Météocube
-Documentation-

- Lepetit -

Sommaire :

1. Introduction **P.3**

2. Tech Stack **P.4**
    - Choix des technologies

3. Sonde **P.6**
    - Justification
    - Axe d'amélioration

4. Server **P.8**
    - Justification
    - Axe d'amélioration

5. BDD **P.9**
    - Justification
    - Axe d'Amélioration

6. API **P.10**
    - Justification
    - Axe d'Amélioration

7. Site **P.11**
    - Justification
    - Axe d'Amélioration

8. Lexique **P.12**

Introduction

Ce document est écrit dans le but de nous préparer à la soutenance, aussi bien pour la présentation en groupe qu'en individuel.

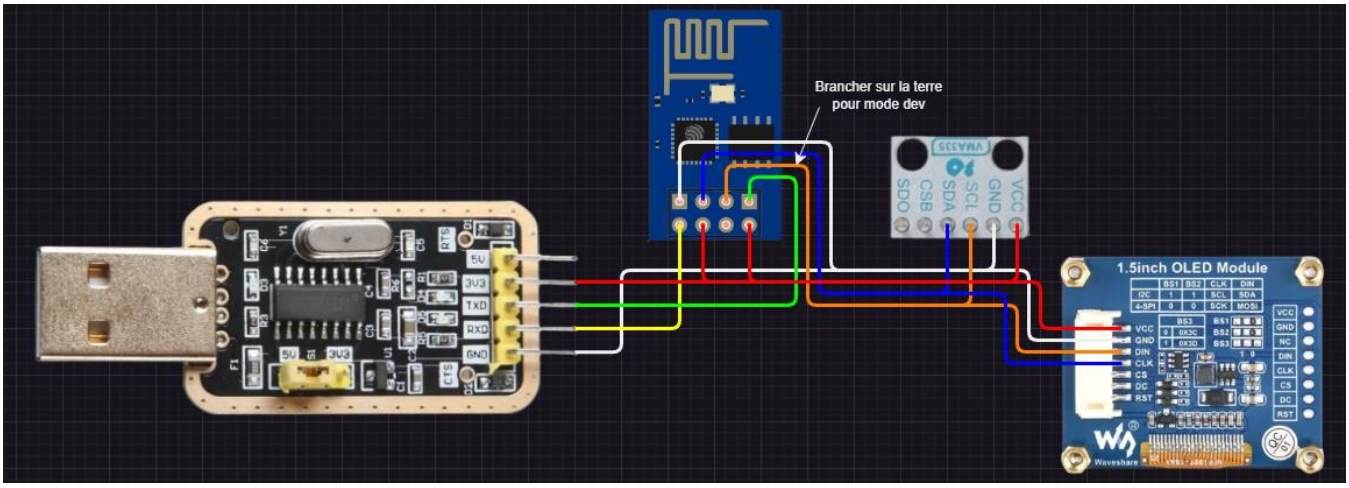
Le fonctionnement du projet et toutes ses parties sera détaillé et justifié, suivi d'une liste d'améliorations potentielles et des termes à rechercher si vous souhaitez une compréhension plus poussée des concepts présents pour chaque partie.

Vous trouverez à la fin de ce document un lexique contenant certains des termes les plus ésotériques et utiles pour la présentation, ces mots ou acronymes seront ***surlignés en gras et italique***.

Tech Stack

Matériel :

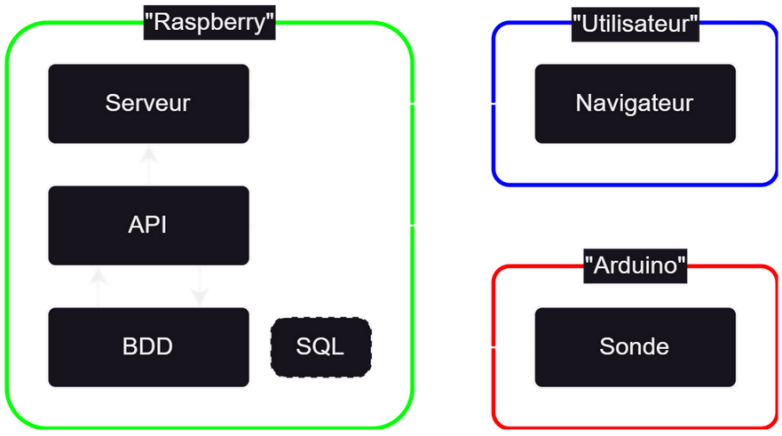
- rock-4c-plus :
  - Un mini-ordinateur contenant notre server, notre base de données, notre site et notre ***API REST.***
- ESP8266 :
  - Une puce capable de se connecter en WIFI, il supporte le couche TCP/IP, permettant de communiquer avec d'autres machines sur le même réseau ou Internet.
  - Cette puce est programmable, elle est ***flashée*** avec notre code C++.
- Ecran I2C (ajouter la référence)
  - Un écran branché à l'ESP8266, utilisé pour afficher les relevés en temps réel et l'IP de l'ESP.
- Sonde (ajouter la référence)
  - Relève la température, l'humidité et la pression ambiante.



