

Rapport Technique

Développeur assistant ingénieur projet Belfort e-Start

Laboratoire Connaissance et Intelligence Artificielle Distribuées

Sous la tutelle de l'Université de Technologie Belfort-Montbéliard (UTBM) et
de l'Université de Bourgogne-Franche-Comté (UBFC)

BELFORT
e-start

Maxime THEVENEAU

BUT3 Informatique (IUT de Belfort-Montbéliard 2023-2024)

Tuteur en entreprise : **GAUD Nicolas**

Professeur référent : **GUYEUX Christophe**

Sommaire

Introduction.....	4
1. Définition du projet.....	5
2. Installation du projet.....	6
2.1. Technologies utilisés.....	6
2.2. Récupération du code source.....	6
2.3. Configuration locale.....	7
2.3.1. Configuration API Spring.....	7
2.3.2. Configuration Client VueJs.....	7
2.4. Configuration docker.....	8
3. Structure du projet.....	9
3.1. Conception générale.....	9
3.2. Architecture Backend.....	10
3.3. Architecture Frontend.....	12
3.4. Modèle conceptuel de données.....	13
4. Fonctionnalités implémentées.....	14
4.1. Backend.....	14
4.1.1. Gestion des appareils.....	14
4.1.2. Gestion de la maintenance.....	15
4.1.3. Gestion des données.....	15
4.1.4. Gestion de l'export de données.....	16
4.1.5. Gestion des dashboards Grafana.....	17
4.1.6. Gestion des rôles.....	17
4.1.7. Gestion des organisations.....	18
4.2. Frontend.....	20
4.2.1. Axios.....	20
4.2.2. Routeur.....	21
4.2.3. Dashboards.....	21
Table des figures.....	23
Table des annexes.....	24

Introduction

Ce document fait office de rapport technique complétant le rapport du stage réalisé au Laboratoire Connaissance et Intelligence Artificielle Distribuées (CIAD) dans le cadre d'un BUT3 Informatique (parcours A) à l'IUT de Belfort-Montbéliard du 15 janvier 2024 au 22 avril 2024.

Le sujet abordé est celui du développement du module de supervision du système de management du projet E-START. Ainsi, à travers ce rapport, sera exposée la manière d'appréhender le projet, la façon dont il a été conçu et les problèmes rencontrés.

Ce document s'adresse donc à des lecteurs ayant des connaissances techniques et aux futur développeurs de mises à jour ultérieures.

1. Définition du projet

Le projet Belfort E-Start vise à la création d'une communauté d'énergie renouvelable basée sur le principe d'autoconsommation collective avec la production et la consommation d'énergie locale. Ainsi des actifs de production photovoltaïque (ombrières photovoltaïques) et des actifs de stockage par batterie et hydrogène seront installés sur le Techn'Hom Belfortain.

L'objectif du CIAD est de concevoir un système de management de l'énergie (EMS) composé de différents modules¹ afin de gérer l'ensemble des appareils du projet et les données mesurées et enregistrées.

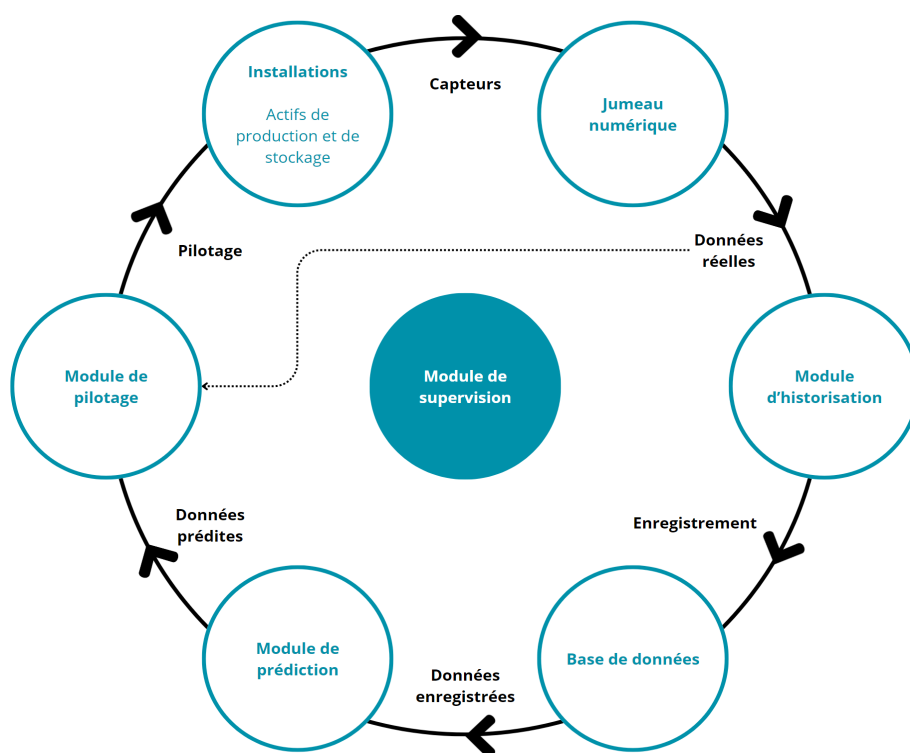












Figure 1: Schéma du fonctionnement du système de management de l'énergie

¹ cf. [Annexes](#)

2. Installation du projet

2.1. Technologies utilisés


-  Docker: <https://www.docker.com/get-started>
-  Flux: <https://docs.influxdata.com/flux/v0/get-started>
-  Gradle: <https://gradle.org/install>
-  Grafana: <https://grafana.com/docs/grafana/latest/setup-grafana/installation>
-  InfluxDB: <https://docs.influxdata.com/influxdb/v2/install>
-  Java: <https://www.java.com/fr/download/manual.jsp>
-  PostgreSQL: <https://www.postgresql.org/download>
-  Spring Boot: <https://spring.io/projects/spring-boot>
-  ViteJs: <https://vitejs.dev>
-  VueJs: <https://v2.fr.vuejs.org/v2/guide/installation.html>

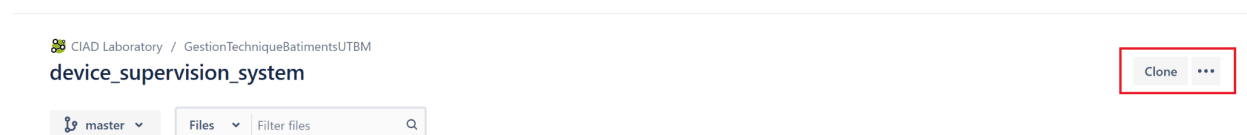
2.2. Récupération du code source

Le code source du projet est hébergé sur la plateforme de gestion de code BitBucket (basé sur Git) du Laboratoire CIAD et s'intitule [device_supervision_system](https://bitbucket.org/ciadlabfr/device_supervision_system).

