Projet Weather Data

Dossier de conception

Niranjana Subha Rajeev - 2023

Sommaire

1 - Introduction

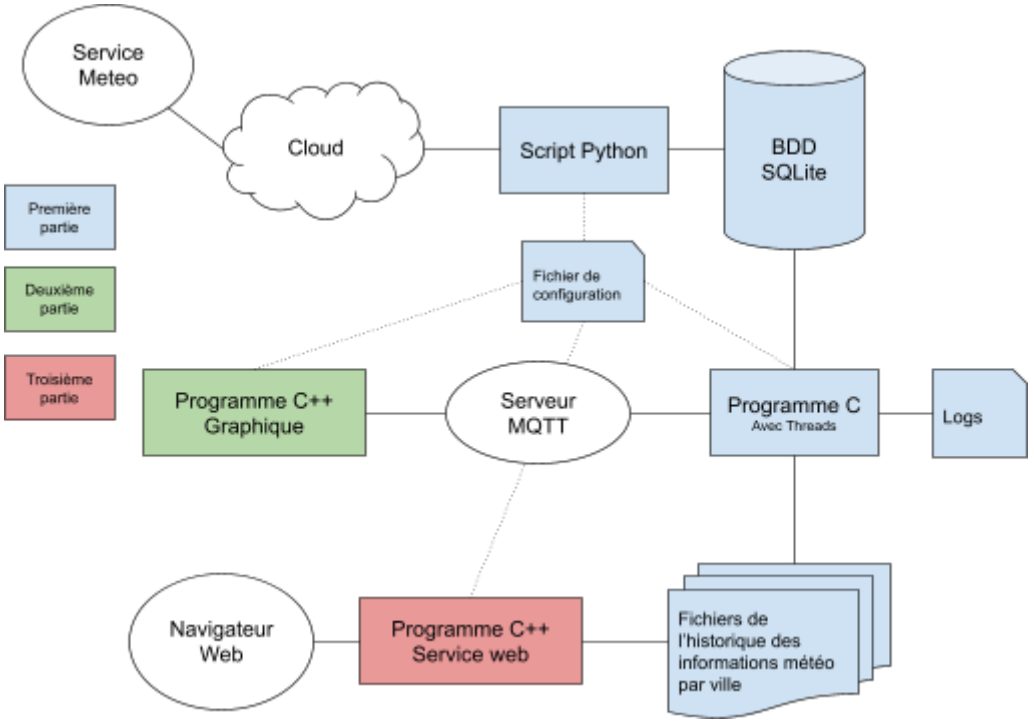
Aperçu général du projet et de son objectif

Ce projet a été mis en place dans le cadre d'une formation POEI encadrée par AJC Formation.

Le but de ce projet est de récupérer des données météorologiques depuis une API existante, et de générer des fichiers d'historiques pour ces données.
L'utilisateur pourra définir des paramètres dans un fichier de configuration unique.
Un affichage pourra permettre à un utilisateur de visualiser les données.

2 - Architecture du projet

L'architecture du projet se présente de la manière suivante :



Ce projet est divisé en trois parties. Chacune de ces parties est définie ci-dessus par un code couleur :

- La première partie (en bleu dans le schéma), est directement liée à un service Météo, qui contient des données météorologiques mises à jour continuellement.
Le script Python utilisera une API de ce service afin de récupérer ces données. Il les transmettra ensuite à la BDD SQLite.
Le programme C utilisera une méthode de polling afin d'utiliser les données de la base, dans le but de créer les fichiers d'historique. Il créera également des fichiers de logs.
Le fichier de configuration permettra au script Python et au Programme C de savoir quelles données récupérer lorsqu'ils feront leurs requêtes respectives.

