

Passerelle industrielle 4.0



Lycée Lafayette
21 Bd Robert Schuman, 63000 Clermont-Ferrand

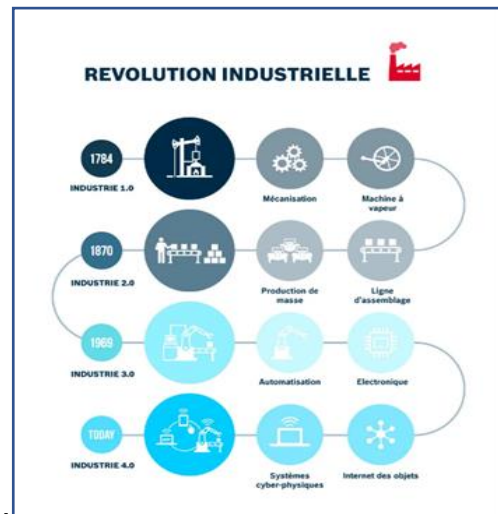
BTS Systèmes Numériques Option A Informatique et réseaux
2023/2024

MOHAMED Amine-Dada

Table des matières

Présentation du système.....	3
Description du système.....	4
Cahier des charges / Application utilisées.....	4
Matériel utilisé.....	5
Coût total du projet :.....	7
Répartition des tâches et rôles du groupe.....	7
Objectifs du projet.....	9
UML.....	9
Diagramme des cas d'utilisation.....	9
Diagrammes de déploiement.....	10
Diagramme de séquence :.....	11
Organisation du groupe.....	12
Planning prévisionnel.....	12
Planning prévisionnel de Clément.....	13
Planning prévisionnel d'Amine :.....	13
Eleve3 : Amine-Dada.....	14
Contextualisation :.....	14
Objectifs du projet :.....	15
Documentations :.....	15
C'est quoi une API REST ?.....	15
Qu'est-ce qu'une API ?.....	15
REST ?.....	16
Le fonctionnement et UML :.....	17
1. Création de l'API REST pour la récupération des données et la commande :.....	17
2. Création d'une application de démonstration utilisant l'API :.....	18
3. Voici un schéma simple qui explique mes tâches :.....	19
4. Voyons ça avec le diagramme de séquence :.....	19
5. Voici le diagramme de déploiement.....	20
Logiciels utilisés :.....	21
Création de l'API :.....	23
Créations de l'application :.....	39
Conclusion :.....	52

Présentation du système



Grâce aux percées dans les domaines des technologies de l'information, des communications mobiles et de la robotique, les technologies numériques sont de plus en plus utilisées dans les entreprises du monde entier.

Il s'agit d'une véritable révolution industrielle, la 4e, après celle de la mécanisation, celle de la production de masse et celle de l'automatisation. Avec l'industrie 4.0 ou « usine connectée » l'industrie devient, grâce à l'arrivée de la numérisation, un système global interconnecté dans lequel les machines, les systèmes et les produits communiquent en permanence.

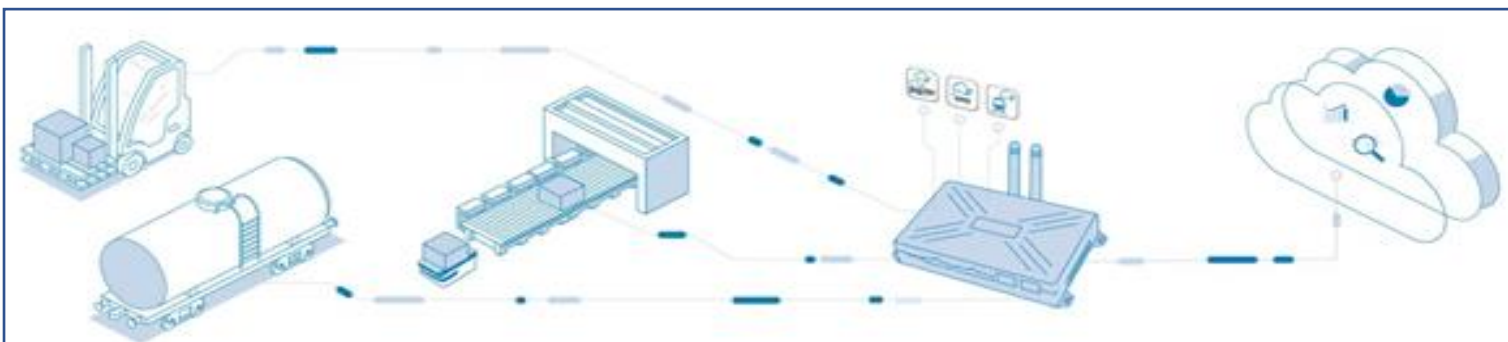
L'industrie 4.0 consiste donc à surveiller et à contrôler en temps réel les machines et les équipements en installant des capteurs à chaque étape du processus de production.

Afin de surveiller et piloter l'usine connectée, une solution consiste à rendre possible l'accès à ces données sur le réseau informatique de l'entreprise.

Il faut cependant tenir compte de la cybersécurité et donc ajouter une couche de protection.

La passerelle industrielle 4.0 met en réseau les capteurs, les machines et les plateformes IoT pour la collecte et le prétraitement de données de capteur et de processus locaux.

Elle interconnecte l'usine connectée au réseau de l'entreprise.



Description du système

Cahier des charges / Application utilisées

La passerelle est interconnectée a deux réseaux, un côté machines industrielles et l'autre côté réseaux de l'entreprise.

Un firewall sera mis en place.

Les capteurs et actionneurs pourront utiliser différents protocoles (Ethernet TCP, ModBus TCP, WIFI etc.).

La programmation des automatisations et la visualisation de données utilisera Node Red.

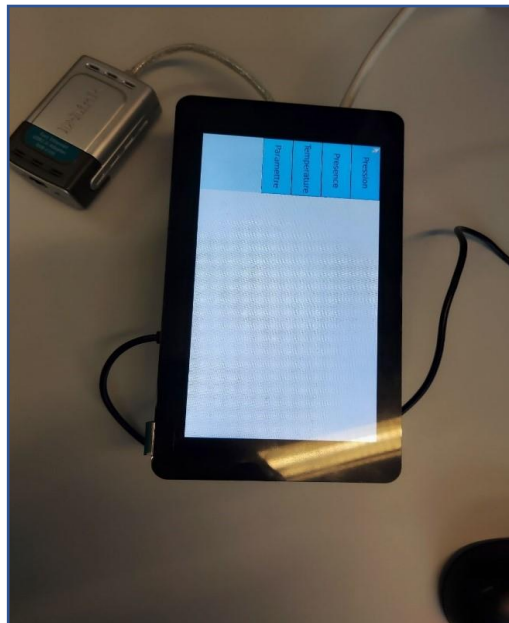
Une sauvegarde dans une base de données permettra une exploration historique des données sur 30 jours.

Un serveur web intégré permettra l'observation des données.

Le paramétrage et la visualisation pourront se faire sur un écran tactile intégré.

Une API REST sera implantée pour la gestion de la passerelle.

Pour mettre en place ce système nous utiliserons les logiciels phpMyAdmin pour la création et la gestion de base de données, Apache pour le côté serveur, QT afin de créer l'API REST, Visual Studio Code pour la création du site web et Node Red afin de gérer et piloter les capteurs.



Matériel utilisé

Le système sera composé d'une carte Raspberry PI 4 avec une extension USB-RJ45, un écran et un OS adapté.

Pour les mesures nous utiliserons des capteurs IFM et un maître IO-Link.



- Capteur de vibration VVB001 :

<https://www.ifm.com/fr/fr/product/VVB001>

Il nous servira à mesurer la vitesse (en mm/s), l'accélération (en m/s^2), Crest et la température en $^{\circ}\text{C}$.



- Compteur d'air comprimé SD5500 :

<https://www.ifm.com/fr/fr/product/SD5500>

Ce capteur permet de mesurer la pression (en bar), le débit (en m^3) et la température (en $^{\circ}\text{C}$).

- Maître IO-Link avec interface PROFINET AL1302 :

<https://www.ifm.com/fr/fr/product/AL1302?tab=documents>

Le maître IO-Link, ou IO-Link Master, sert à connecter des terrain IO-Link comme des capteurs et des actionneurs.



Pourquoi utiliser IO-Link ?

La technologie IO-Link possède de nombreux avantages qui nous paraissent importants à exploiter.

Pour commencer, qu'est ce que IO-Link ?

IO-Link est une interface de communication point à point permettant aux capteurs et actionneurs de dialoguer avec les systèmes de commandes. Apparue à la fin des années 2000, il s'agit de la première technologie d'entrées/sorties normalisée, répondant à la norme IEC 61131-9. Un système IO-Link se compose en général d'un moins un module "Maître" raccordé à des périphériques p e. Les appareils IO-Link peuvent par exemple être raccordés au moyen d'un simple connecteur M12.

Et voici donc notre premier avantage : les dysfonctionnements dus à des câblages incorrects sont donc exclus.

De plus, ce dispositif permet de réduire les coûts machines. Grâce à IO-Link l'entreprise peut réaliser des économies sur la maintenance, mais aussi sur les coûts d'ingénierie et de documentation. Il favorise aussi une réduction des stocks grâce à des appareils multi-usages intelligents

IO-Link va aussi numériser les communications des capteurs et actionneurs. IO-Link permet de passer facilement de l'analogique au numérique avec une connectique standard. Les équipements sont ainsi raccordés via un simple câble non blindé, ce qui facilite et réduit les coûts d'installation.

IO-Link permet aussi de gagner du temps en pilotant à distance les équipements. Les réglages des périphériques peuvent être modifiés à distance, sans intervenir physiquement sur site et sur l'ensemble des équipements simultanément.

Pour finir, cet outil permet de gagner en productivité et fiabilité. Grâce aux données transmises des diagnostics sont effectués. IO-Link permet de réduire les risques de pannes, de planifier plus facilement les opérations de maintenance et ainsi de réduire les temps d'arrêts inopinés. Un port IIOT (Internet Industriel des Objets ou en anglais "Industrial Internet of Things") est d'ailleurs dédié aux diagnostics.

Coût total du projet :

Matériel	Cout
Raspberry PI4	63€
Écran de 7 pouce	50€
Adaptateur USB-Ethernet	15€

Le projet a donc un coût total de 128€ dans ce calcul nous ne comptons pas les capteurs que l'on nous a fournis. Les capteurs seront présents sur site.

Répartition des tâches et rôles du groupe

Différentes fonctionnalités ont été attribuées à différents étudiants en suivant le cahier des charges.

1. Elève n°1 (Clement)

Fonctionnalités en charge :

- Installation et paramétrage de la carte Raspberry.
Il faudrait donc installer un système d'exploitation compatible avec la carte, une extension pour avoir deux cartes réseaux ainsi qu'un écran tactile.

On utilisera un écran 7 pouces.

- Installation et paramétrage du Firewall Installation et paramétrage du Broker MQTT Mosquitto.
- Installation et paramétrage de Node Red.
- Création de l'application de gestion sur écran tactile.
- Création du boîtier pour la carte ainsi que l'écran.

2. Elève n°2 (Christelle)

Fonctionnalités en charge :

- Installation et paramétrage de la base de données.
Une base de données sera créée afin de stocker les données reçues par les capteurs.
- Sauvegarde des données sur 30 jours.
- Affichage des données issues des capteurs.
Les données recueillies devront être affichées sur un site web accessible.
- Pilotage des actionneurs.
Il faudra pouvoir modifier les réglages des capteurs à distance grâce au site web.

3. Elève n°3 (Amine)

Fonctionnalités en charge :

- Création de l'API REST pour la récupération des données et la commande :
 - Le but est de créer une API en PHP avec l'architecture REST pour pouvoir récupérer les données des capteurs et les afficher sous forme de JSON via le protocole HTTP.
- Création d'une application de démonstration utilisant l'API :
 - Il va falloir créer une interface et un programme sous QT Creator pour pouvoir lire les données récupérées sous format JSON généré par l'API.

Objectifs du projet

Ce projet a pour objectif d'apporter aux techniciens une vue simplifiée de l'automate utilisé.

La passerelle industrielle 4.0 permet de mesurer différentes données grâce à des capteurs afin de faciliter la visualisation et l'utilisation des données.

