utiliser directement la commande `mvn clean install`.

3. Exécuter l'application depuis Maven

! Exécuter le projet JavaFX via la commande `mvn javafx:run`.

4.2 Installation pour utilisation

4.2.1 Prérequis

1. Installer le Java Runtime Environment (JRE)

! Vérifier que Java est installé sur la machine en exécutant la commande `java -version` dans un terminal.

! Si Java n'est pas installé, télécharger et installer le JRE 8 ou version ultérieure depuis le site officiel de Java : [Java - Télécharger Java](#).

2. Installer Python 3

! Vérifier que Python en version 3 est installé sur la machine en exécutant la commande `python -version` ou `python3 -version` dans un terminal.

! Si Python n'est pas installé, télécharger et installer la dernière version disponible sur le site officiel de Python : [Python - Downloads](#).

4.2.2 étapes d'installation

1. Télécharger l'application

! Télécharger l'archive de l'application (fichier ZIP) située sous le répertoire `livrables/IoT` du dépôt GitHub du projet : [GitHub - Livrables IoT](#)

Pour une installation sur Mac OS, préférez l'archive `application_jar_mac_os.zip`.

Pour une installation sur Windows ou Linux, préférez l'archive `application_jar_windows.zip`.

2. Décompresser l'archive de l'application

! Décompresser l'archive téléchargée dans un répertoire à l'aide d'un outil de décompression tel que WinRAR ou 7-Zip.

! L'arborescence de l'application après décompression doit ressembler à ceci :

```
application/
|-- ressources/
|   |-- data/
|   |-- configuration.ini
|   |-- mqtt.py
|-- application_iot-1.0-SNAPSHOT-shaded.jar
```

4.2.3 étapes de lancement

1. Lancer l'application dans le gestionnaire de fichiers

! Lancer l'exécutable `application_iot-1.0-SNAPSHOT-shaded.jar` en double-cliquant sur celui-ci.

! *Le menu principal de l'application devrait alors apparaître à l'écran.*

2. Lancer l'application en ligne de commande

! Ouvrir un terminal et se placer dans le répertoire `application` à l'aide de la commande `cd`.

! Lancer ensuite l'exécutable de l'application via la commande :

```
java -jar application_iot-1.0-SNAPSHOT-shaded.jar
```

! *Le menu principal de l'application devrait alors apparaître à l'écran.*

Étudiants
Nolhan Biblocque
Léo Guinvarc'h
Victor Jockin
Mathys Laguilliez
Mucahit Lekesiz

Enseignant
André Péninou

Formation
BUT Informatique
2^{ème} Année
Promotion 2024-2025

Établissement
IUT de Blagnac
Université Toulouse II Jean Jaurès (31)