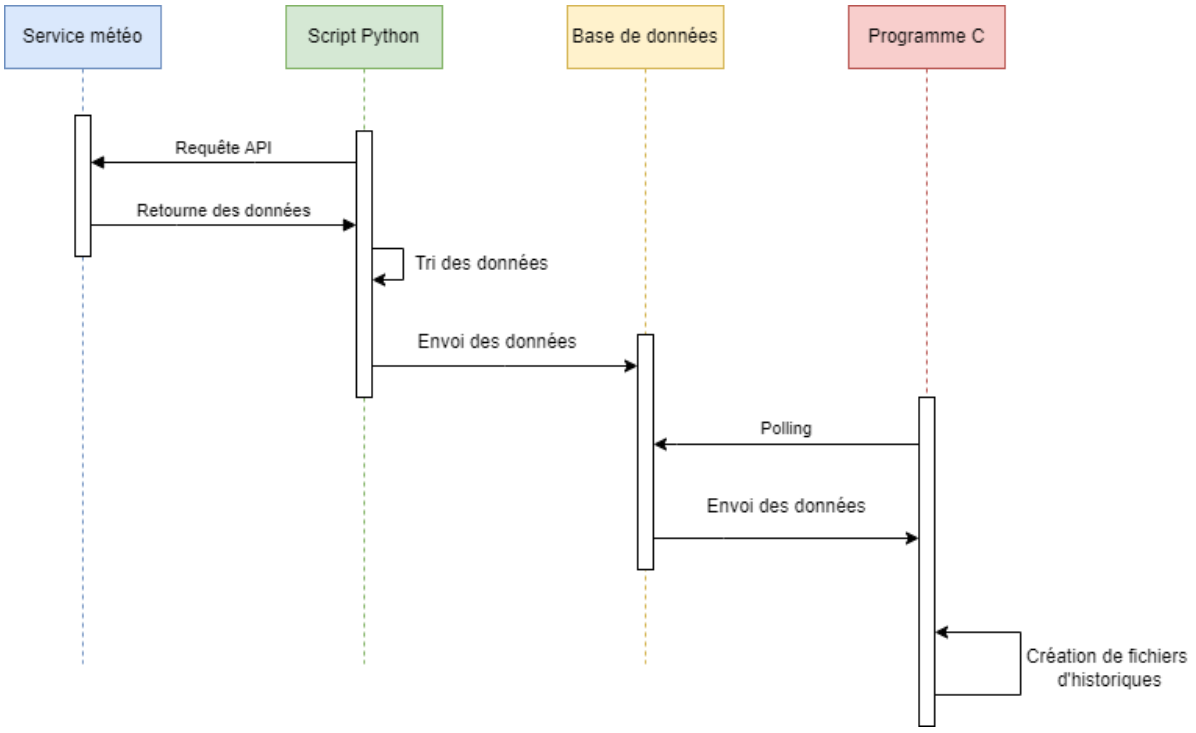
- **La seconde partie** (en vert dans le schéma), inclut l'utilisation du serveur MQTT, et la création d'un programme en C++ permettant d'afficher les données reçues de ce serveur.

- **La troisième partie** (en rouge dans le schéma), permet, par le biais d'un programme en C++ et d'un service web, de permettre à l'utilisateur de récupérer des données des fichiers d'historique depuis un site web.