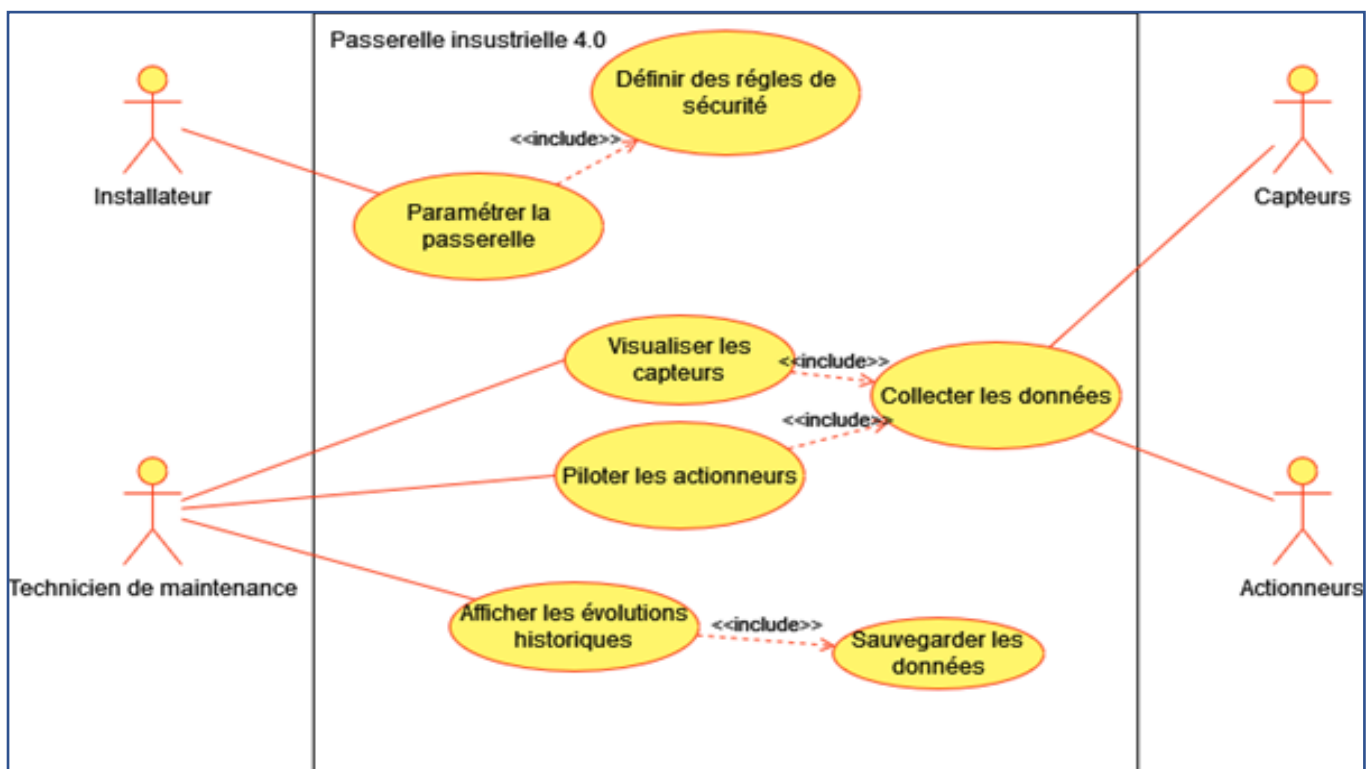
Le site web offre une vision simplifiée de toutes ces données sous forme de liste, de widget et de graphiques ce qui permet une lecture et une analyse plus rapide.

De plus, le site permet le paramétrage de certaines fonctions en temps réel et à n'importe quel endroit ce qui est beaucoup plus simple et qui permet de gagner énormément de temps.

N'importe qui peut s'essayer à paramétrer sans avoir de connaissances spécifiques en informatique grâce aux widgets intuitifs.

UML

Diagramme des cas d'utilisation



L'installateur a plusieurs responsabilités. Tout d'abord, il peut définir des règles de sécurité pour le système, garantissant ainsi un fonctionnement sûr et conforme aux normes. Ensuite, il paramètre la passerelle, ce qui implique la configuration des paramètres spécifiques de l'appareil. L'installateur peut également visualiser les capteurs, ce qui lui permet de surveiller les données collectées par ces derniers. De plus, il est en charge de collecter les données des capteurs, ce qui est essentiel pour le bon fonctionnement du système. Enfin, l'installateur a la possibilité de sauvegarder les données, assurant ainsi leur préservation et leur disponibilité ultérieure.

Le technicien de maintenance, quant à lui, a un rôle crucial dans le maintien en bon état du système. Il peut également visualiser les capteurs, ce qui lui permet de surveiller l'état du système et de détecter d'éventuels problèmes. Le technicien est chargé de collecter les données pour la maintenance, ce qui lui permet d'analyser les performances et de résoudre les problèmes éventuels. Enfin, il peut afficher les évolutions historiques, ce qui lui donne un aperçu des tendances et des changements au fil du temps.

Les capteurs ont un rôle fondamental : ils collectent les données et les envoient au système. Leur précision et leur fiabilité sont essentielles pour garantir le bon fonctionnement global du système.

Enfin, les actionneurs sont contrôlés par le système. Ils permettent d'effectuer des actions physiques, comme l'arrêt ou la mise en marche d'une machine, en réponse aux données collectées par les capteurs.

Diagrammes de déploiement

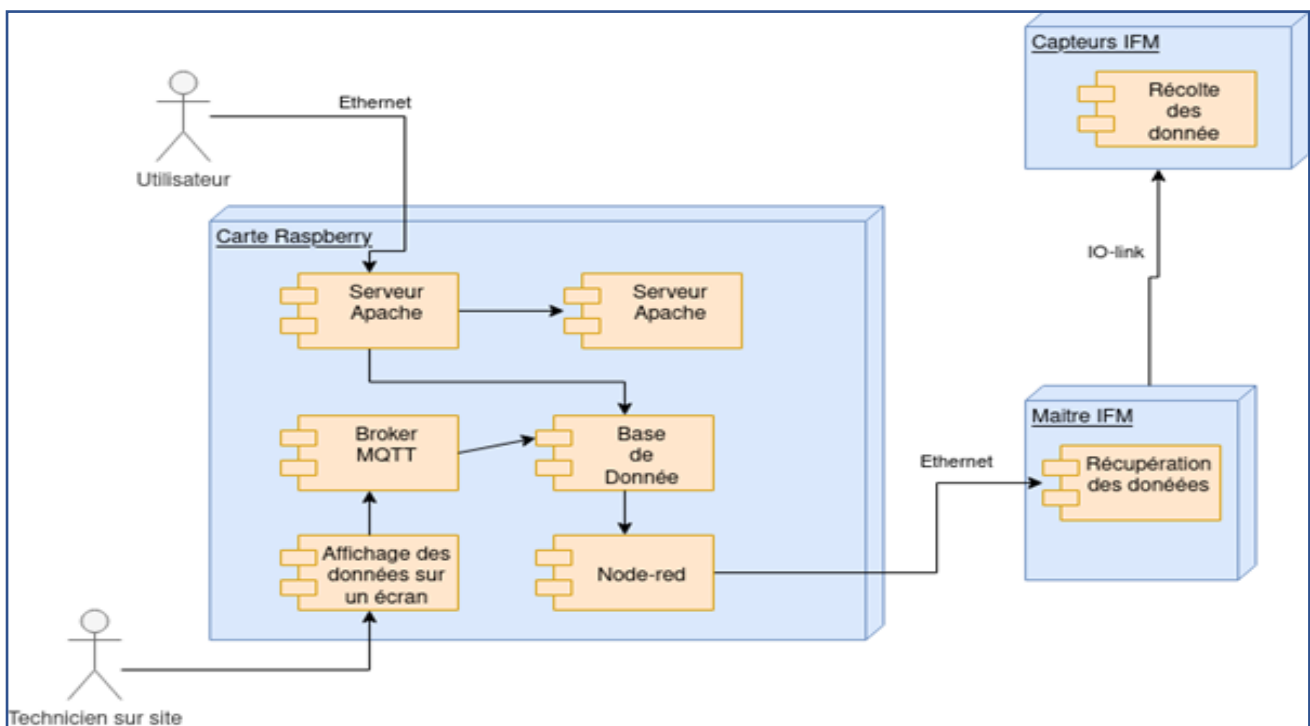
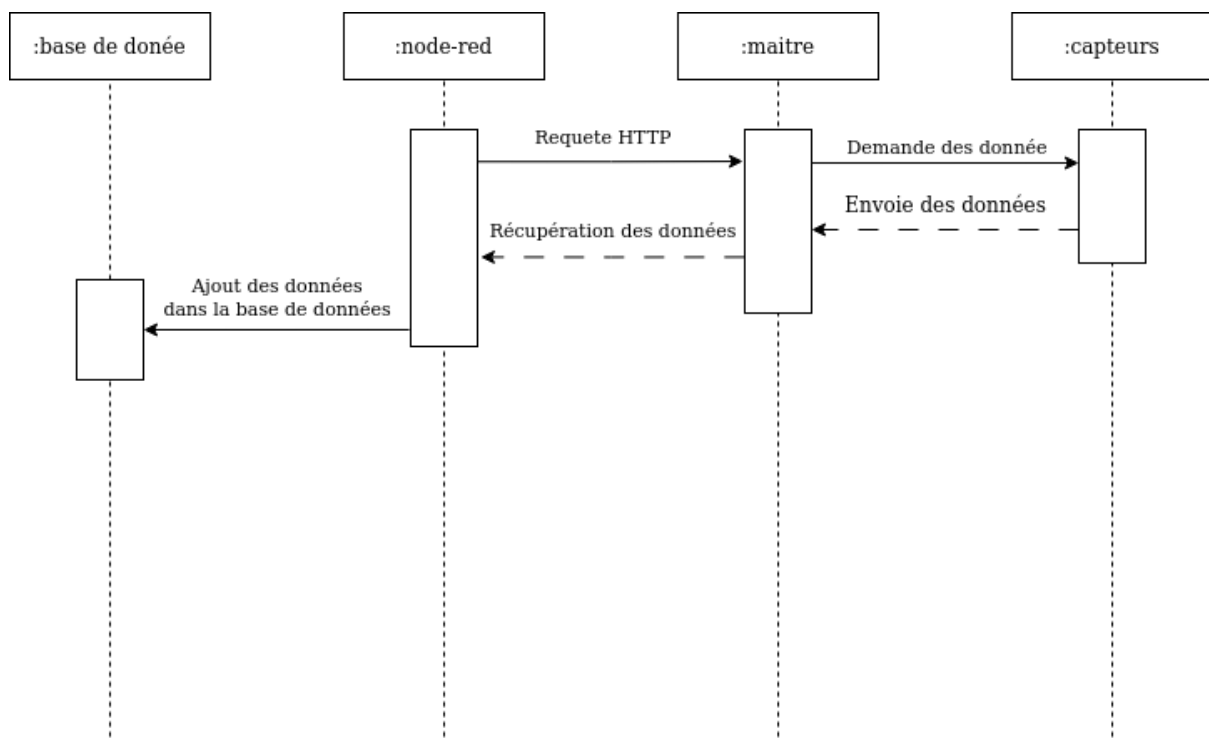
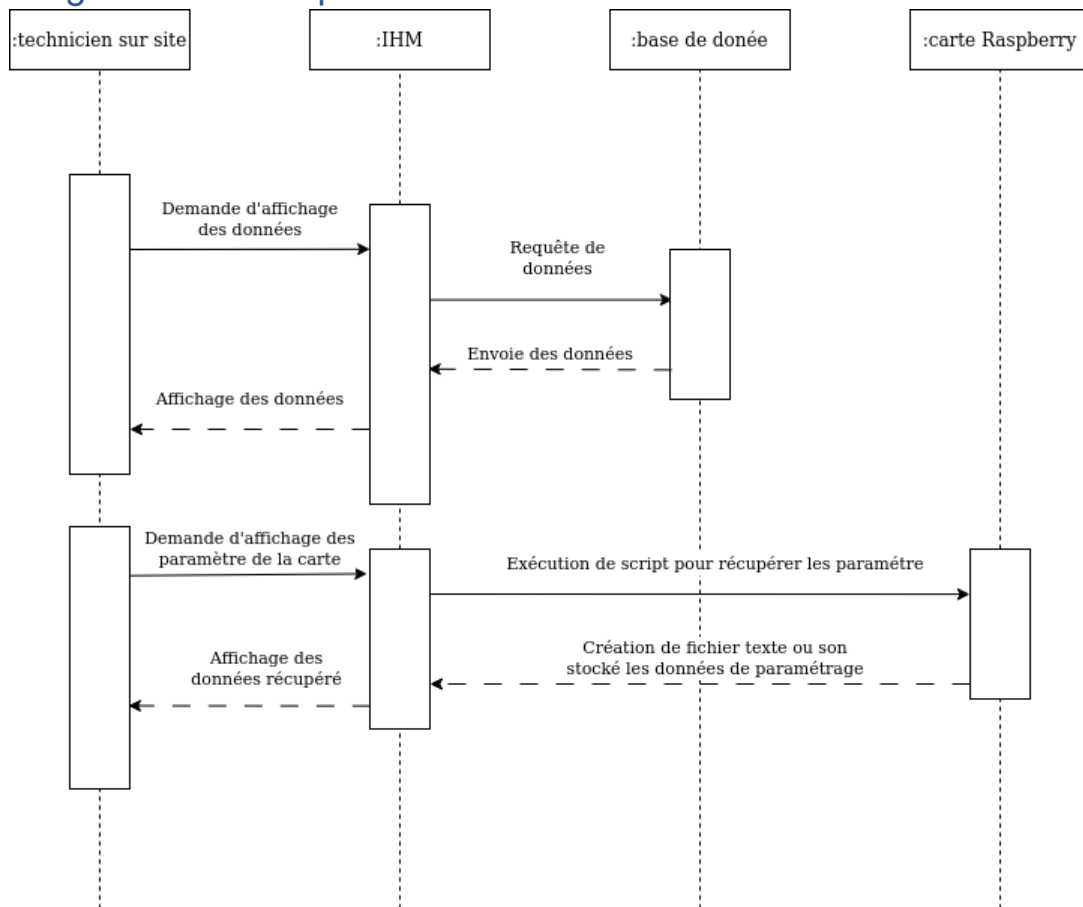


Diagramme de séquence :