Ouvrir un terminal et cloner le projet en local :

git clone https://<user>@bitbucket.org/ciadlabfr/device_supervision_system.git

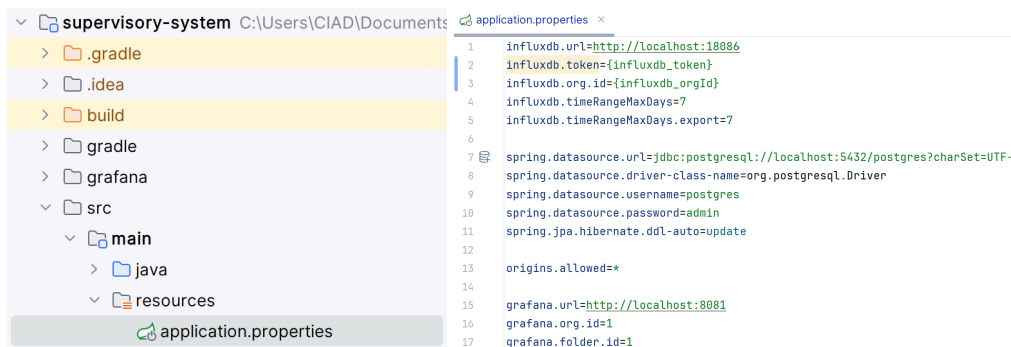
⚠ Remplacer **<user>** par votre username sur bitbucket ou copier directement la commande dans l'interface bitbucket du projet en cliquant sur le bouton  (voir figure suivante).



2.3. Configuration locale

2.3.1. Configuration API Spring

Le fichier de configuration Backend de l'API Spring Boot se trouve dans **/src/main/ressources/application.properties** :

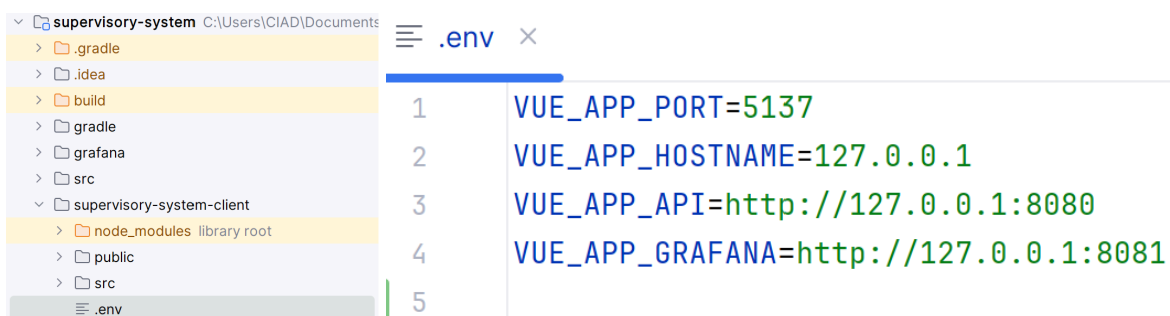


Il permet de configurer l'accès à la base de données **InfluxDB** (**influxdb.**), l'accès à la base de données **PostgreSQL** (**spring.datasource.**) et l'accès à **Grafana** (**grafana.**).

⚠ L'accès à la base de données InfluxDB n'est autorisé qu'aux adresses MAC enregistrées. Dans le cadre du stage, la connexion à la base se fait via un raspberry autorisé avec une connexion SSH (via Putty). Pour plus d'informations, se rapprocher des personnes compétentes.

2.3.2. Configuration Client VueJs

Le fichier de configuration Frontend du Client VueJs se trouve dans **/supervisory-system-client/.env** :



Il permet de configurer l'accès à l'API Spring (**VUE_APP_API**).

2.4. Configuration docker

L'ensemble de l'application est conteneurisé via Docker et comporte plusieurs instances dans **docker-compose.yml** :

```
postgres:
  container_name: postgres_db
  image: postgres:latest
  #ports:
  # - "5432:5432"
  environment:
    POSTGRES_DB: postgres
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
  volumes:
    - postgres_data:/var/lib/postgresql/data
  networks:
    - supervisory-system-network
```

Le conteneur **postgres_db** utilise la dernière image postgres. Il sauvegarde ses données dans le volume **postgres_data** et se trouve sur le réseau local **supervisory-system-network**.

```
spring:
  container_name: spring_api
  depends_on:
    - postgres
  build:
    context: .
    dockerfile: Dockerfile
  ports:
    - "3000:8080"
  environment:
    - SPRING_PROFILES_ACTIVE=production
    - SPRING_DATASOURCE_URL=jdbc:postgresql://postgres:5432/postgres?charset=UTF-8
    - SPRING_DATASOURCE_USERNAME=postgres
    - SPRING_DATASOURCE_PASSWORD=postgres
    - INFLOUXDB_URL=http://host.docker.internal:18086
    - GRAFANA_URL=http://grafana:3000
    - ALLOWED_ORIGINS=*
  networks:
    - supervisory-system-network
```

Le conteneur **spring_api** utilise le dockerfile présent à la racine du projet. Il dépend du conteneur **postgres** et se trouve sur le réseau local **supervisory-system-network**.

```
vuejs:
  container_name: vue_client
  depends_on:
    - spring
  build:
    context: ./supervisory-system-client
    dockerfile: Dockerfile
  ports:
    - "8080:8080"
  environment:
    - VUE_APP_PORT=8080
    - VUE_APP_HOSTNAME=vuejs
    - VUE_APP_API=http://localhost:3000 # web client
    - VUE_APP_GRAFANA=http://localhost:8081 # web client
  networks:
    - supervisory-system-network
```

Le conteneur **vue_client** utilise le dockerfile présent à la racine du client dans **supervisory-system-client**. Il dépend du conteneur **spring** et se trouve sur le réseau local **supervisory-system-network**.

```
grafana:
  container_name: grafana
  image: grafana/grafana-oss
  restart: unless-stopped
  ports:
    - "8081:3000"
  volumes:
    - grafana_data:/var/lib/grafana
    - ./grafana/config/grafana.ini:/etc/grafana/grafana.ini
    - ./grafana/provisioning:/etc/grafana/provisioning
  networks:
    - supervisory-system-network
```

Le conteneur **grafana** utilise l'image open-source grafana. Il se trouve sur le réseau local **supervisory-system-network** et sauvegarde sa configuration dans **/grafana/config**.