3 - Diagrammes de séquence

Partie 1

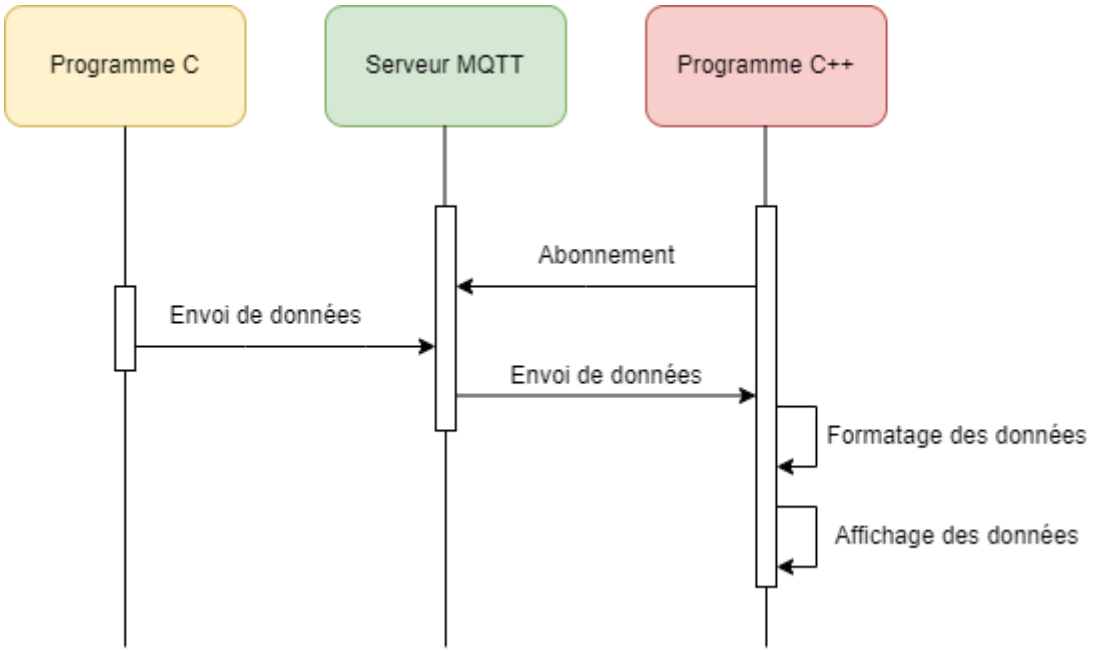
Le diagramme de séquence ci-dessous décrit le déroulé de la première partie du projet. Il ne contient donc pas les parties impliquées dans les parties 2 et 3.



Tout d'abord, le service météo, se verra recevoir une requête de la part du script Python. Cette requête sera basée sur les informations contenues dans le fichier de configuration. Il lui répondra alors avec les données demandées.
Le script Python arrangera ainsi les données, pour les envoyer à la base de données à l'aide d'une requête SQL.
Le programme C réagira à l'ajout de données dans la base de données. Il fera ainsi un polling et la base de données en utilisant le fichier de configuration. La base de données lui renverra ainsi les informations demandées.
Ce même programme C utilisera un système de threads afin de créer en parallèle les fichiers d'historique.

Partie 2

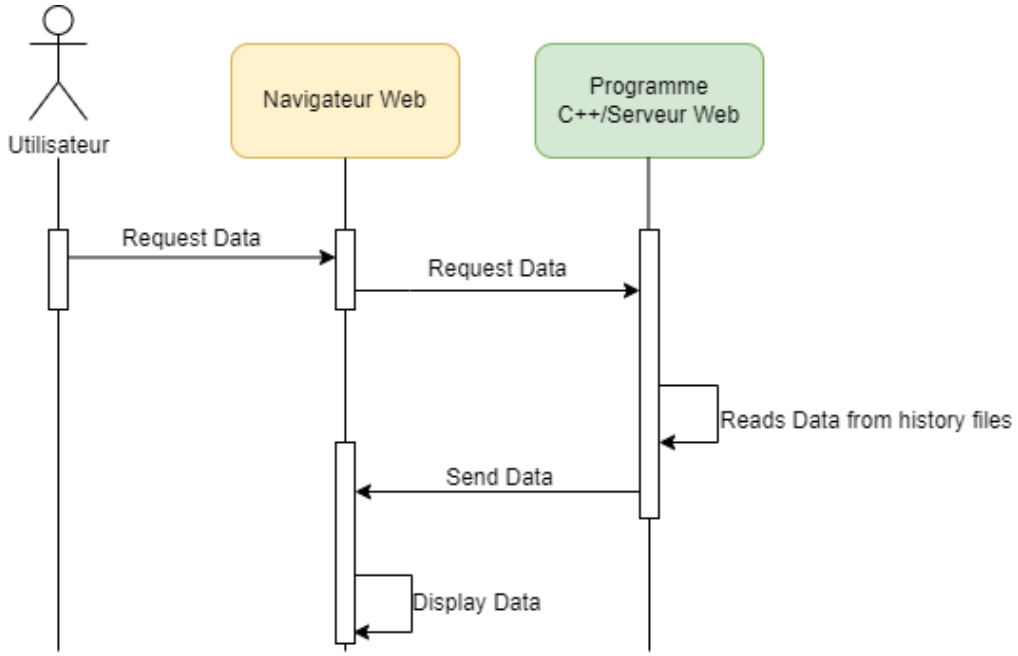
Le diagramme de séquence ci-dessous décrit le déroulé de la seconde partie du projet. Il ne contient donc pas les parties impliquées dans les parties 1 et 3. Il se situe à la suite du premier diagramme de séquence.