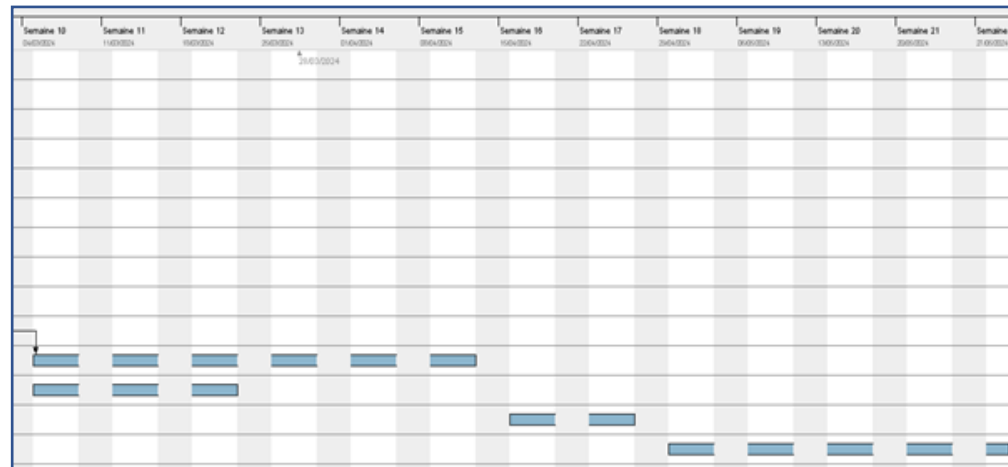
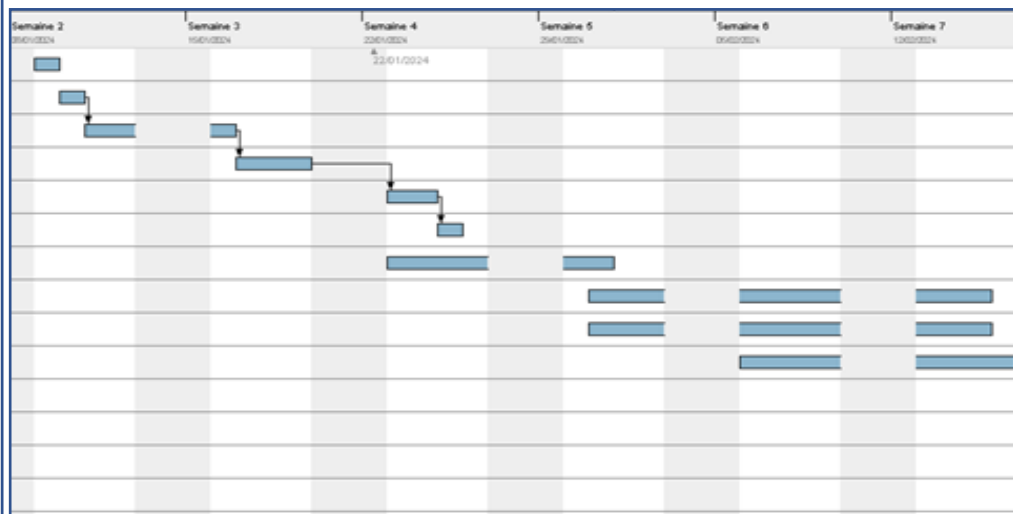
Organisation du groupe

Planning prévisionnel

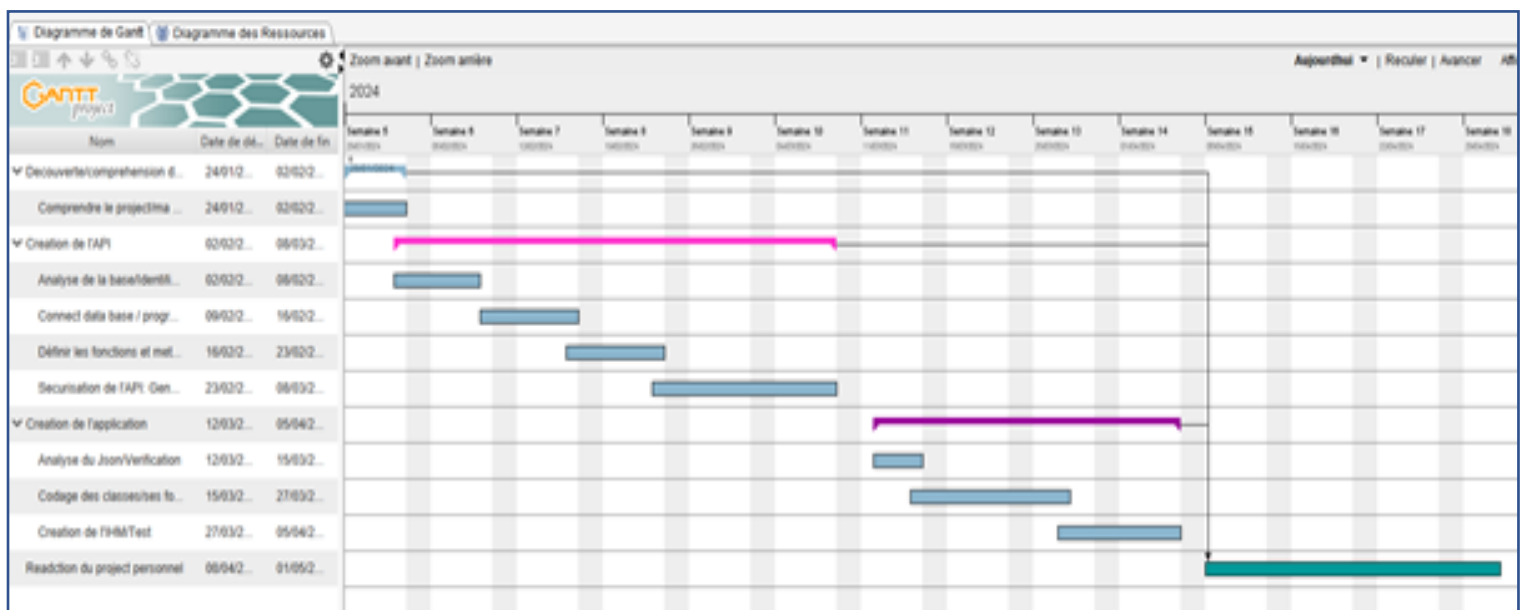
	S5/S6 29/01-09/02	S7/S10 12/02-08/03	S11/S12 11/03-22/03	
Amine	Rédaction et répartition des tache de la partie commune	Création de l'API: connection à la base/les classe et ses fonctions	définir les methodes des fonctions securisation:generer des clés automatique	
Christelle	Rédaction et répartition des tache de la partie commune	Création du site web et de la base de données		
Clement	Rédaction et répartition des tache de la partie commune	Mise en place du par-feux serveur MQTT, nodred, 2ème carte réseaux,	Mise en relation de la base de donnée et les donnée des capteurs	
	S13/S14 25/03-05/04	S15/S18 08/04-29/04	S19/S20 06/05-17/05	S21/22 20/04-31/04
Amine	classe/ses fonctions/ ses definition	interface/afficher les données	Rédaction du rapport de la partie personnelle	
Christelle	Importation du site web et de la base de donnée dans l'interface		Tests et vérifications finales	Rédaction de la partie personnelle
Clement	Création de l'application de gestion sur écran tactile		Rédaction du rapport de la partie personnelle	

Planning prévisionnel de Clément

Gantt project		
Nom	Date de dé...	Date de fin
Recherche des différents logi...	09/01/2...	09/01/2...
Recherche de la carte a utilisé	10/01/2...	10/01/2...
Installation du fire-wall	11/01/2...	16/01/2...
Installation du broker MQTT ...	17/01/2...	19/01/2...
Sécurisation de MQTT	23/01/2...	24/01/2...
Installation de nodRed	25/01/2...	25/01/2...
Cross Compilation pour la car...	23/01/2...	31/01/2...
modification du boitier de la c...	31/01/2...	15/02/2...
Recherche sur la deuxieme c...	31/01/2...	15/02/2...
Recherche pour l'API	06/02/2...	16/02/2...
Création de l'api	05/03/2...	12/04/2...
mise en relation base de don...	05/03/2...	22/03/2...
vérification des relations et te...	16/04/2...	26/04/2...
réctification des possible prob...	30/04/2...	29/05/2...



Planning prévisionnel d'Amine :



Eleve3 : Amine-Dada

Contextualisation :

Dans le cadre du projet de passerelle industrielle 4.0, ma responsabilité consiste à développer l'API REST pour la récupération des données des capteurs. De plus, je suis chargé de créer une application de démonstration utilisant cette API.

Objectifs du projet :

Mon objectif principal est de créer une API REST robuste qui permettra la récupération des données des capteurs. Cette API sera essentielle pour interagir avec la passerelle industrielle 4.0. De plus, je développerai une application de démonstration utilisant cette API avec Qt, afin de mettre en avant les fonctionnalités et la puissance de la passerelle à travers une interface utilisateur intuitive et conviviale.

Documentations :

C'est quoi une API REST ?

Une API REST (également appelée API RESTful) est une interface de programmation d'application (API ou API web) qui respecte les contraintes du style d'architecture REST et permet d'interagir avec les services web RESTful. L'architecture REST (Representational State Transfer) a été créée par l'informaticien Roy Fielding.

Qu'est-ce qu'une API ?

Une API, pour Application Programming Interface (interface de programmation d'application en français), est un ensemble de protocoles qui facilite la communication entre une ou plusieurs applications.

Les API permettent à un produit ou un service de communiquer avec d'autres produits et services, sans forcément connaître comment il a été construit. Cela simplifie le développement d'applications et font ainsi gagner beaucoup de temps et d'argent. Les développeurs réalisent moins de code et contribuent à créer plus de cohérence avec les applications d'une même plateforme.

REST ?

REST est un ensemble de contraintes architecturales. Il ne s'agit ni d'un protocole, ni d'une norme. Les développeurs d'API peuvent mettre en œuvre REST de nombreuses manières.

Voici quelques principes clés de REST :

- Interface uniforme : Chaque action sur une ressource, comme obtenir ou mettre à jour des données, doit être effectuée de manière standardisée.
- Sans état : Chaque requête d'un client à un serveur doit contenir toutes les informations nécessaires pour comprendre et traiter la requête. Il n'y a pas de session client stockée sur le serveur.
- Séparation entre le client et le serveur : Le client ne doit pas se soucier de la logique de stockage des données, qui reste interne au serveur.
- Mise en cache : Les réponses doivent être définies comme cachables ou non, pour éviter la réutilisation de données obsolètes.
- Système en couches : Le client ne doit pas pouvoir distinguer s'il communique directement avec le serveur final, ou avec un intermédiaire.

En résumé, REST est une manière de structurer une API pour qu'elle soit simple, fiable et performante.

Le fonctionnement et UML :

1. Création de l'API REST pour la récupération des données et la commande :

L'objectif principal ici est de mettre en place une API REST (Representational State Transfer) en utilisant PHP. Cette API permettra de récupérer les données des capteurs et de les exposer sous forme de JSON via le protocole HTTP.

Voici comment cela fonctionne :

- **Architecture REST** : L'architecture REST est basée sur le principe de ressources (ou entités) exposées via des URLs (Uniform Resource Locators). Chaque ressource est identifiée par une URL unique, et les opérations (comme la récupération de données) sont effectuées en utilisant les méthodes HTTP standard (telles que GET, POST, PUT, etc.).
- **Récupération des données des capteurs** : L'API sera conçue pour interagir avec les capteurs du système. Lorsqu'un client (comme une application ou un autre service) envoie une requête à l'API, celle-ci récupérera les données des capteurs appropriés et les renverra au client au format JSON.
- **Format JSON** : Le format JSON (JavaScript Object Notation) est largement utilisé pour représenter des données structurées. Il est léger, facile à lire et à écrire, et est pris en charge par la plupart des langages de programmation.
- **Protocole HTTP** : L'API sera accessible via des requêtes HTTP. Par exemple, pour récupérer les données d'un capteur spécifique, le client enverra une requête GET à l'URL correspondante de l'API.

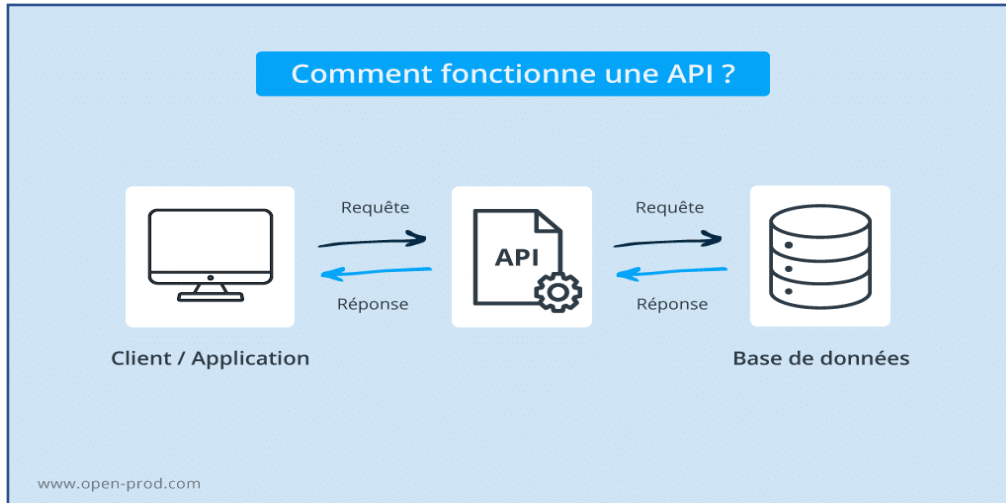
2. Création d'une application de démonstration utilisant l'API :

Après la création de notre API REST pour récupérer les données des capteurs, nous allons créer une application de démonstration pour illustrer son fonctionnement.