3. Structure du projet

3.1. Conception générale

Le module de supervision est composé de 5 grandes instances:

- **InfluxDB**
La base de données qui enregistre les mesures des appareils.
- **PostgreSQL**
La base de données qui enregistre les informations des appareils.
- **Spring Boot**
L'API qui s'occupe de vérifier les autorisations (vérification des tokens) et de répondre aux requêtes via des contrôleurs qui utilisent des services qui interrogent les bases de données.
- **VueJs**
Le client s'occupe d'interroger l'API via des requêtes Axios et d'afficher les résultats (appareils et mesures).
- **Grafana**
La plateforme qui permet de créer des dashboards qui s'intègrent dans VueJs.

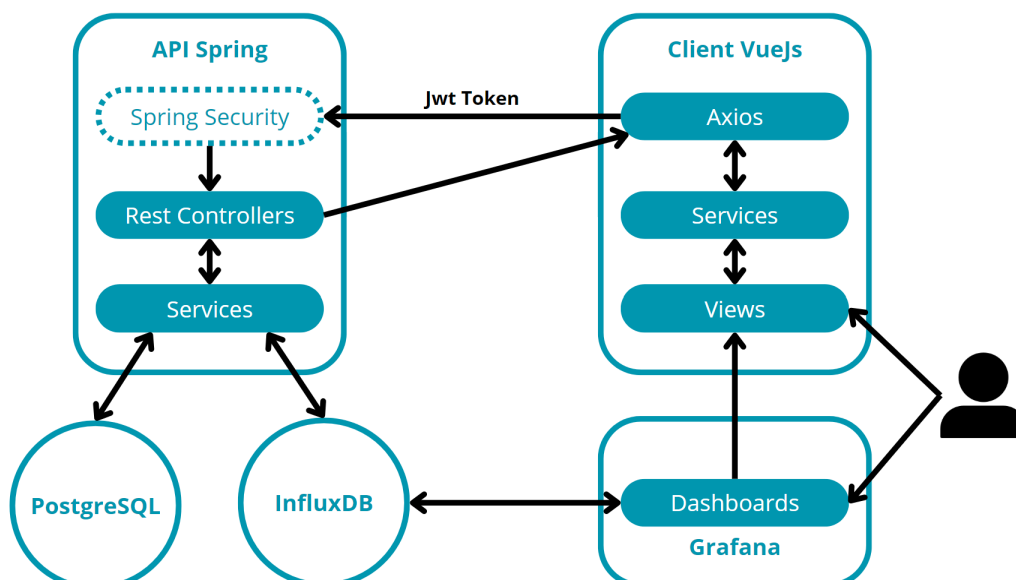


Figure 2: Schéma de l'architecture globale du module de supervision

3.2. Architecture Backend

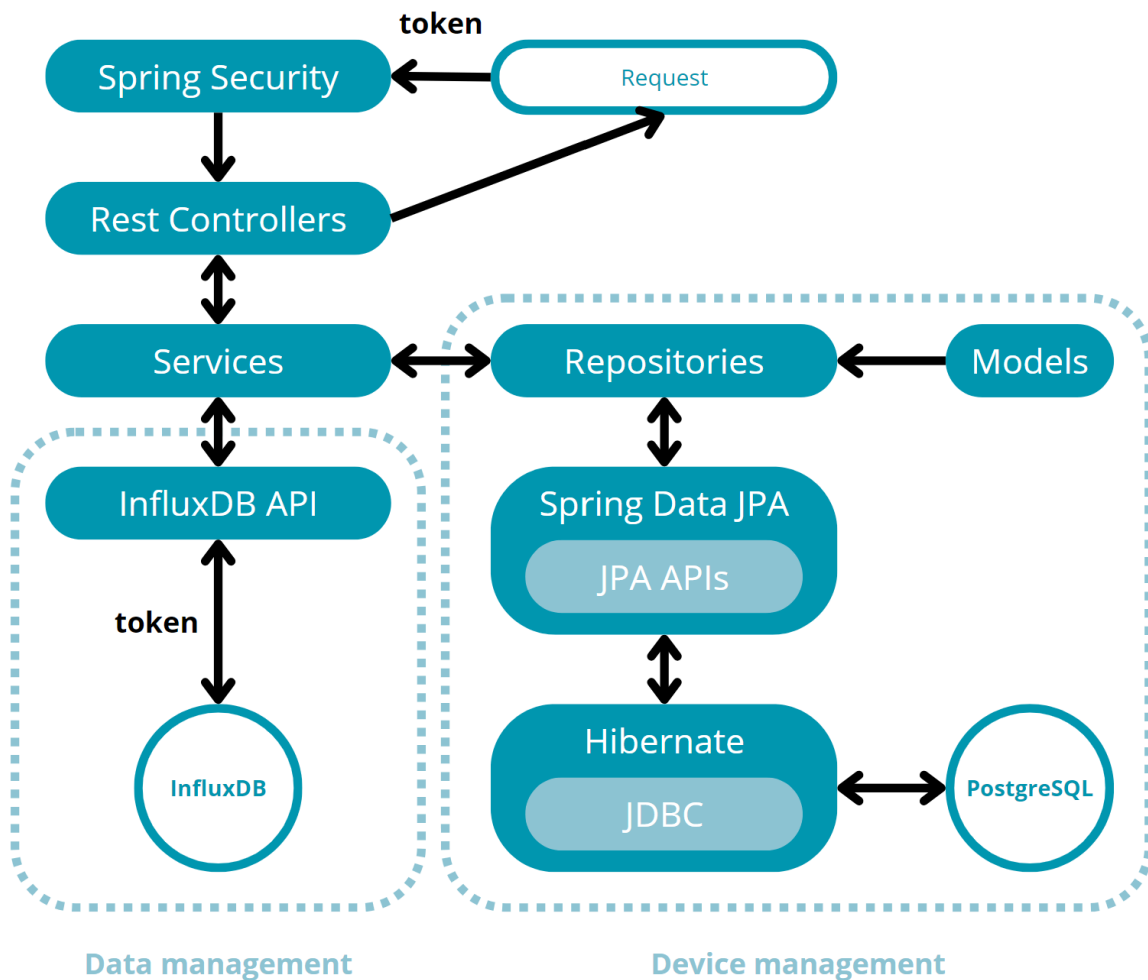


Figure 3: Schéma de l'architecture backend du module de supervision

La gestion des requêtes par l'API Spring se déroule en plusieurs étapes:

1) Interception de la requête par Spring Security

Lorsqu'une requête est adressée à l'API, Spring Security l'intercepte afin de vérifier si elle comporte un token d'authentification et si la destination est autorisée en fonction du rôle attribué au client de la requête (Principal).

2) Redirection de la requête vers un service grâce aux contrôleurs

Les contrôleurs agissent en tant qu'endpoint de l'API, ils permettent de rediriger une requête vers un service et, une fois la requête traitée par le service, renvoient la réponse à l'expéditeur de la requête.

3) **Gestion de la requête par le service**

Les services permettent, en fonction de leur nature, de récupérer certaines données des bases de données. Il y a 2 types de données:

a) Requête liée aux appareils

Si la requête utilise un service nécessitant la récupération de données d'un appareil dans la base de données PostgreSQL, celui-ci va utiliser un repository lié à Spring Data JPA pour associer un objet Java à un modèle et requêter la base de données via hibernate.

b) Requête liée aux mesures des appareils

Si la requête utilise un service nécessitant la récupération de mesures d'un appareil dans la base de données InfluxDB, celui-ci utilisera l'API Influx avec un token d'authentification pour directement envoyer une requête en langage flux à la base de données.

4) **Renvoi de la réponse par le contrôleur**

Lorsque le contrôleur reçoit une réponse du service, il renvoie cette réponse au client.

3.3. Architecture Frontend

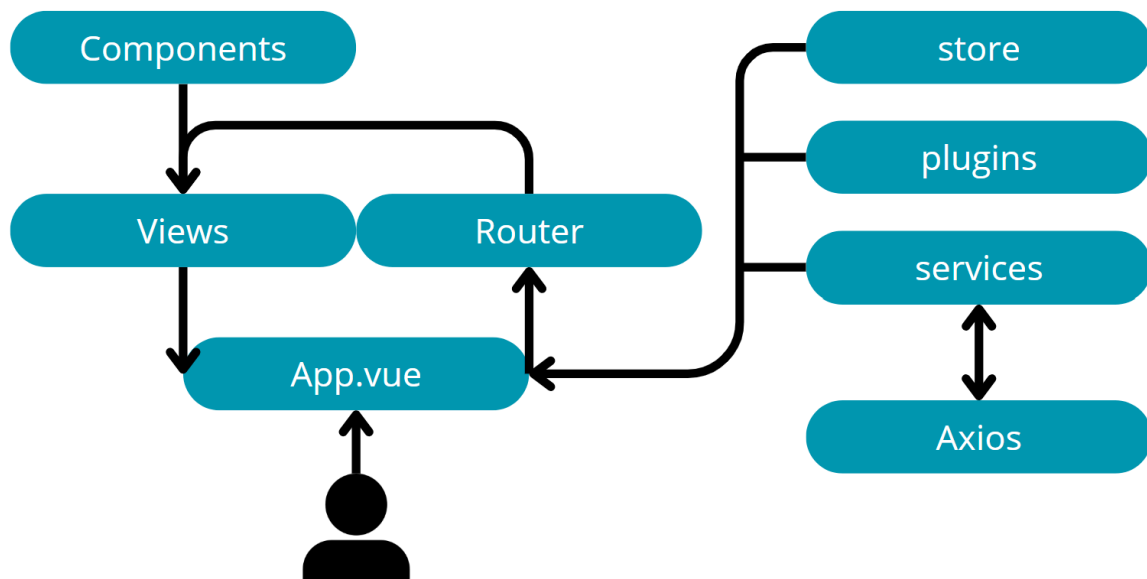


Figure 4: Schéma de l'architecture frontend du module de supervision

Le frontend du module de supervision est basé sur le moteur de rendu VueJs et est développé avec ViteJs. Il suit l'architecture One-page classique de Vue avec le fichier `App.vue` racine des rendus. Le routeur ([vue-router](#)) permet de définir les routes de l'application et de modifier en conséquence les vues générées dans `App.vue`.

VueJs utilise également un système de store ([vuex](#)) pour sauvegarder en local des informations tel que l'utilisateur, et différents plugins comme Vuetify qui permet de générer des composants graphiques plus facilement.

Enfin l'application doit pouvoir interagir avec l'API grâce aux services qui utilisent une instance personnalisée d'[Axios](#) pour faire des requêtes.

⚠️ Axios utilise le jwt token stocker dans [vue-cookies](#) lors de l'authentification pour requêter l'API (envoi via un header).