Logiciel :

- OS :
  - Une version modifiée de ***Debian*** distribuée par l'entreprise manufacturant les Rocks.
- Server :
  - Apache :
    - Ecoute sur un port donné et traite les requêtes
      C'est notre point d'entrée du site et de l'API.
      Apache peut être configuré par le biais de fichiers de configurations.

- ***API REST*** :
  - Développée en Python, utilise les librairies ***flask*** et ***request***.
- Site :
  - Backend : PHP
  - Frontend : HTML, CSS, JS avec ***Bootstrap*** et ***Chart.js***
- BDD :
  - MariaDB
- Sonde :
  - C++



Sonde

La sonde est constituée de l'ESP8266, l'écran I2C, et le capteur de relevés météorologiques.

Le code C++ dirigeant la sonde est flashé sur l'ESP8266, il gère le capteur, l'affichage à l'écran, la connexion au wifi ainsi que l'envoi de relevés vers l'API. Le programme est composé en 2 parties.

Une phase d'initialisation :

Dès que l'ESP est sous tension, il tente de se connecter au wifi, puis démarre l'écran.

La boucle principale :

Une autre boucle démarre, dans cette boucle un relevé est récupéré via le capteur avant d'être placé dans une structure. Toutes valeurs aberrantes (température de 8864°c par exemple) sont ignorées et on recommence la boucle jusqu'à avoir 5 relevés valide.

On additionne l'humidité, la température, et la pression dans des variables pour garder le total de chaque relevé. Un timer pause le programme pendant 12 secondes. On quitte la boucle.

Ensuite, on fait une moyenne de chaque nombre (somme de toutes les températures divisées par le nombre de température relevé, exemple : (23 + 24 + 22.5 + 23.2 + 22) / 5 = 22.94).

On prépare une requête ***HTTP POST*** vers l'API (exemple de lien : http://127.02.04.10/measures/insert).

On construit une string ***json*** contenant le relevé (exemple : ‘{"temperature": 24, "humidity": 30, "pressure": 998}’)

On exécute la requête ***POST.***

On affiche la moyenne des relevés envoyé sur l’écran I2C.

On recommence la boucle principale du début.

Justifications :

Le C++ est un standard de l’industrie, c’est un langage compilé et performant disposant de nombreuses librairies ce qui réduit le temps de développement de manière significative. De plus, c’est un langage pris en charge par Arduino IDE.

Le code C++ écrit dans le cadre de ce projet n’utilise pas *d’allocation dynamique* (hors librairies), ce qui réduit nettement le risque *de fuite de mémoire* et de crashs.

Le programme passe 5 fois dans une boucle et on attend 12 secondes à chaque ***itération*** pour envoyer une moyenne de 5 relevés toutes les minutes.

Axe d’amélioration :

- Ajout d’un ID propre à chaque sonde permettant leur, cela ajouterait la possibilité de gérer la sonde plus facilement dans la base de données.

- Ajout d’une gestion d’erreur plus poussée si le capteur ou la requête ***POST*** échoue, avec affichage d’un message d’erreur pertinent à l’écran

- Améliorer l’affichage par l’utilisation de pictogrammes

- Faire en sorte que la sonde mette à jour son état lors de chaque démarrage, par une requête ***POST***. Envoyer une requête toutes les 5 minutes pour attester que la sonde est bien démarrée.

Serveur

Le serveur se trouve sur le Rock, lui-même contenant le site, l'API et la BDD.

Le serveur Apache permet la création d'hôtes virtuels, les hôtes virtuels sont des points d'entrés dans nos applications. Deux hôtes sont configurés sur le serveur, un pour le site (IP : localhost, port : 8000), un pour l'API (IP : localhost, port : 5000).

Une fois lancé, le serveur écoute les requêtes entrantes et les traites.

Apache est lancé automatiquement au démarrage du PI

Justifications :

Un serveur moins complexe aurait pu être choisi, cependant Apache possède une documentation fournie, une possibilité de configuration étendue, et une journalisation d'évènements (logs) utile. Ces aspects représentent un atout véritable si l'on considère le projet susceptible d'évoluer.

Axe d'amélioration :

Trouver une méthode pour contacter le serveur depuis l'ESP8266 sans avoir à modifier le code de ce dernier lorsque l'IP du serveur change. Peut-être essayer de configurer */etc/hosts* ou Apache.

BDD

La base de données utilisé par ce projet est MariaDB, une base de données relationnelle possédant trois tables :

- Une table Measures contenant les relevés
- Une table Sensor contenant les différentes sondes liées au Rock
- Une table Users permettant l'inscription et la gestion d'utilisateurs

Chaque relevé est lié à une sonde par une contrainte de clé étrangère assignée à une colonne nommée #id_sensor. Lorsqu'une sonde est supprimée de la table Sensor tous les relevés qui lui sont associés sont eux aussi supprimés.