Voici comment cela fonctionne :

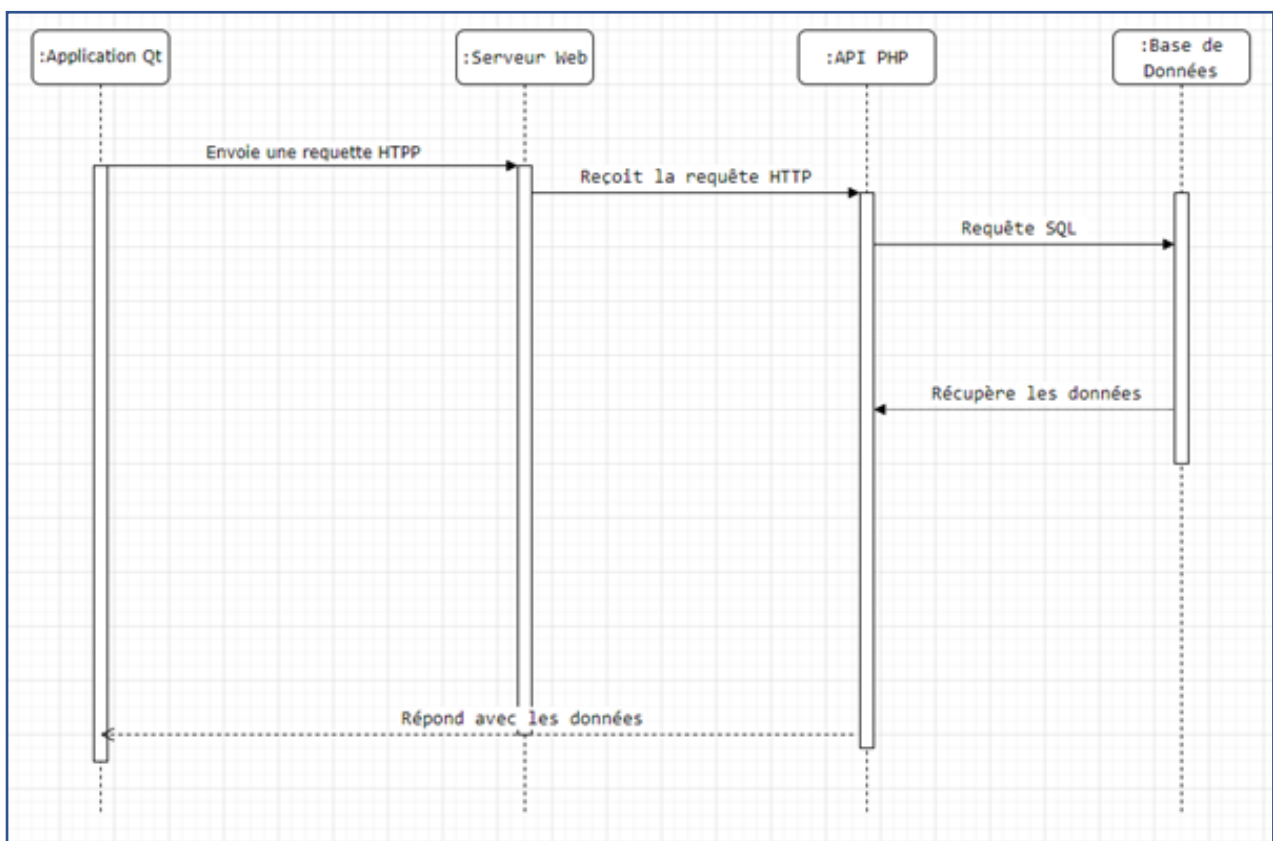
- Interface utilisateur (UI) : Nous allons concevoir une interface utilisateur (probablement une application de bureau) pour permettre aux utilisateurs d'interagir avec l'API. Cette interface affichera les données des capteurs de manière conviviale.
- Programme sous QT Creator : QT Creator est un environnement de développement intégré (IDE) populaire pour la création d'applications graphiques. Nous allons écrire un programme en utilisant QT Creator qui se connectera à l'API, récupérera les données au format JSON et les affichera dans l'interface utilisateur.
- Traitement des données JSON : Le programme QT Creator analysera les données JSON reçues de l'API et les affichera de manière appropriée. Par exemple, on pourrait lui dire d'afficher les valeurs des capteurs dans des tableaux, des graphiques ou d'autres éléments visuels.
- Interaction avec l'API : L'application de démonstration enverra des requêtes à l'API (par exemple, pour récupérer les dernières données des capteurs) et affichera les résultats à l'utilisateur.

3. Voici un schéma simple qui explique mes tâches :



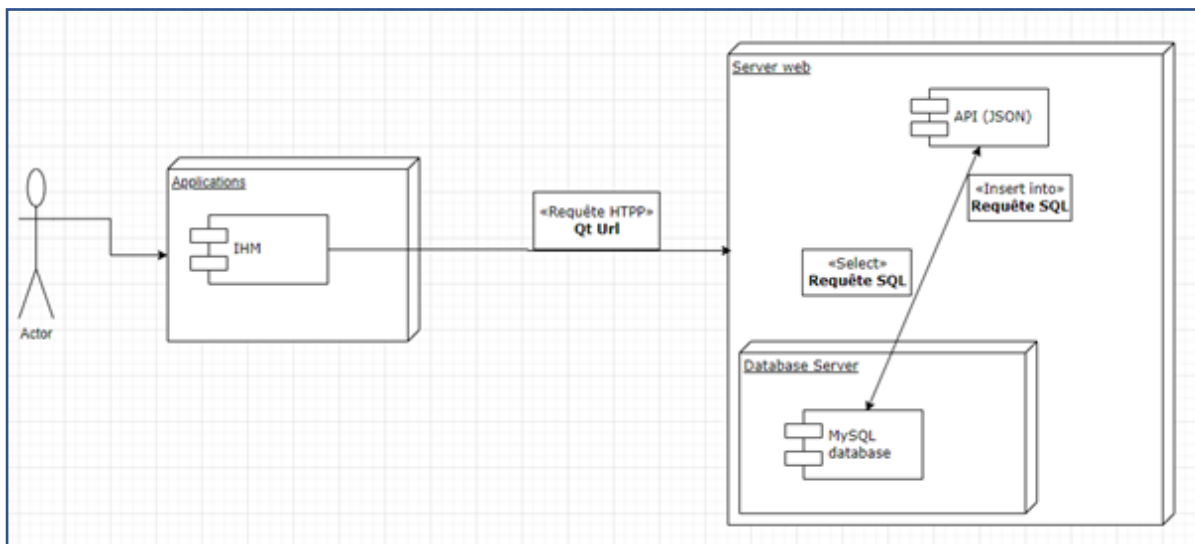
A la fin donc l'application doit être capable d'envoyer une requête à l'API via le réseau du lycée pour avoir les données. Après l'API va faire de même en envoyant une requête à la base de données. Ensuite la base de données répondra à l'API avec les données. Pour finir l'API répondra à l'application avec les données demandées.

4. Voyons ça avec le diagramme de séquence :



Donc notre application va pouvoir envoyer une requête http au server web installé sur la carte pour avoir les données. Le serveur web envoie la requête à l'API. L'API fait de même en envoyant une requête SQL au server SQL. Ensuite la base de données répond avec les données demandées par l'API. Enfin l'API répond directement à l'application avec les données demandées.

5. Voici le diagramme de déploiement



Comme on peut le voir au centre de l'application, nous avons l'IHM (Interface Homme-Machine), qui est le point de contact pour les utilisateurs, leur permettant d'envoyer des requêtes http contenue dans un Qt Url. Ces requêtes sont traitées par le serveur web, qui utilise une API en format JSON pour communiquer avec la base de données. Pour cela l'API exécute des requêtes SQL comme « select » ou « insert » des données à la base de données. Enfin, le serveur de base de données MySQL stocke et gère toutes les informations nécessaires et les renvoie si possible.

Logiciels utilisés :

1. Pour notre API on utilisera ici comme logiciel :



Visual Studio Code (VS Code) est un éditeur de code open-source développé par Microsoft qui offre plusieurs avantages :

Polyvalence : VS Code est un outil polyvalent qui peut être utilisé pour différents types de programmation (dans notre cas on va l'utiliser en PHP).

Rapidité de lancement et Simplicité d'utilisation : Il est connu pour se lancer rapidement et sa simplicité d'utilisation même quand on travaille sur beaucoup de fichier, ce qui en fait un choix idéal pour notre projet.

Intégration avec Git : VS Code offre une bonne intégration avec Git, ce qui facilite la gestion des versions.

Extensions : Il dispose d'un marché d'extensions qui permet d'ajouter de nouvelles fonctionnalités et fonctionnalités à VS Code. Ça permet de bien gérer la gestion de notre API.

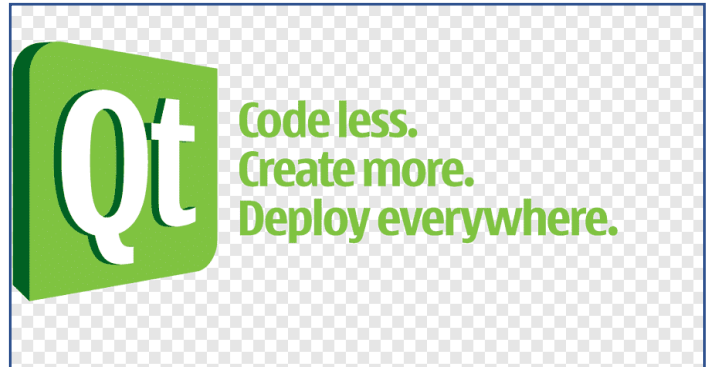
2. Comme vous l'auriez compris j'ai choisi comme langages de programmations de l'API REST le PHP.



PHP, qui signifie PHP Hypertext Preprocessor (auparavant Personal Home Pages), est un langage de scripts généraliste et Open Source, spécialement conçu pour le développement d'applications web. Il peut être intégré facilement au HTML. Le code PHP est exécuté sur le serveur, générant ainsi le HTML, qui sera ensuite envoyé au client. Le client ne reçoit que le résultat du script, sans aucun moyen d'avoir accès au code qui a produit ce résultat.

Pourquoi ? parce que PHP a une excellente intégration avec MySQL (ma collègue a donc décidé de stocker les données sous SQL) : c'est-à-dire se connecter à la base de données MySQL et exécuter des requêtes SQL. De plus PHPMYADMIN est écrit en PHP.

3. Et pour notre application on utilisera QT Creator :



Qt Creator est un EDI (environnement de développement intégré) multiplateformes faisant partie du Framework Qt.

Il est donc orienté pour la programmation en C++. Qt intègre aussi un outil de création d'interfaces graphiques qui est utilisé pour créer des applications mobiles et autres.

Qt intègre aussi de nombreux modules, nous avons absolument besoin :

- Qt Chart - Qui intègre toutes les bibliothèques nécessaires pour gérer l'affichage des graphiques.
- QDebug - Pour écrire la sortie d'une variable dans le terminal de Qt, Pour identifier les éventuels problèmes.

4. Et enfin notre application QT sera sous C++ car c'est le seul langage qu'on a beaucoup programmé en cours.



C++ est un langage de programmation compilé permettant la programmation sous de multiples paradigmes, dont la programmation procédurale, la programmation orientée objet et la programmation générique. Ses bonnes performances, et sa compatibilité avec le langage C en font un des langages de programmation les plus utilisés dans les applications où la performance est critique.

Création de l'API :

Pour la création de l'API il fallait que j'attende que ma collègue crée d'abord la base de données car on veut que l'API exploite les données des capteurs stocké sur la base de données en les affichants sous format JSON.

Après sa création j'ai décidé de copier la base de données sur ma machine à moi car il peut y avoir des pannes, vu que mon collègue faisait l'installation de la carte et le serveur.

Donc voici les différentes tables :

projet_bts connexion	
🔑	c_nom_utilisateur : varchar(255)
🔑	c_mot_de_passe : varchar(255)
🔑	apikey : varchar(255)

projet_bts pression	
🔑	id_press : int(11)
#	pression : decimal(5,2)
🔑	unite : varchar(3)
📅	date_press : date
🕒	time_press : time

projet_bts temperature	
🔑	id_temp : int(11)
#	temperature : decimal(5,2)
🔑	unite : varchar(3)
📅	date_temp : date
🕒	time_temp : time

projet_bts debit	
🔑	id_debit : int(11)
#	pression : decimal(5,2)
🔑	unite : varchar(5)
📅	date_debit : date
🕒	time_debit : time

La table connexion contiens :

- Noms d'utilisateurs
- Mots de passe de chaque utilisateur.
- La colonne apiKeys elle est ajouté par l'API je vous explique pourquoi après.

La table pression contiens :

- L'id.
- La valeur de la pression et son unité.
- La date d'enregistrement ainsi que l'heure.