3.4. Modèle conceptuel de données

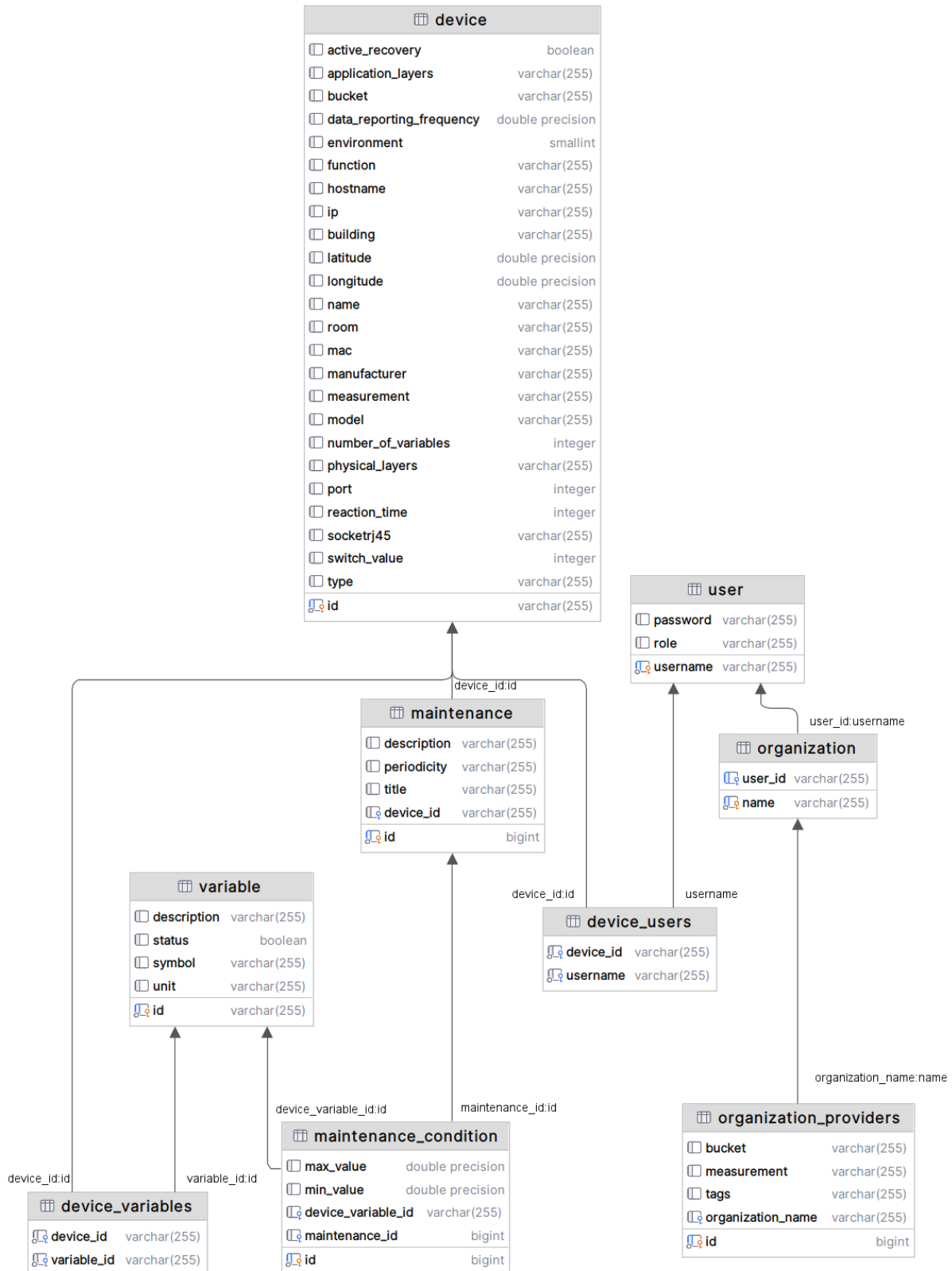


Figure 5: Modèle conceptuel de données (MCD)

4. Fonctionnalités implémentées

4.1. Backend

4.1.1. Gestion des appareils

La gestion des appareils implique l'implémentation de différentes fonctionnalités tel que:

- La visualisation des ou d'un appareil
 - **GET** /api/devices
 - **GET** /api/devices/{deviceId}
- L'ajout d'un nouvel appareil
 - **POST** /api/devices/add
 - deviceId (*body*)
 - measurement (*body*)
- La modification d'un appareil
 - **POST** /api/devices/{deviceId}/update/{fieldType}
- La suppression d'un appareil
 - **GET** /api/devices/{deviceId}/delete

Une gestion des variables à également été implémentée afin de lier les mesures d'un appareil à une description qui n'est pas présente dans InfluxDB :

- Visualisation des variables
 - **GET** /api/devices/variables
- Ajout ou modification de variable
 - **POST** /api/devices/variables/add
 - deviceVariableId (*body*)
 - description (*body*)
 - unit (*body*)
 - symbol (*body*)
 - status (*body*)
- Suppression de variable
 - **GET** /api/devices/variables/{deviceVariableId}/delete

- Attribution d'une variable à un Device
 - **GET** /api/devices/{deviceId}/addVariable/{deviceVariableId}

4.1.2. Gestion de la maintenance

La gestion des maintenances implique l'implémentation de différentes fonctionnalités tel que:

- La visualisation des maintenance et des maintenances propres à un appareil
 - **GET** /api/maintenances
 - **GET** /api/maintenances/{deviceId}
- L'ajout d'une nouvelle maintenance
 - **POST** /api/maintenances/{device_id}/add
 - title (*body*)
 - description (*body*)
 - periodicity (*body*)
- La suppression d'une maintenance
 - **GET** /api/maintenances/{deviceId}/delete/{maintenanceId}

Chaque maintenance peut également définir des plages de fonctionnement anormal en choisissant une valeur maximale ou minimale pour chaque mesure de l'appareil

- Ajout d'une condition pour une maintenance
 - **POST** /api/maintenances/{maintenanceId}/conditions/add
 - variableId (*body*)
 - maxValue (*body*)
 - minValue (*body*)
- Suppression d'une condition
 - **GET** /api/maintenances/{maintenanceId}/conditions/{conditionId}/delete

4.1.3. Gestion des données

L'accès aux mesures (données) des appareils se fait depuis le contrôleur **getDeviceData** :

- **GET** /api/devices/{deviceId}/data
 - timeRangeStart (*query*)
 - timeRangeStop (*query*)

Le contrôleur va vérifier si la période de temps entre timeRangeStart et timeRangeStop est valide:

- Période non négative (timeRangeStart > timeRangeStop)
 - sinon inversement de timeRangeStart et timeRangeStop
- Période non supérieur à timeRangeMaxDays
 - sinon timeRangeStart + timeRangeMaxDays