Le programme en C, en parallèle de la génération des fichiers (via un système de threads) , va envoyer les données qu'il aura reçues au serveur MQTT, qui va les envoyer à son tour au programme C++. Le programme C++ affichera ces données.

Partie 3

Le diagramme de séquence ci-dessous décrit le déroulé de la troisième partie du projet. Il se situe à la suite des deux premiers diagrammes de séquence.



L'utilisateur aura accès à un navigateur web depuis lequel il pourra interagir. Il pourra ainsi faire une requête, qui sera transmise au programme C++. Ce programme lira les données des fichiers de l'historique, et les renverra au navigateur, pour qu'il les affiche.

4 - Spécifications techniques

Technologies utilisées

Le projet prévoit l'utilisation des technologies suivantes :

- **Python**: Python est un langage de programmation interprété et orienté objet.
  Dans ce projet, un script écrit en python sera utilisé pour communiquer entre l'API et une base de données SQLite.

- **SQLite**: SQLite est une bibliothèque écrite en C qui fournit un moteur de base de données relationnelle accessible via le langage SQL.
  Dans ce projet, elle sera utilisée pour stocker des données météorologiques.

- **C:** Le langage C est un langage de programmation bas niveau procédural.
  Dans ce projet, un programme en C sera utilisé pour récupérer les données de la base de données pour créer les fichiers d'historiques.
  Ce programme permettra également d'envoyer les données à un programme à un serveur MQTT.

- **MQTT**: MQTT (pour Message Queuing Telemetry Transport) MQTT est un protocole de messagerie publish-subscribe basé sur le protocole TCP/IP. Il permettra dans le cadre du projet de faire la transmission entre le programme C et le programme C++.

- **C++**: C++ est un langage de programmation orienté objet, compilé, et avec de nombreux usages. Dans le cadre du projet, un programme en C++ recevra des informations de la part de MQTT pour les afficher à l'aide de QT Creator.

- **QT Creator**: QT Creator est un environnement de développement intégré qui fournit une boîte à outils d'interface graphique pour le développement en C++. Il permettra pour le projet d'afficher les données récupérées.

Versions

- **Python 3**
- **SQLite 3**
- **Mosquitto 2.0.15 (MQTT)**
- **QT Creator 5.15.2**
- **GCC 9.4.0**
- **C++17**

Méthodes

Dans le script Python :

- **“read_config(config_file)”** est une fonction permettant de lire le fichier de configuration. Elle prend le chemin du fichier de configuration en paramètre.

- **“create_table(db_filepath, table_name)”** permet de créer une table dans la base de données si elle n’existe pas. Cette fonction prend en paramètre le chemin du fichier de base de données, et le nom de la table.

- **“fetch_weather_data(api_url, api_key, city)”** permet de récupérer les données météorologiques en créant une requête pour une ville donnée. Elle prend en paramètre les données nécessaires à l’appel de l’api.

- **“update_database(db_filepath, table_name, city, temperature, temp_min, temp_max, humidity, timestamp)”** est une fonction permettant la mise à jour en base de données des données reçues depuis l’api. Elle reçoit en paramètre le chemin du fichier de base de données, le nom de la table et les paramètres que l’on veut envoyer en base de données.

Dans le programme C :

- **“int load_config(const char *filename, Config *config)”** est une méthode permettant de charger le fichier de configuration. Elle prend en paramètre le chemin du fichier de configuration et une structure des paramètres de configuration. Cette dernière sera remplie avec les valeurs du fichier de configuration.