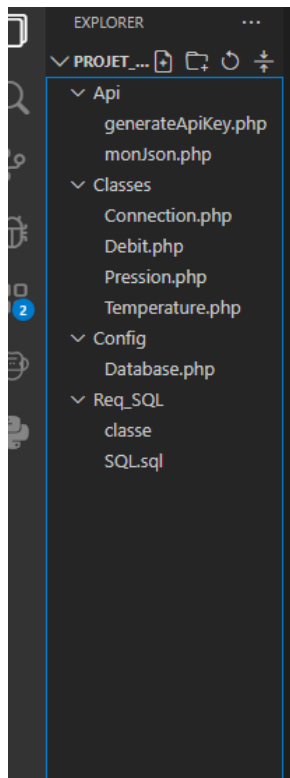
La table température contiens :

- L'id.
- La valeur de la température et son unité.
- La date d'enregistrement ainsi que l'heure.

La table débit contiens :

- L'id.
- La valeur du débit et son unité.
- La date d'enregistrement ainsi que l'heure.

Donc je commence par créer les dossiers où sera stocké mon code.



Tous les dossiers sont contenus dans le dossier **PROJET_BTS**. Ensuite on a un dossier **Api** où sera stocké le code principal pour générer le JSON.

On a le dossier **Classes** où sera stocker les différentes classes qui correspond aux tables. Ici on code la manière dont on va extraire les données.

Et enfin le dossier **Config** pour gérer la connexion.

“Le dossier **Req_SQL** il ne sert à rien pour l’API c’est juste pour que j’ai une trace des requêtes que j’ai déjà effectué.”

1. La connexion :

Pour commencer faut que je fasse d’abord la connexion entre PHP et MySQL comme ça on aura accès à la base de données :

Pour ça dans mon programme j’utilise un objet propre à PHP qui se nomme PDO (PHP Data Object).

PDO est une interface d’abstraction qui permet d’utiliser différents types de base de données en PHP. Ça permet d’exécuter des requête SQL, récupérer des résultats et gérer des transactions.

Voici le code (...\\projet_bts\\config\\database.php) :

```
<?php
class Database{
    // Connexion à la base de données
    private $host = "localhost";
    private $db_name = "projet_bts";
    private $username = "root";
    private $password = "root";
    public $connexion;

    // Constructeur
    public function __construct(){
        $this->getConnection();
    }

    // Méthode pour obtenir la connexion à la base de données
    public function getConnection(){
        $this->connexion = null;
        try{
            $this->connexion = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
            $this->connexion->exec("set names utf8");
        }catch(PDOException $exception){
            echo "Erreur de connexion : " . $exception->getMessage();
        }
        return $this->connexion;
    }
}
```

Je commence par déclarer mes attributs en privée qui sont le nom du serveur où est stocké la table de données, le nom de la base de données, le nom d'utilisateur et le mot de passe pour connecter à la base de données.

Ensuite en public l'attribut connexion ainsi que les fonctions comme getConnection() pour obtenir la connexion. Et le constructeur qui appel la méthode getConnection lors de l'instanciation de l'objet.

Dans la méthode getConnection on utilise l'objet PDO pour tenter de se connecter à la base de données en lui renseignant les attributs déclaré avant.

```
89 // Instancier un objet de la classe Database
90 $database = new Database();
91 $db = $database->getConnection();
92
```

Une fois la connexion établie on va créer une instance de la classe **Database** pour pouvoir utiliser cette connexion dans d'autre classes.

2. Les classes :

Ici je vais uniquement expliquée une des classes car je les ai codées pareil.

Le but dans cette partie c'est de coder la manière dont on extrait les données c'est à dire les requêtes SQL et la présentation du JSON.

Pour que les données soit facilement exploitables avec l'applications, j'ai décidé de les affichées ainsi sur le JSON :

```
{
  "type": "",
  "data": [
    {
      "date": "",
      "heure": "",
      "valeur": {
        "pression": "",
        "unite": ""
      }
    }
  ]
}
```

Voici la structure d'un JSON :

- Un JSON commence par '{' et finie par '}'.

- Ensuite on a une clé : c'est un identifiant unique pour chaque élément de donnée. Les clés sont des chaînes de caractère entourée des guillemets doubles. Dans notre cas notre clé c'est **type**.
- Ensuite on a la valeur de la clé : cela peut être une chaîne de caractère, un nombre, un booléen etc.... Dans notre cas ça sera une chaîne de caractère car on stockera **le type du capteur**.
- Ensuite j'ai mis la clé **data** qui est un tableau d'objet et dedans on a des clés comme **date**, **heure** et **valeur**.
- La clé **valeur** c'est une liste qui contient 2 clés : **l'unité** et la **pression** qui correspond à la valeur du type de capteur.

Voici le code (...\\projet_bts\\classes\\pression.php) :

```

classes > Pression.php
1  <?php
2  class Pression {
3      private $connexion;
4      private $table = "pression";
5
6      public $date_press;
7      public $time_press;
8      public $unite;
9      public $pression;
10
11
12     public function __construct($db) {
13         $this->connexion = $db;
14     }
15
16     public function monJson() {
17         $sql_pression = "SELECT date_press, time_press, unite, pression FROM pression ORDER BY date_press DESC, time_press DESC LIMIT 10";
18         $query_pression = $this->connexion->prepare($sql_pression);
19         $query_pression->execute();
20
21         $pressions = array();
22
23         while ($row = $query_pression->fetch(PDO::FETCH_ASSOC)) {
24             $pression = array(
25                 "date" => $row['date_press'],
26                 "heure" => $row['time_press'],
27                 "valeur" => array(
28                     "pression" => $row['pression'],
29                     "unite" => $row['unite']
30                 )
31             );
32             $pressions[] = $pression;
33         }
34
35         return $pressions;
36     }
37 }
38

```

On commence par déclarer les attributs de la classe **Pression** notamment pour la connexion à la base de données et le nom de la table dans la base de données et les données de la table qu'on veut extraire. Et le constructeur qui a comme paramètre (**\$db**) qui va assurer la

connexion à la table en utilisant la méthode de connexion de la classe **Database**.

Dans la méthode `monJson()`, on code la requête SQL qui nous permet d'extraire les données qu'on veut :

```
$sql_pression = "SELECT date_press, time_press, unite, pression FROM pression  
ORDER BY date_press DESC, time_press DESC LIMIT 10";
```

Ici je sélectionne la date, l'heure, la valeur de la pression et son unité de la table `pression`.

Je les ai rangés en ordre décroissant de la date et l'heure comme ça on aura les valeurs les plus récents au début du JSON car sur la base de données elles sont rangées en ordre croissant.

Et j'ai mis une limite des 10 données. Cependant je ne sais pas si c'est assez pour étudier les données.

Ensuite dans la boucle "while" on décide la manière dont on stock les données dans le JSON.

```
while ($row = $query_pression->fetch(PDO::FETCH_ASSOC)) {  
    $pression = array(  
        "date" => $row['date_press'],  
        "heure" => $row['time_press'],  
        "valeur" => array(  
            "pression" => $row['pression'],  
            "unite" => $row['unite']  
        )  
    );  
    $pressions[] = $pression;  
}
```

Ici on crée un tableau **\$pression** qui va stocker les données relever la requête SQL. Ensuite on prend le tableau `pression` et on le met dans le tableau principal **\$pressions** qu'on a initialiser juste avant la boucle.

Tous les classe dans le dossier Classes sont codé de la même manière sauf la classe Connection qui ne sert strictement à rien :

...\projet_bts\classes\temperature.php

```
Classes > Temperature.php
1  <?php
2  class Temperature {
3      private $connexion;
4      private $table = "temperature";
5
6      public $date_temp;
7      public $temperature;
8      public $time_temp;
9      public $unite;
10
11     public function __construct($db) {
12         $this->connexion = $db;
13     }
14
15     public function monJson() {
16         $sql_temperature = "SELECT date_temp, temperature, time_temp, unite FROM temperature ORDER BY date_temp DESC, time_temp DESC LIMIT 10";
17         $query_temperature = $this->connexion->prepare($sql_temperature);
18         $query_temperature->execute();
19
20         $temperatures = array();
21
22         while ($row = $query_temperature->fetch(PDO::FETCH_ASSOC)) {
23             $temperature = array(
24                 "date" => $row['date_temp'],
25                 "heure" => $row['time_temp'],
26                 "valeur" => array(
27                     "temperature" => $row['temperature'],
28                     "unite" => $row['unite']
29                 )
30             );
31             $temperatures[] = $temperature;
32         }
33     }
34
35     return $temperatures;
36 }
37 }
```

...\projet_bts\classes\debit.php

```
Classes > Debit.php
1  <?php
2  class Debit {
3      private $connexion;
4      private $table = "debit";
5
6      public $date_debit;
7      public $pression;
8      public $time_debit;
9      public $unite;
10
11     public function __construct($db) {
12         $this->connexion = $db;
13     }
14
15     public function monJson() {
16         $sql_debit = "SELECT date_debit, pression, time_debit, unite FROM debit ORDER BY date_debit DESC, time_debit DESC LIMIT 10";
17         $query_debit = $this->connexion->prepare($sql_debit);
18         $query_debit->execute();
19
20         $debts = array();
21
22         while ($row = $query_debit->fetch(PDO::FETCH_ASSOC)) {
23             $debit = array(
24                 "date" => $row['date_debit'],
25                 "heure" => $row['time_debit'],
26                 "valeur" => array(
27                     "pression" => $row['pression'],
28                     "unite" => $row['unite']
29                 )
30             );
31             $debts[] = $debit;
32         }
33     }
34
35     return $debts;
36 }
37 }
```

3. La génération de l'API ...\projet_bts\api\monJson.php:

```
1. <?php
2.
3. include_once '../config/Database.php';
4. include_once '../Classes/Debit.php';
5. include_once '../Classes/Pression.php';
6. include_once '../Classes/Temperature.php';
7.
8. // Headers requis pour gérer les requêtes CORS
9. header("Access-Control-Allow-Origin: *");
10. header("Content-Type: application/json; charset=UTF-8");
11. header("Access-Control-Allow-Methods: GET");
12. header("Access-Control-Max-Age: 3600");
13. header("Access-Control-Allow-Headers: Content-Type, Access-
    Control-Allow-Headers, Authorization, X-Requested-With");
14.
15. // Récupérer la clé API de la requête
16. $api_key = $_GET['api_key'];
17.
18. // Vérifier si la clé API est présente dans la requête
19. if (!isset($_GET['api_key'])) {
20.     http_response_code(401);
21.     echo json_encode(array("message" => "Clé d'API manquante."));
22.     exit;
23. }
24.
25. // Instancier un objet de la classe Database
26. $database = new Database();
27. $db = $database->getConnection();
28.
29. // Vérifier si la clé API est valide
30. if (!$database->isApiKeyValid($api_key)) {
31.     http_response_code(403);
32.     echo json_encode(array("message" => "Clé d'API invalide."));
33.     exit;
34. }
35.
36. // Instancier les objets Vibration et Air
37. $pression = new Pression($db);
38. $debit = new Debit($db);
39. $temperature = new Temperature($db);
40. // Récupérer les données de vibration et d'air et temperature
41. $pressionData = $pression->monJson();
42. $debitData = $debit->monJson();
43. $temperatureData = $temperature->monJson();
44.
45.
46. // Ajout les données de pression au tableau
```

```

47. $data[] = array(
48.     "type" => "pression",
49.     "data" => $pressionData
50. );
51.
52. // Ajout les données debit au tableau
53. $data[] = array(
54.     "type" => "debit",
55.     "data" => $debitData
56. );
57.
58. // Ajout les données de temperature au tableau
59. $data[] = array(
60.     "type" => "temperature",
61.     "data" => $temperatureData
62. );
63.
64. // Afficher les données au format JSON
65. echo json_encode($data);
66. ?>

```

```

include_once '../config/Database.php';
include_once '../Classes/Debit.php';
include_once '../Classes/Pression.php';
include_once '../Classes/Temperature.php';

```

Pour commencer j'inclue d'abord les fichiers nécessaires comme la connexion à la base de données.

```

8 // Headers requis pour gérer les requêtes CORS
9 header("Access-Control-Allow-Origin: *");
10 header("Content-Type: application/json; charset=UTF-8");
11 header("Access-Control-Allow-Methods: GET");
12 header("Access-Control-Max-Age: 3600");
13 header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
14

```

Et pour de la sécurité et de flexibilité des échangent de données entre le serveur et le client je définie les règles des en-têtes comme limiter les méthodes HTTP autorisé à GET, le type de contenu envoyé par la réponse « JSON » et son encodage « UTF -8 ».

```

$database = new Database();
$db = $database->getConnection();

```

J'instancie la classe database pour avoir la connexion a la base de données.

```
$pression = new Pression($db);  
$debit = new Debit($db);  
$temperature = new Temperature($db);
```

Je crée des instances de chaque classe et je passe en argument la connexion à la base de données.

```
$pressionData = $pression->monJson();  
$debitData = $debit->monJson();  
$temperatureData = $temperature->monJson();
```

On appelle la méthode monJson codée dans chaque classe pour récupérer les données sous forme de JSON.

```
$data[] = array(  
    "type" => "pression",  
    "data" => $pressionData  
);  
  
$data[] = array(  
    "type" => "debit",  
    "data" => $debitData  
);  
  
$data[] = array(  
    "type" => "temperature",  
    "data" => $temperatureData  
);
```

On ajoute les données récupérées dans un tableau \$data tout en les structurant comme je le voulais.

```
echo json_encode($data);
```

Et les données sont encodées en JSON en réponse de la requête http qui s'écrit ainsi :

http://localhost/projet_bts/Api/monJson.php?

On peut constater que c'est le chemin où est stocké le code.