Puis, en fonction de la durée de la période, il va automatiquement définir une période d'agrégation des données dans la requête InfluxDB:

```
String windowPeriod = "1w";           // range > 1mo
String unit = "week";
if (differenceInHours <= 1) {          // range <= 1h
    windowPeriod = "10s";
    unit = "second";
} else if (differenceInHours < 24) {   // 1h < range < 1j
    windowPeriod = "1m";
    unit = "minute";
} else if (differenceInHours <= 24 * 7) { // 1j <= range <= 1w
    windowPeriod = "1h";
    unit = "hour";
} else if (differenceInHours < 24 * 31) { // 1w < range < 1mo
    windowPeriod = "1d";
    unit = "day";
}
```

4.1.4. Gestion de l'export de données

L'export de données permet de pouvoir exporter des données dans des fichiers CSV de manière asynchrone en tâche de fond, pour plusieurs Device et sans restriction de période:

- **POST** /api/influxdb/export
 - timeRangeStart (*query*)
 - timeRangeStop (*query*)

- windowPeriod (*query*)
- email (*body*)
- devicesId (*body*)

Une fois l'export terminé, un fichier .zip contenant les CSV créés est présent dans le dossier exports de l'API et contient un nom unique (son ID). Une fonction permet au client de récupérer ce fichier avec son ID :

- **GET** /api/influxdb/export/download/{id}

L'identifiant du fichier est envoyé par email lors de la fin du processus.

4.1.5. Gestion des dashboards Grafana

La gestion des dashboards prend en considération que tous les dashboards sont dans le même dossier de la même organisation et permet de récupérer l'ensemble des dashboards via l'API Grafana:

- **GET** /api/grafana/dashboards

Une fois les dashboards récupérés avec leur propre uid, les données d'un dashboard peuvent être récupérées avec son uid:

- **GET** /api/influxdb/dashboards/{uid}

4.1.6. Gestion des rôles

La gestion des rôles est entièrement implémentée avec Spring Security et permet d'une part d'authentifier un utilisateur avec un jwtToken et d'autre part de restreindre l'accès à certains services.

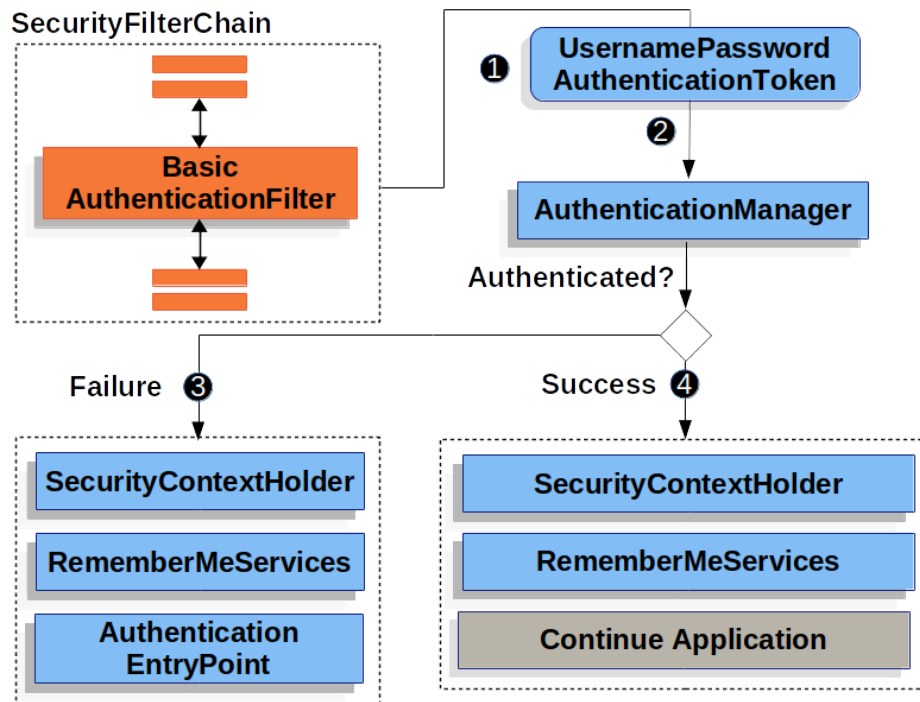


Figure 6: Schéma d'authentification Spring Security

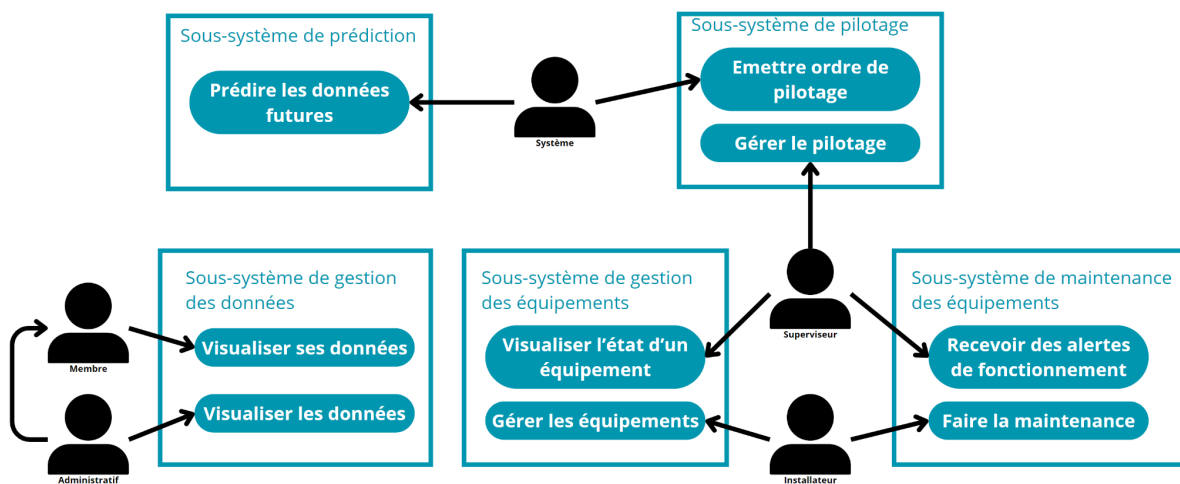


Figure 7: Diagramme UML des rôles du module de supervision

4.1.7. Gestion des organisations

La gestion des organisations implique l'implémentation de différentes fonctionnalités tel que:

- La visualisation des ou d'une organisation

- **GET** /api/orgs
- **GET** /api/orgs/{name}
- L'ajout d'une nouvelle organisation
 - **POST** /api/orgs/add
 - name (*body*)
- La suppression d'une organisation
 - **GET** /api/orgs/{name}/delete

Chaque organisation doit être affectée à une ou des données mesurées qui généralement correspondent à un capteur qui appartient à cette organisation. Pour cela, des Providers doivent être liés à une organisation, c'est-à-dire un lien dans la base de données influxDB avec le nom du bucket, du measurement et du/des tags.