- **“int connect_to_database(const Config *config, sqlite3 **db)”** permet de se connecter à la base de données. Cette méthode prend en paramètre une structure des paramètres de configuration et un pointeur. Ce pointeur sera initialisé avec les informations de connexion à la base de données.

- **“int callback(void *data, int argc, char **argv, char **column_names)”** est une méthode qui va trier les informations reçues suite à l’appel de la base de données.

- **“bool is_database_updated(sqlite3 **db, int *previous_version)”** vérifie si la base de données a été mise à jour. Elle prend en paramètre la base de données et la dernière version des données connues, pour les comparer, et définir si de nouvelles données sont arrivées en base de données.

- **“void* generate_files_thread(void* arg)”** permet de générer les fichiers d’historique en utilisant un système de threading, permettant une génération parralélisée.

- **“void log_action(char* message)”** permet de modifier le fichier de logs. Cette méthode prend un message en paramètre, qu’elle écrira dans le fichier.

Les trois méthodes qui suivent sont utilisées pour la seconde partie du projet.

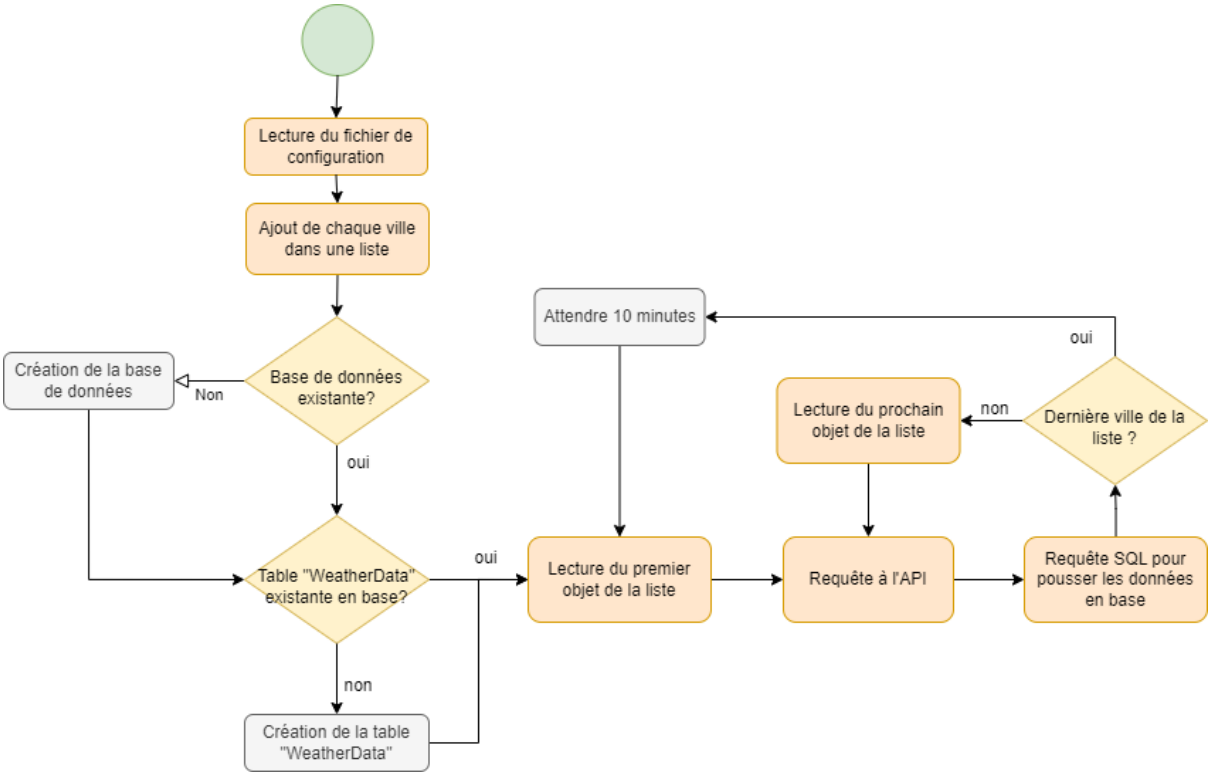
- **“void on_connect(struct mosquitto *mosq, void *userdata, int rc)”** est une méthode de callback permettant de savoir si la connexion est effectuée suite à un essai de connexion au serveur MQTT.