4. Sécurisation :

Pour que n'importe qui ne puisse pas avoir accès aux données générées par l'API, j'ai décidé de générer une clé pour chaque utilisateur dans la classe Database.

Voici à quoi ressemble la requête http pour obtenir le JSON :

http://localhost/projet_bts/Api/monJson.php?api_key=660d2031cd4a2

- Dans la classe database :

Donc pour ça on crée dans la table connexion une colonne que j'ai appelée apiKeys et on stock la clé.

Pour éviter que la colonne apiKeys soit créée à chaque instance de la classe database je vérifie d'abord s'elle n'existe pas déjà avec la requête SQL :

```
public function verifiSiApiKeysExiste() {  
    $sql = "SHOW COLUMNS FROM connexion LIKE 'apikey'";  
    $stmt = $this->connexion->prepare($sql);  
    $stmt->execute();  
    $result = $stmt->fetch(PDO::FETCH_ASSOC);  
    return $result;  
}
```

S'elle n'existe pas alors on l'ajoute dans la table connexion :

```
public function addApiKeysColumn() {  
    if(!$this->verifiSiApiKeysExiste()) {  
        $sql = "ALTER TABLE connexion ADD COLUMN apikeys VARCHAR(255)";  
        $stmt = $this->connexion->prepare($sql);  
        $stmt->execute();  
    }  
}
```

On va générer la clé unique :

generateApiKeysSiBesoin() vérifie si la clé n'existe pas déjà sinon il la génère grâce à la fonction **uniqid** qui génère une clé en chaine de caractère basé sur l'heure et date actuel.

getApiKey() récupère la clé existante dans la table

```
public function generateApiKeySiBesoin() {
    $apiKey = $this->getApiKey();
    if(empty($apiKey)) {
        $apiKey = uniqid(); // Génère une clé unique
        $this->storeApiKey($apiKey);
    }
    return $apiKey;
}

public function getApiKey() {
    $sql = "SELECT apikeys FROM connexion LIMIT 1";
    $stmt = $this->connexion->prepare($sql);
    $stmt->execute();
    $result = $stmt->fetch(PDO::FETCH_ASSOC);
    return $result['apikeys'];
}
```

stockApiKey() va stocker cette clé dans la table :

```
public function stockApiKey($apiKey) {
    $sql = "UPDATE connexion SET apikeys = :apikeys";
    $stmt = $this->connexion->prepare($sql);
    $stmt->bindParam(':apikeys', $apiKey);
    $stmt->execute();
}
```

isApiKeyValid vérifie si la clé fournie dans la requête existe dans la table connexion :

```
public function isApiKeyValid($apiKey) {
    $sql = "SELECT * FROM connexion WHERE apikeys = ?";
    $stmt = $this->connexion->prepare($sql);
    $stmt->execute([$apiKey]);
    $rowCount = $stmt->rowCount();
    return $rowCount > 0;
}
```

- Ensuite j'utilise cette méthode de vérification dans le fichier monJson.php :

Si la clé renseignée dans la requête ne correspond pas à celui stocké dans la table alors on affiche une erreur http 403 :

```
if (!$database->isApiKeyValid($api_key)) {  
    http_response_code(403);  
    echo json_encode(array("message" => "Clé d'API invalide."));  
    exit;  
}
```

Mais avant toujours dans monJson on vérifie si la clé est bien présente dans la requête :

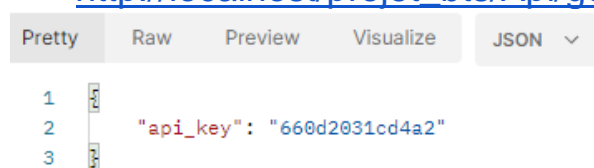
```
if (!isset($_GET['api_key'])) {  
    http_response_code(401);  
    echo json_encode(array("message" => "Clé d'API manquante."));  
    exit;  
}
```

Ensuite pour pouvoir faire la verification avec isApiKeyValid() on l'a stock dans \$api_key

```
$api_key = $_GET['api_key'];
```

- Pour que l'utilisateur ait accès à la clé il doit lancer une requête http :

http://localhost/projet_bts/Api/generateApiKey.php?



```
Pretty Raw Preview Visualize JSON v  
1 {  
2   "api_key": "660d2031cd4a2"  
3 }
```

Voici le code :

```
Api > generateApiKey.php
1  <?php
2  include_once '../config/Database.php';
3
4  // Instancier un objet de la classe Database
5  $database = new Database();
6  $db = $database->getConnection();
7
8  // Générer une clé API aléatoire si nécessaire et la récupérer
9  $apiKey = $database->generateApiKeySiBesoin();
10
11 // Afficher les en-têtes CORS
12 header("Access-Control-Allow-Origin: *");
13 header("Content-Type: application/json; charset=UTF-8");
14 header("Access-Control-Allow-Methods: GET");
15 header("Access-Control-Max-Age: 3600");
16 header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");
17 header("Content-Type: application/json; charset=UTF-8");
18
19 echo json_encode(array("api_key" => $apiKey));
20 ?>
21
```

On génère et on récupère la clé générée par la méthode `generateApiKeySiBesoin` ensuite on l'affiche en format JSON.

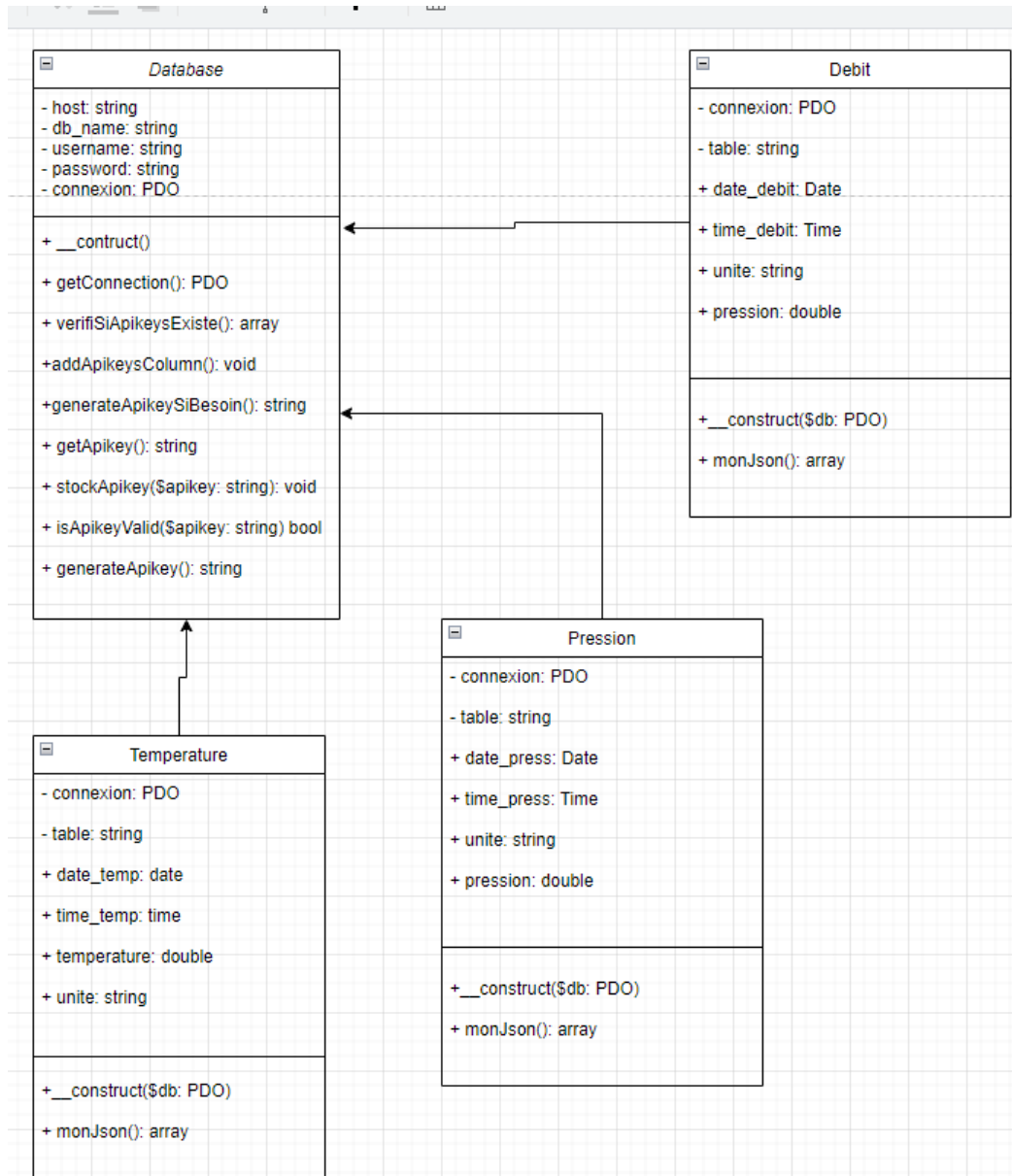
5. Voici à quoi ressemble le JSON :

Je vais les couper en 3 captures d'écran mais ils sont bien sur le même JSON :

<div>▼ 0:</div> <div><div>type: "pression"</div><div>▼ data:</div><div><div>▼ 0:</div><div><div>date: "2024-05-02"</div><div>heure: "01:30:00"</div><div>▼ valeur:</div><div><div>pression: "8.50"</div><div>unite: "bar"</div></div></div><div><div>▼ 1:</div><div><div>date: "2024-05-02"</div><div>heure: "01:00:00"</div><div>▼ valeur:</div><div><div>pression: "7.50"</div><div>unite: "bar"</div></div></div></div></div></div>	<div><div>unite: "bar"</div><div>▼ 1:</div><div><div>type: "debit"</div><div>▼ data:</div><div><div>▼ 0:</div><div><div>date: "2024-05-02"</div><div>heure: "01:30:00"</div><div>▼ valeur:</div><div><div>pression: "80.50"</div><div>unite: "m³/h"</div></div></div><div><div>▼ 1:</div><div><div>date: "2024-05-02"</div><div>heure: "01:00:00"</div><div>▼ valeur:</div><div><div>pression: "70.50"</div><div>unite: "m³/h"</div></div></div></div></div></div></div>	<div>▼ 2:</div> <div><div>type: "temperature"</div><div>▼ data:</div><div><div>▼ 0:</div><div><div>date: "2024-05-02"</div><div>heure: "01:30:00"</div><div>▼ valeur:</div><div><div>temperature: "27.50"</div><div>unite: "°C"</div></div></div><div><div>▼ 1:</div><div><div>date: "2024-05-02"</div><div>heure: "01:00:00"</div><div>▼ valeur:</div><div><div>temperature: "26.50"</div><div>unite: "°C"</div></div></div></div></div></div>
<div>▼ 2:</div> <div><div>date: "2024-05-02"</div><div>heure: "00:30:00"</div><div>▼ valeur:</div><div><div>pression: "6.50"</div><div>unite: "bar"</div></div></div>	<div>▼ 2:</div> <div><div>date: "2024-05-02"</div><div>heure: "00:30:00"</div><div>▼ valeur:</div><div><div>pression: "60.50"</div><div>unite: "m³/h"</div></div></div>	<div>▼ 2:</div> <div><div>date: "2024-05-02"</div><div>heure: "00:30:00"</div><div>▼ valeur:</div><div><div>temperature: "25.50"</div><div>unite: "°C"</div></div></div>
<div>▼ 3:</div> <div><div>date: "2024-05-02"</div><div>heure: "00:00:00"</div><div>▼ valeur:</div><div><div>pression: "5.50"</div><div>unite: "bar"</div></div></div>	<div>▼ 3:</div> <div><div>date: "2024-05-02"</div><div>heure: "00:00:00"</div><div>▼ valeur:</div><div><div>pression: "50.50"</div><div>unite: "m³/h"</div></div></div>	<div>▼ 3:</div> <div><div>date: "2024-05-02"</div><div>heure: "00:00:00"</div><div>▼ valeur:</div><div><div>temperature: "24.50"</div><div>unite: "°C"</div></div></div>
<div>▼ 4:</div> <div><div>date: "2024-05-01"</div><div>heure: "01:30:00"</div><div>▼ valeur:</div><div><div>pression: "4.50"</div><div>unite: "bar"</div></div></div>	<div>▼ 4:</div> <div><div>date: "2024-05-01"</div><div>heure: "01:30:00"</div><div>▼ valeur:</div><div><div>pression: "40.50"</div><div>unite: "m³/h"</div></div></div>	<div>▼ 4:</div> <div><div>date: "2024-05-01"</div><div>heure: "01:30:00"</div><div>▼ valeur:</div><div><div>temperature: "23.50"</div><div>unite: "°C"</div></div></div>
<div>▼ 5:</div> <div><div>date: "2024-05-01"</div><div>heure: "01:00:00"</div><div>▼ valeur:</div><div><div>pression: "3.50"</div><div>unite: "bar"</div></div></div>	<div>▼ 5:</div> <div><div>date: "2024-05-01"</div><div>heure: "01:00:00"</div><div>▼ valeur:</div><div><div>pression: "30.50"</div><div>unite: "m³/h"</div></div></div>	<div>▼ 5:</div> <div><div>date: "2024-05-01"</div><div>heure: "01:00:00"</div><div>▼ valeur:</div><div><div>temperature: "22.50"</div><div>unite: "°C"</div></div></div>
<div>▼ 6:</div> <div><div>date: "2024-05-01"</div><div>heure: "00:30:00"</div><div>▼ valeur:</div><div><div>pression: "2.50"</div></div></div>	<div>▼ 6:</div> <div><div>date: "2024-05-01"</div><div>heure: "00:30:00"</div><div>▼ valeur:</div></div>	<div>▼ 6:</div> <div><div>date: "2024-05-01"</div><div>heure: "00:30:00"</div></div>

On a donc le type du capteur ainsi que le tableau data qui contiens les informations telle que la date, l'heure, la valeur et l'unité. Et les données extrait sont bien celle contenue dans les tables.