D'après le schéma ci-dessous, si nous voulons obtenir les données de l'**Org I** dans le **Measurement 1**, ces informations seules ne sont pas suffisantes car l'**Org II** stocke également des données dans ce measurement. Ainsi, pour chaque donnée à été attribué un tag *device_id* qui correspond à l'id du device lui-même et permet de filtrer les données dans le measurement. (dans notre exemple : measurement=measurement 1 & device_id=Device C)

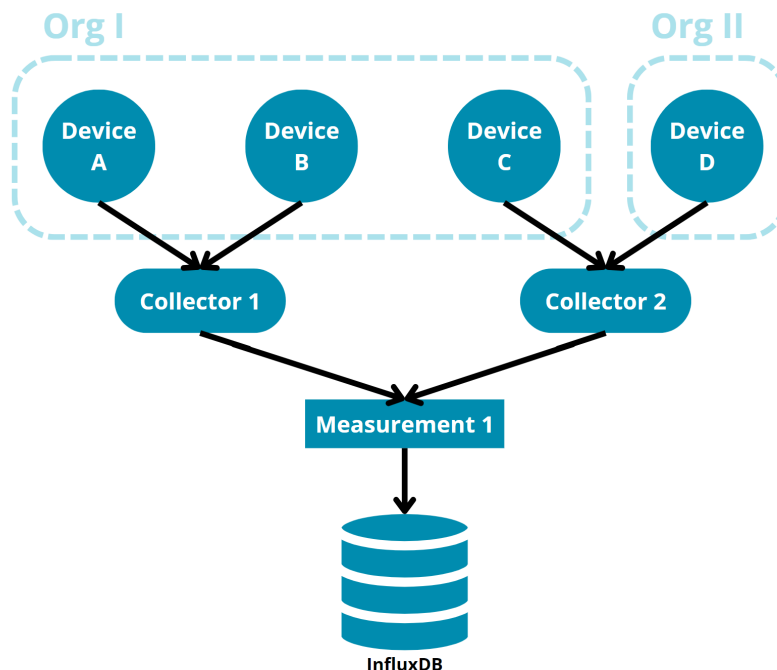


Figure 8: Schéma de récupération des données influxDB

- Ajout d'un provider
 - **POST** /api/orgs/{name}/providers/add
 - bucket (*body*)
 - measurement (*body*)
 - tags (*body*)
- Modification des tags d'un provider
 - **POST** /api/orgs/{name}/providers/setTags
 - bucket (*body*)
 - measurement (*body*)
 - tags (*body*) -> *ex: device_id=Device C*
- Suppression d'un provider
 - **POST** /api/orgs/{name}/providers/delete
 - bucket (*body*)
 - measurement (*body*)

Une fois les providers rattachés à l'organisation, les mesures de l'organisation peuvent être obtenues en spécifiant le provider (timeRange limitée à la dernière heure).

- **POST** /api/orgs/{name}/data
 - bucket (*body*)
 - measurement (*body*)

4.2. Frontend

4.2.1. Axios

Chaque service interrogeant l'API utilise une instance unique d'Axios personnalisée dans **src/services/axios.service.js**.

Cette instance ajoute des headers avant chaque requête et notamment le header Bearer Authorization comportant le jwtToken nécessaire lors de l'authentification dans l'API.

Elle permet également d'analyser chaque erreur liée à une requête ou une réponse et de renvoyer le message. Dans le cas d'une réponse avec un status 401 (accès non autorisé), Axios déconnecte l'utilisateur :

```
if(err.response) {
  error = {title: err.message, message: err.response.data}
  if(err.response.status === 401) {
    error = {title: 'Unauthorized Access', message: 'User does not have required permissions'}
    store.commit( type: 'clearUser')
  }
}
```

4.2.2. Routeur

Le routeur permet d'associer pour chaque endpoint une vue mais agit également comme première barrière d'authentification en interdisant l'accès à l'application si aucun jwtToken n'est présent dans les cookies du client :

```
router.beforeEach( guard: (to : RouteLocationNormalized , from : RouteLocationNormalized , next : NavigationGuardNext ) : void => {
  const token = VueCookies.get('jwtToken');
  if (!token && to.name !== 'Authentication') {
    store.commit( type: 'clearUser')
    next({ name: 'Authentication' });
  }
  else if (token && to.name === 'Authentication') {
    next({ name: 'Home' });
  }
  else if (to.meta.roles && !store.state.user.roles.some(role : T => to.meta.roles.includes(role))) {
    next({ name: 'Home' })
  }
  else {
    next();
  }
});
```

4.2.3. Dashboards

Les dashboards sont générés dans les vues avec un iframe qui utilise les urls fournis dans les données renvoyées par l'API :

```
<iframe
  v-if="panels.length > 0"
  v-for="(panel, index) in panels"
  :key="index"
  :src="'${grafana_uri}/d-solo/${selectedDashboardUid}?orgId=1&panelId=${panel.id}&theme=${currentTheme.dark ? 'dark' : 'light'}'"
  :style="getPanelStyle[index]"
></iframe>
```