- **“void on_publish(struct mosquitto *mosq, void *userdata, int mid)”** est une méthode de callback permettant de savoir si la publication au serveur a été effectuée suite à un essai de publication de données au serveur MQTT.

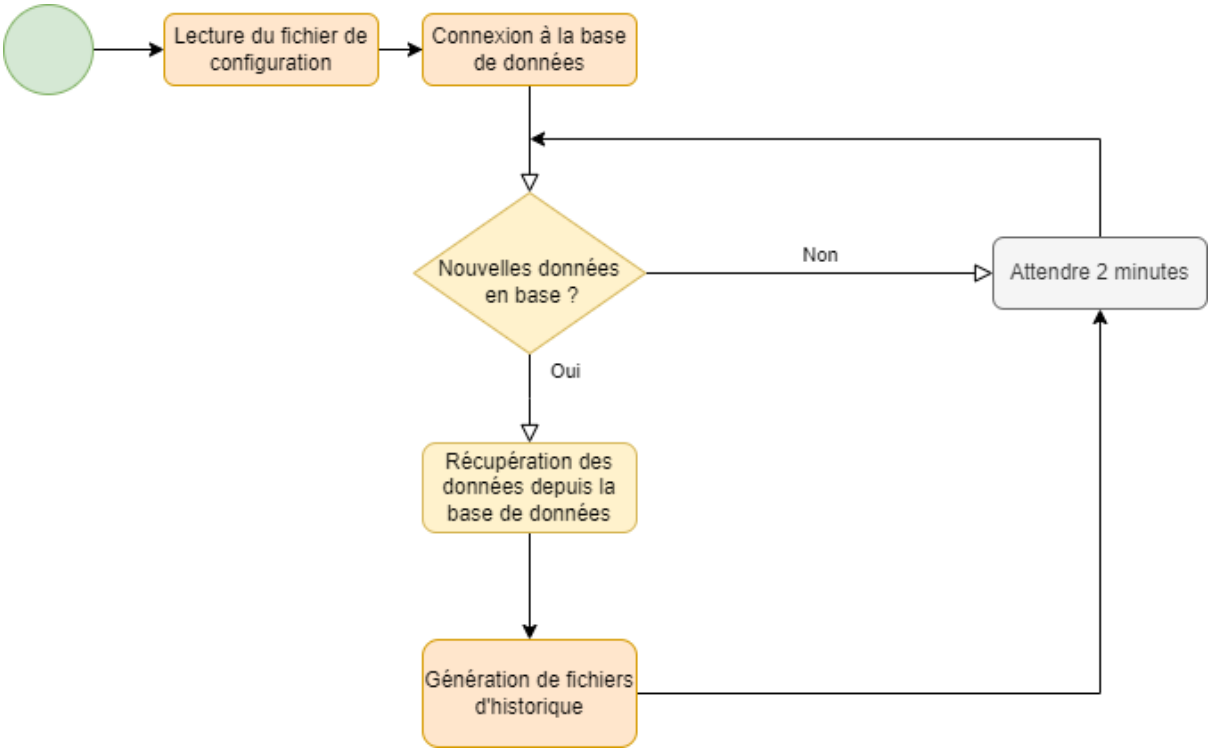
- **“void* publish_data_thread(void* arg)”** permet d’envoyer au serveur MQTT les données reçues de la base de données.

Algorithmes

Algorithme du script Python



Algorithme du programme C



5 - Tests

Afin de tester les méthodes du script Python, le module “unittest” sera utilisé pour réaliser des tests unitaires.

Pour le programme en C, le framework Unity de http://www.throwtheswitch.org/unity sera utilisé afin de réaliser les tests unitaires.

Pour les programmes en C++, Google Test, une bibliothèque de tests unitaires sera utilisée.

6 - Installation

L’installation du projet est détaillée dans le PDF “Dossier d’instructions d’installation et de compilation”.