6. Voici le diagramme de classe de l'API :

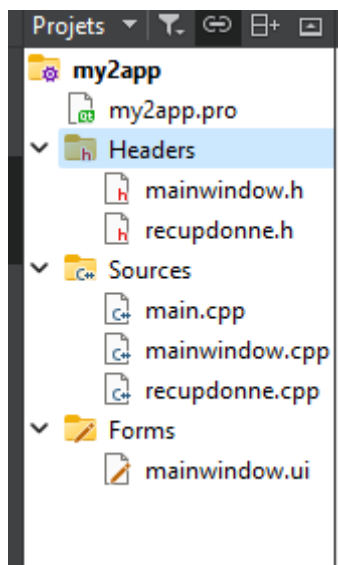


Créations de l'application :

Le but c'est de mettre en place une vue simplifié des données.

J'ai décidé d'afficher les données les plus récentes de chaque capteurs et l'évolutions des données dans le temps.

Voici comment sont structurée mes fichiers qui contienne le code :



Le fichier .pro est le fichier de projet principal pour l'application. Il contient des informations telle que les en-têtes, les fichiers sources et les formulaires...

Le Headers contient les fichiers .h qui contiennent les déclarations de fonctions, des variables...

Dans sources ils contiennent les fichiers .cpp qui contiennent les définitions des fonctions déclarer dans le .h.

Et dans Forms qui contient le .ui qui contient la description de l'interface utilisateur faite avec des widgets.

1. Obtention des données récentes (recupdonne.cpp) :

Pour l'obtention des données de l'url de l'API j'utilise cette fonction **get** qui va effectuer une requête http synchrone en utilisant Qt au serveur web, elle attend la fin de la réponse avant de continuer et gère les erreurs potentielles.

```
346 QByteArray Recupdonne::get(QUrl url) {
347     // Utiliser le QNetworkAccessManager de la classe
348     QNetworkRequest request(url);
349     QNetworkReply *reply = m.get(request);
350     QEventLoop loop;
351
352     // Connecter les signaux et les slots pour gérer la réponse
353     QObject::connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);
354
355     // Attendre que la réponse soit terminée
356     loop.exec();
357
358     // Vérifier s'il y a des erreurs dans la réponse
359     if (reply->error() != QNetworkReply::NoError) {
360         qDebug() << "Erreur de requête HTTP:" << reply->errorString();
361         reply->deleteLater();
362         return QByteArray();
363     }
364
365     // Lire les données de la réponse
366     QByteArray data = reply->readAll();
367
368     // Fermer la réponse et nettoyer
369     reply->deleteLater();
370
371     return data;
372 }
```

J'ai passé l'url en question dans le constructeur :

```
3 Recupdonne::Recupdonne()
4 {
5     qurl = QUrl("http://localhost/projet_bts/Api/monJson.php?api_key=665436867c358");
6     qDebug() << "Requête: " << qurl << Qt::endl;
7 }
```

Donc pour chaque capteur je vais récupérer la date, l'heure et sa valeur.

- Récupération de la date :

Ici on va récupérer la date du capteur température car pour les autres capteurs sont aussi codé pareils :

```

233  QString Recupdonne::get_date_temperature()
234  {
235      QByteArray data = get(qurl);
236      QJsonDocument doc = QJsonDocument::fromJson(data);
237
238      // Vérifier si le document JSON est valide
239      if (!doc.isArray()) {
240          qDebug() << "Erreur: Le document JSON n'est pas valide.";
241          return QString(); // Retourner une chaîne vide en cas d'erreur
242      }
243
244      // Extraire les données de température du document JSON
245      QJsonArray jsonArray = doc.array();
246
247      // Parcourir les éléments du tableau JSON
248      for (const QJsonValue& value : jsonArray) {
249          // Vérifier si l'élément est un objet
250          if (value.isObject()) {
251              QJsonObject obj = value.toObject();
252              // Vérifier si l'objet contient le type "debit"
253              if (obj["type"].toString() == "temperature") {
254                  // Extraire la première date de température et la retourner
255                  QJsonArray temperatureData = obj["data"].toArray();
256                  if (!temperatureData.isEmpty()) {
257                      QJsonObject firstTemperature = temperatureData.at(0).toObject();
258                      QString date = firstTemperature["date"].toString();
259                      return date;
260                  }
261              }
262          }
263      }
264
265      qDebug() << "Aucune date temperature trouvée.";
266      return QString(); // Retourner une chaîne vide si aucune date temeperature n'est trouvée
267  }

```

Donc on comment par appeler la méthode get et on le met comme argument l'url. Cette méthode fait une requête http et retourne les données en QByteArray.

Ensuite on transforme ses données en document JSON et on verifie si le document JSON est un tableau. Si le cas les données sont extraite sous format de tableau JSON

```

QByteArray data = get(qurl);
QJsonDocument doc = QJsonDocument::fromJson(data);

// Vérifier si le document JSON est valide
if (!doc.isArray()) {
    qDebug() << "Erreur: Le document JSON n'est pas valide.";
    return QString(); // Retourner une chaîne vide en cas d'erreur
}

// Extraire les données de température du document JSON
QJsonArray jsonArray = doc.array();

```

Ensuite pour chaque élément du tableau dans la boucle **for** on vérifie si c'est bien du JSON.

Ensuite on vérifie si l'objet contient la clé "type" avec la valeur "température".

Si l'objet "température" est trouvé alors il extrait la clé "data" et retourne la première valeur du tableau qui correspond à la clé "date".

Si aucun objet "température" est trouvé alors il retourne une chaîne vide.

```

// Parcourir les éléments du tableau JSON
for (const QJsonValue& value : jsonArray) {
    // Vérifier si l'élément est un objet
    if (value.isObject()) {
        QJsonObject obj = value.toObject();
        // Vérifier si l'objet contient le type "température"
        if (obj["type"].toString() == "température") {
            // Extraire la première date de température et la retourner
            QJsonArray temperatureData = obj["data"].toArray();
            if (!temperatureData.isEmpty()) {
                QJsonObject firstTemperature = temperatureData.at(0).toObject();
                QString date = firstTemperature["date"].toString();
                return date;
            }
        }
    }
}

qDebug() << "Aucune date température trouvée.";
return QString(); // Retourner une chaîne vide si aucune date température n'est trouvée
}

```

- Récupération de l'heure de pression :

```

15 QString Recupdonne::get_heure_pression()
16 {
17     QByteArray data = get(ql);
18     QJsonDocument doc = QJsonDocument::fromJson(data);
19
20     // Vérifier si le document JSON est valide
21     if (!doc.isArray()) {
22         qDebug() << "Erreur: Le document JSON n'est pas valide.";
23         return QString(); // Retourner une chaîne vide en cas d'erreur
24     }
25
26     // Extraire les données de pression du document JSON
27     QJsonArray jsonArray = doc.array();
28
29     // Parcourir les éléments du tableau JSON
30     for (const QJsonValue& value : jsonArray) {
31         // Vérifier si l'élément est un objet
32         if (value.isObject()) {
33             QJsonObject obj = value.toObject();
34             // Vérifier si l'objet contient le type "pression"
35             if (obj["type"].toString() == "pression") {
36                 // Extraire la première heure de pression et la retourner
37                 QJsonArray pressionData = obj["data"].toArray();
38                 if (!pressionData.isEmpty()) {
39                     QJsonObject secondPression = pressionData.at(0).toObject();
40                     QString heure = secondPression["heure"].toString();
41                     return heure;
42                 }
43             }
44         }
45     }
46
47     qDebug() << "Aucune heure de pression trouvée.";
48     return QString(); // Retourner une chaîne vide si aucune heure de pression n'est trouvée
49 }

```

Pareil ici comme dans la méthode pour récupérer la date température : on fait appel à la méthode **get**, on transforme les données en documents JSON...

Donc à chaque itération la boucle for vérifie si chaque élément du tableau JSON est un objet JSON. Si le cas elle l'extrait.

Ensuite elle vérifie si l'objet contient la clé "type" avec la valeur "pression".

Si c'est le cas alors elle extrait la seconde valeur qui correspond à "heure" de la clé "data" qui est un tableau.

```

// Parcourir les éléments du tableau JSON
for (const QJsonValue& value : jsonArray) {
    // Vérifier si l'élément est un objet
    if (value.isObject()) {
        QJsonObject obj = value.toObject();
        // Vérifier si l'objet contient le type "pression"
        if (obj["type"].toString() == "pression") {
            // Extraire la première heure de pression et la retourner
            QJsonArray pressionData = obj["data"].toArray();
            if (!pressionData.isEmpty()) {
                QJsonObject secondPression = pressionData.at(0).toObject();
                QString heure = secondPression["heure"].toString();
                return heure;
            }
        }
    }
}

qDebug() << "Aucune heure de pression trouvée.";
return QString(); // Retourner une chaîne vide si aucune heure de pression n'est trouvée
}

```

- Récupération de la valeur de débit :

```

193 double Recupdonne::get_debit_press()
194 {
195     QByteArray data = get(qurl);
196     QJsonDocument doc = QJsonDocument::fromJson(data);
197
198     // Vérifier si le document JSON est valide
199     if (!doc.isArray()) {
200         qDebug() << "Erreur: Le document JSON n'est pas valide.";
201         return -1; // Retourner une valeur invalide en cas d'erreur
202     }
203
204     // Parcourir les éléments du tableau JSON
205     QJsonArray jsonArray = doc.array();
206     for (const QJsonValue& value : jsonArray) {
207         // Vérifier si l'élément est un objet
208         if (value.isObject()) {
209             QJsonObject obj = value.toObject();
210             // Vérifier si l'objet contient le type "debit"
211             if (obj.contains("type") && obj["type"].toString() == "debit") {
212                 // Extraire les données debit
213                 QJsonArray debitData = obj["data"].toArray();
214                 if (!debitData.isEmpty()) {
215                     // Extraire la première entrée du tableau
216                     QJsonObject firstDebit = debitData.at(0).toObject();
217                     // Extraire la pression du debit r et la convertir en double
218                     QJsonObject valeur = firstDebit["valeur"].toObject();
219                     double pression = valeur["pression"].toString().toDouble();
220                     qDebug() << "Pression du debit extraite du JSON: " << pression;
221                     return pression;
222                 } else {
223                     qDebug() << "Aucune entrée de données debit trouvée.";
224                 }
225             }
226         }
227     }
228
229     qDebug() << "Aucune donnée debit trouvée.";
230     return -1; // Retourner une valeur invalide si aucune donnée debit n'est trouvée

```

Pareil ici comme dans la méthode pour récupérer la date température : on fait appel à la méthode **get**, on transforme les données en documents JSON...

Dans la boucle **for** on parcourt chaque élément du tableau et on vérifie si c'est un objet JSON.

Il vérifie si on a la clé "type" et si sa valeur est "débit". Si le cas il extrait le tableau "data".

On extrait la première valeur du tableau et on le stock dans **FirstDebit**.

Ensuite on extrait la valeur de pression associé au débit et on le convertie en double.

```

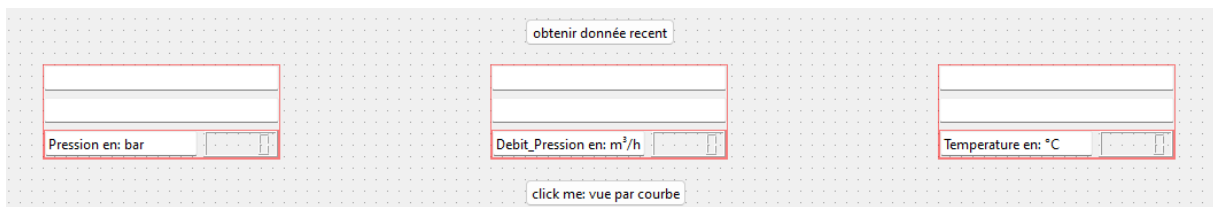
for (const QJsonValue& value : jsonArray) {
    // Vérifier si l'élément est un objet
    if (value.isObject()) {
        QJsonObject obj = value.toObject();
        // Vérifier si l'objet contient le type "debit"
        if (obj.contains("type") && obj["type"].toString() == "debit") {
            // Extraire les données debit
            QJsonArray debitData = obj["data"].toArray();
            if (!debitData.isEmpty()) {
                // Extraire la première entrée du tableau
                QJsonObject firstDebit = debitData.at(0).toObject();
                // Extraire la pression du debit r et la convertir en double
                QJsonObject valeur = firstDebit["valeur"].toObject();
                double pression = valeur["pression"].toString().toDouble();
                qDebug() << "Pression du debit extraite du JSON: " << pression;
                return pression;
            } else {
                qDebug() << "Aucune entrée de données debit trouvée.";
            }
        }
    }
}

```

2. Affichage des données récentes :

- **Dans le mainwindow.cpp :**

On code les actions des widgets faite dans mainwindow.ui :
 Par exemple dès qu'on click sur le bouton « obtenir données récentes » il nous affiche la date, l'heure de la mesure ainsi que la mesure.