La position de chaque panel d'un dashboard est calculée directement avec la fonction `getPanelStyle` qui utilise les données de `gridPos` renvoyées par l'API Grafana :

```
getPanelStyle() {  
  return this.panels.map((panel) => {  
    return this.windowWidth > 1000  
      ? `  
        position: absolute;  
        width: ${panel.gridPos.w * 100 / 24}%;  
        height: ${panel.gridPos.h * 100 / 21}%;  
        left: ${panel.gridPos.x * 100 / 24}%;  
        top: ${panel.gridPos.y * 100 / 21}%;  
      `  
      : `  
        position: block;  
        width: 100%;  
        height: 30%;  
      `;  
  });  
},
```

⚠ Grafana considère que `max-width = 24` et `max-height = 21`

Table des figures

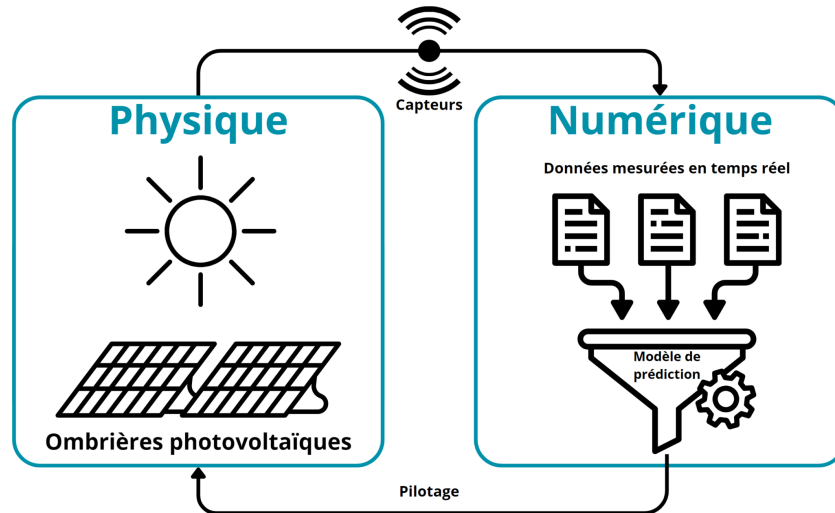
Figure 1: Schéma du fonctionnement du système de management de l'énergie.....	5
Figure 2: Schéma de l'architecture globale du module de supervision.....	9
Figure 3: Schéma de l'architecture backend du module de supervision.....	10
Figure 4: Schéma de l'architecture frontend du module de supervision.....	12
Figure 5: Modèle conceptuel de données (MCD).....	13
Figure 6: Schéma d'authentification Spring Security.....	18
Figure 7: Diagramme UML des rôles du module de supervision.....	18
Figure 8: Schéma de récupération des données influxDB.....	19

Table des annexes

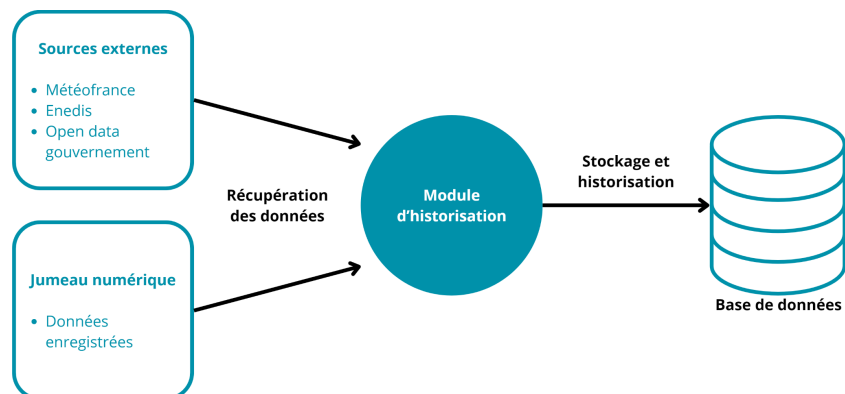
Annexe I : Module du Jumeau numérique.....	25
Annexe II : Module d'historisation.....	25
Annexe III : Module de prédiction.....	25
Annexe IV : Module de pilotage.....	26
Annexe V : Module de supervision.....	26

Annexes

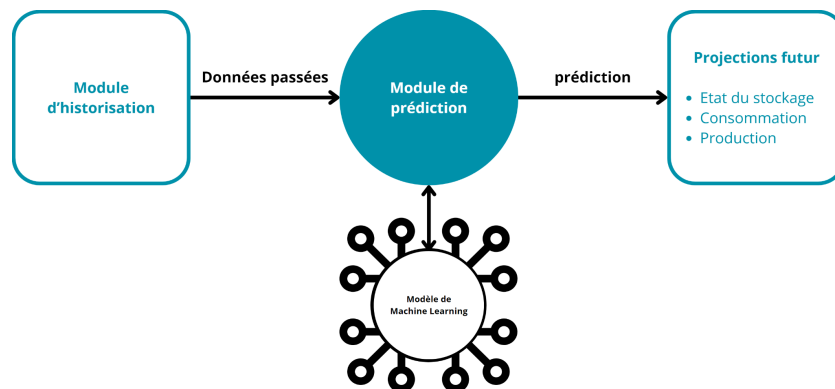
Annexe I : Module du Jumeau numérique



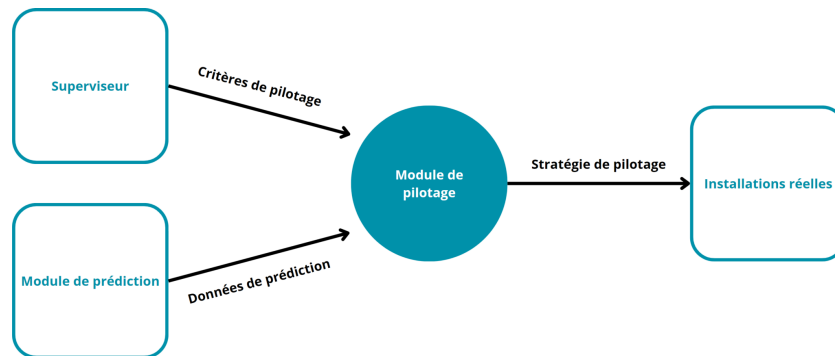
Annexe II : Module d'historisation



Annexe III : Module de prédiction



Annexe IV : Module de pilotage



Annexe V : Module de supervision