Justifications :

MariaDB/MySQL comptent parmi les BDD les plus utilisées sur le secteur, elles disposent d'outils pratiques tels que PHPMyAdmin et sont souvent prises en charge par défaut dans les stacks tels que WAMP et LAMP, la mise en place d'un environnement de développement commun s'en retrouve facilité.

Axe d'amélioration :

- Ajout d'une table API, possédant une colonne 'API-KEY' servant à l'identification des potentiels utilisateurs de l'API, pour plus de sécurité

- Mettre en place des opérations automatiques pour vider les relevés trop anciens

API

L'API est la glue assemblant les différentes parties du projet, elle est le seul point d'accès à la base de données et expose des opérations ***CRUD*** sur celle-ci.

Elle est écrite en Python, et utilise ***flask*** et ***request*** pour faciliter le ***routage*** et la récupération des valeurs passées en requête.

Lorsqu'Apache reçoit une requête destinée à l'API, il exécute *index.py*.

Index.py contient des fonctions qui sont appelés selon le lien utilisé pour contacter l'API (exemple : http://127.04.12.1/measures/get/all, /measures/get/all est extrait du lien et la fonction correspondante est appelé)

Ladite fonction construit puis exécute une requête SQL selon les valeurs fournies (si nécessaire)

Le résultat renvoyé par la base de données est encodé en ***json*** et retourné par la fonction.

Justifications :

Le nombre de routes de l'API n'a pas besoin d'être élevé, le programme reste donc simple et la flexibilité de Python se prête parfaitement aux besoins du projet. La décentralisation du serveur fait que le besoin en performance est nettement réduit.

De plus, la présence de nombreuses librairies documentées et simple d'utilisation réduit le temps de développement de manière considérable.

Axe d'amélioration :

- Une meilleur gestion des erreurs avec les bons codes HTTP
- Ajout de routes et de fonctions selon les besoins
- Rendre nécessaire l'authentification des machines essayant d'utiliser l'API, pour plus de sécurité, par le biais de clés uniques enregistrées dans la BDD

Site

Le site envoie des requêtes GET à l'API pour récupérer les derniers relevés, ainsi que les sondes ajoutées à la BDD. Il permet la représentation de ces relevés par des graphiques, grâce à ***chart.js***. L'utilisateur peut supprimer des sondes ou en ajouter selon ses besoins.

Il est responsive et utilise ***Bootstrap.***

Justifications :

La partie visible de l'iceberg, c'est l'interface avec lequel les utilisateurs finaux seront familiers lorsqu'ils achèteront le produit

Il doit être simple d'utilisation, engageant et capable de s'adapter à différents appareils, aussi bien ordinateurs que téléphones.

Dans ce but, il a été convenu d'utiliser ***Bootstrap***, une librairie CSS permettant de créer des pages ***responsives*** avec plus de simplicité, en utilisant directement les attributs classes dans l'HTML.

Axe d'amélioration :

- Ajout d'utilisateurs, avec login et inscription (gérée sur un panneau admin)

Lexique

API REST :

- Programme utilisé pour assurer la communication entre différents services (client communiquant avec un serveur via une API par exemple). Le client peut demander des données stockées dans la BDD sans avoir à faire une requête SQL ou avoir connaissance de la structure exacte d'une BDD. L'API s'en charge.

Flasher :

- Ecrire un programme dans la mémoire flash d'un microprocesseur.

Debian :

- Système d'Exploitation basé sur Linux souvent utilisé pour accueillir des servers du fait de sa stabilité.

Flask :

- Librairie python faisant office de web framework minimaliste car il ne requiert aucune librairie supplémentaire pour être utilisé.

Bootstrap :

- Framework CSS permettant de styliser des pages HTML et les rendre responsive facilement.

Request :

- Librairie python facilitant la prise en charge de requêtes HTTP.

Chart.js :

- Librairie javascript aidant la création de graphiques.

Responsive :

- Une approche au design web prioritisant un affichage adapté sur différentes tailles d'écrans.

HTTP :

- Protocole du modèle TCP/IP régissant les communication internet, par le biais de requêtes et de réponses.

POST :

- Type de requête HTTP servant à envoyer des données.

GET :

- Type de requête HTTP pour demander des données.

PUT :

- Type de requête HTTP pour mettre à jouer des données existantes.

DELETE :

- Type de requête HTTP pour supprimer des données.

CRUD :

- ***Create, Read, Update, Delete***, des opérations courantes sur bases de données.

Itération :

- Désigne un passage dans une boucle.

Routage :

- Pratique consistant à définir des chemins spécifiques accessible par d'autres machines/programmes lorsqu'ils le demandent et dans le cadre d'une API ou un Site de fournir différentes fonctionnalités selon le chemin.

Json :

- Un format texte très courant permettant de représenter des données.