Donc pour affichées les données récupérées avec les méthodes de recupdonnees.cpp...j'ajoute la définition du bouton « obtenir donnée recent » que j'ai nommé dans le code « donnee_recent ».

```

void MainWindow::on_donne_recent_clicked()
{
    //affichage de pression
    QString daate = donne.get_date_pression();
    ui->date_pression->setText(daate);

    QString heur = donne.get_heure_pression();
    ui->heur_pression->setText(heur);

    double press = donne.get_pression();
    ui->v_pression->display(press);

    //affichage du debit de pression
    QString datte = donne.get_date_debit();
    ui->date_debit_press->setText(datte);

    QString heure = donne.get_heure_debit();
    ui->heur_debit_press->setText(heure);
}

```

Dès que l'utilisateur click sur le bouton : on fait appel à la méthode **get_date_pression** pour récupérer la date de pression et le stocker dans **daate**. Ensuite on met cette valeur qui est une chaine de caractère dans le widget que j'ai nommé « date_pression ».

On fait appel aussi **get_heure_pression** pour récupérer l'heure récente et on le stock dans heur sous forme de chaine de caractère. Ensuite on met cette valeur dans le widget que j'ai nommé « **heur_pression** ».

Ensuite on fait appel à la méthode **get_pression** pour récupérer la valeur récente et on le stock dans '**press**' sous format double. Et on met cette valeur dans le widget que j'ai nommé « **v_pression** ».

Ensuite c'est pareil pour l'affichage des données récentes des autres capteurs...

obtenir donnée recent		
2024-05-02	2024-05-02	2024-05-02
01:30:00	01:30:00	01:30:00
Pression en: bar 0.5	Debit_Pression en: m³/h 00.5	Temperature en: °C 27.5

3. Vue par courbe :

- Pour que l'utilisateur ait accès à l'historique des données j'ai décidé d'afficher leur évolution sous formats de courbes :

Donc pour ça j'ai créé un nouveau bouton dès qu'il sera cliqué on récupère les données et on trace 3 courbes différentes.

Voici le code :

On a deux méthodes : un premier slot qui va envoyer une requête GET à l'URL et le deuxième slot va traiter les données récupérer par le premier slot.

```
49 void MainWindow::on_pushButton_clicked()
50 {
51     QUrl url("http://localhost/projet_bts/Api/monJson.php?api_key=660d2031cd4a2");
52     networkManager->get(QNetworkRequest(url));
53 }
```

```
void MainWindow::onApiDataReceived(QNetworkReply *reply)
{
    if (reply->error()) {
        qDebug() << reply->errorString();
        return;
    }
}
```

Cette fonction est le deuxième slot qui est déclenché lorsque les données de l'API sont reçues. Il vérifie d'abord s'il y a une erreur dans la réponse. Si c'est le cas, il affiche l'erreur et sort de la fonction.

```
QByteArray responseData = reply->readAll();
QJsonDocument jsonDoc = QJsonDocument::fromJson(responseData);
if (!jsonDoc.isArray()) {
    qDebug() << "Le document JSON n'est pas un tableau.";
    return;
}
```

On lit ensuite toutes les données de la réponse dans un **QByteArray**, puis on crée un **QJsonDocument** à partir de ces données. Si le document JSON n'est pas un tableau, on affiche un message d'erreur et sort de la fonction.

```

QJsonArray jsonArray = jsonDoc.array();
QLineSeries *seriesPression = new QLineSeries();
seriesPression->setName("Pression");
QLineSeries *seriesDebit = new QLineSeries();
seriesDebit->setName("Débit");
QLineSeries *seriesTemperature = new QLineSeries();
seriesTemperature->setName("Température");

```

On crée ensuite un **QJsonArray** à partir du document JSON, et on initialise trois **QLineSeries** pour stocker les données de pression, de débit et de température.

```

QStringList dates; // Pour stocker les dates
QStringList heures; // Pour stocker les heures

```

On crée deux **QStringList** pour stocker les dates et les heures des données.

```

for (const QJsonValue &typeValue : jsonArray) {
    QJsonObject typeObject = typeValue.toObject();
    QString type = typeObject["type"].toString();
    QJsonArray dataArray = typeObject["data"].toArray();

```

On parcourt ensuite chaque élément du tableau JSON. Pour chaque élément, on crée un **QJsonObject**, récupère le type de données (pression, débit ou température) et on crée un **QJsonArray** pour les données.

```

for (const QJsonValue &dataValue : dataArray) {
    QJsonObject dataObject = dataValue.toObject();
    QString dateString = dataObject["date"].toString(); // Récupérer la date
    QString heureString = dataObject["heure"].toString(); // Récupérer l'heure
    QDateTime dateTime = QDateTime::fromString(dateString + " " + heureString, "yyyy-MM-dd
HH:mm:ss");
    QJsonObject valeurObject = dataObject["valeur"].toObject();

```


Pour chaque élément du tableau de données, on crée un autre **QJsonObject**, récupère la date et l'heure, crée un **QDateTime** à partir de la date et de l'heure, et crée un autre **QJsonObject** pour la valeur.

```

32 ~         if (type == "pression") {
33             double pression = valeurObject["pression"].toString().toDouble();
34             seriesPression->append(dateTime.toMsecsSinceEpoch(), pression);
35 ~         } else if (type == "debit") {
36             double debit = valeurObject["pression"].toString().toDouble(); // Assurez-vous que la clé es
37             seriesDebit->append(dateTime.toMsecsSinceEpoch(), debit);
38 ~         } else if (type == "temperature") {
39             double temperature = valeurObject["temperature"].toString().toDouble();
40             seriesTemperature->append(dateTime.toMsecsSinceEpoch(), temperature);
41         }
42
43         // Ajouter la date et l'heure aux listes correspondantes
44         dates.append(dateString);
45         heures.append(heureString);
46     }
47 }

```

On vérifie ensuite le type de données et on ajoute la valeur correspondante à la série appropriée.

```

        // Ajouter la date et l'heure aux listes correspondantes
        dates.append(dateString);
        heures.append(heureString);
    }
}

```

On ajoute ensuite la date et l'heure aux listes correspondantes.

```

// Créer un nouvel axe des X de type date et heure
QDateTimeAxis *axisX = new QDateTimeAxis;
axisX->setTitleText("Date et heure");
axisX->setFormat("yyyy-MM-dd HH:mm:ss");

```

On crée un nouvel axe des X pour le graphique, qui est de type date et heure.

```

114 // Définir la plage de l'axe X avec les premières et dernières dates et heures du JSON
115 QDateTime dateTimeStart = QDateTime::fromString(dates.first() + " " + heures.first(), "yyyy-MM-dd HH:mm:ss");
116 QDateTime dateTimeEnd = QDateTime::fromString(dates.last() + " " + heures.last(), "yyyy-MM-dd HH:mm:ss");
117 axisX->setRange(dateTimeStart, dateTimeEnd);
118

```

On définit ensuite la plage de l'axe X en utilisant les premières et dernières dates et heures du JSON.

```
// Créer le graphique avec les séries et les axes
QChart *chart = new QChart();
chart->addSeries(seriesPression);
chart->addSeries(seriesDebit);
chart->addSeries(seriesTemperature);
chart->setTitle("Évolution des données");
chart->createDefaultAxes();
chart->addAxis(axisX, Qt::AlignBottom); // Ajoute l'axe X au graphique et le positionne
seriesPression->attachAxis(axisX); // Attache la série à l'axe X
seriesDebit->attachAxis(axisX);
seriesTemperature->attachAxis(axisX);
```

On crée ensuite le graphique, ajoute les séries et les axes, et on attache les séries à l'axe X.

```
// Créer la vue du graphique
QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);
chartView->resize(ui->mon_graphic->size());
```

On crée une vue pour le graphique, active l'antialiasing pour un rendu plus lisse, et redimensionne la vue pour qu'elle corresponde à la taille du widget que j'ai nommé « mon_graphic ».

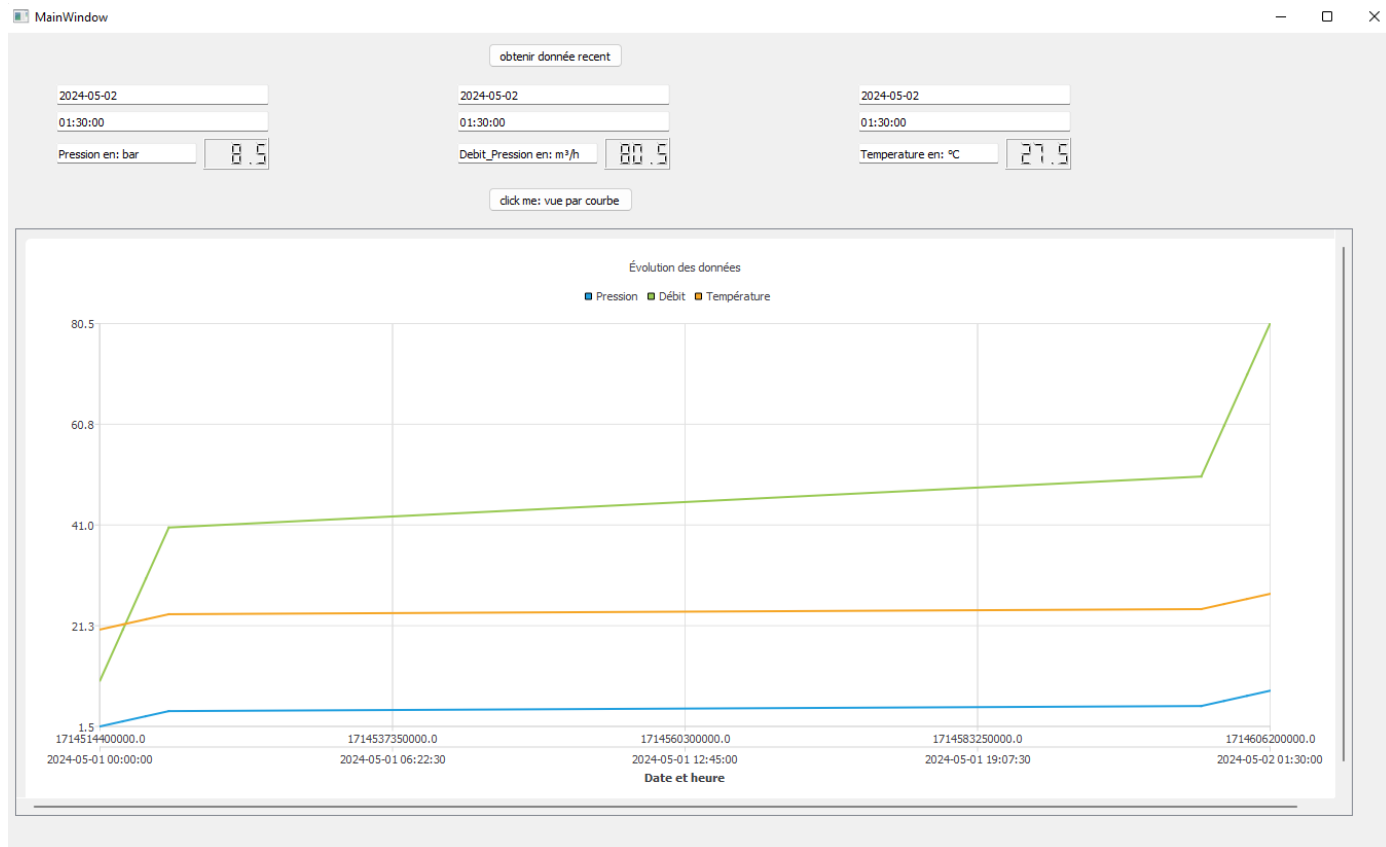
```
// Créer une scène et ajouter la vue du graphique
QGraphicsScene *scene = new QGraphicsScene(this);
scene->addWidget(chartView);
```

On crée une scène, et on ajoute la vue du graphique à la scène.

```
// Afficher la scène dans QGraphicsView
ui->mon_graphic->setScene(scene);
ui->mon_graphic->show();
```

On définit ensuite la scène pour « mon_graphic » et on l'affiche.

Voici donc le graphique :



Conclusion :

La réalisation du projet de passerelle industrielle 4.0 représente une étape importante dans notre parcours académique. Ce projet nous a permis de mettre en pratique nos connaissances théoriques acquises en classe et de développer de nouvelles compétences dans le domaine des systèmes numériques, de l'informatique et des réseaux.

En travaillant sur ce projet, nous avons pu :

- Comprendre les enjeux et les principes de l'industrie 4.0, notamment l'importance de la connectivité, de la surveillance en temps réel et de l'automatisation.
- Acquérir une expérience pratique dans la conception, le développement et l'intégration de systèmes informatiques complexes.
- Collaborer efficacement en équipe pour répartir les tâches, résoudre les problèmes et atteindre les objectifs fixés.
- Apprendre à utiliser une variété d'outils logiciels et matériels, allant des environnements de développement intégrés aux plateformes de gestion de bases de données.

De plus, ce projet nous a permis de développer des compétences transversales telles que la planification, l'organisation, la communication et la résolution de problèmes, qui sont essentielles dans le monde professionnel.

En conclusion, la passerelle industrielle 4.0 est un projet ambitieux qui nous a offert une expérience enrichissante et nous a préparés à relever de nouveaux défis dans notre future carrière professionnelle. Nous sommes fiers du travail accompli et confiants dans notre capacité à contribuer de manière significative à l'innovation et à la transformation numérique.