

**STATION MÉTÉO à base de Raspberry Pi**



**Alexia TURINES / Maxime COLINET**

# **Historique du rapport**

Le tableau suivant fournit un récapitulatif des principales modifications effectuées dans le rapport.

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Auteur | Description de la mise à jour |
| V0 | 12/04/2019 |  | Création du rapport |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table des matières**

[Historique du rapport 2](#_Toc7433058)

[1. Cahier Des Charges 4](#_Toc7433059)

[1.1 Périmètre du projet 4](#_Toc7433060)

[1.2 Équipe projet 4](#_Toc7433061)

[1.3 Tâches, jalons et livrables 4](#_Toc7433062)

[1.4 Planification du projet 5](#_Toc7433063)

[1.5 Diagramme de Gantt 6](#_Toc7433064)

[1.6 Indicateurs de pilotage 7](#_Toc7433065)

[1.7 Quality Assurance 8](#_Toc7433066)

[2. Cahier Des Charges Fonctionnels 9](#_Toc7433067)

[2.1 Contexte 9](#_Toc7433068)

[2.2 Produit 9](#_Toc7433069)

[2.3 Environnement logiciels 13](#_Toc7433070)

[2.4 Fonctionnalités 14](#_Toc7433071)

[2.5 Algorithme de ZAMBRETTI 15](#_Toc7433072)

[2.6 Code 19](#_Toc7433073)

[2.7 Interface de visualisation des données 23](#_Toc7433074)

[2.8 Qt Designer de Qt Creator 24](#_Toc7433075)

[3. Retour d’expérience 25](#_Toc7433076)

[3.1 Suivi des ressources 25](#_Toc7433077)

[3.2 Suivi de la planification du projet 25](#_Toc7433078)

[3.3 Avantages/inconvénients de l’environnement technique 25](#_Toc7433079)

TABLE DES ILLUSTRATIONS 27

# **Cahier Des Charges**

* + 1. Périmètre du projet

Ce projet s’insère dans la formation C++ Linux embarqué qui va permettre de mettre en application les acquis de cette formation. Celui-ci va mettre en œuvre un projet C sur une carte Raspberry Pi.

L’objectif de ce projet est la réalisation d’une station météo avec pour contraintes matérielles l’utilisation d’un Raspberry Pi 3 B+ et d’un capteur BME280 à travers une interface graphique.

* + 1. Équipe projet

L’équipe se compose de 2 personnes : Alexia et Maxime.

Le profil de l’équipe est le suivant :

* Alexia : ingénieur à dominante qualité (système et produit) avec une double compétence en système d’information
* Maxime : ingénieur informatique orienté sécurité informatique

Pour l’organisation du projet, Alexia s’est occupée d’une grande partie de la maîtrise d’ouvrage, chargée de projet/Scrum master ainsi que de la partie interface graphique avec Qt Designer de Qt Creator et Maxime s’est occupé d’une grande partie de la maîtrise d’œuvre ainsi que des tests fonctionnels.

* + 1. Tâches, jalons et livrables

La réunion d’ouverture du projet a eu lieu le vendredi 12 avril 2019. Nous avons défini les différentes tâches, les jalons et les livrables du projet.

Les tâches qui ont été définies sont les suivantes :

* Les réunions d’avancement du projet avec la réunion d’ouverture, les réunions d’avancement journalières (1/4h), les réunions d’avancement hebdomadaires (1h) et la réunion de clôture
* Le rapport du projet avec la réalisation d’un cahier des charges, d’un cahier des charges fonctionnel et technique, un diagramme de Gantt, sa rédaction et son rendu
* La réalisation du projet en lui-même avec la création de l’interface graphique, la programmation et l’association interface graphique/code
* La présentation du projet devant un jury avec la préparation d’un PowerPoint, 2 soutenances blanches (prévues le jeudi 18 et 25 avril 2019 d’une durée de 30 min) et la présentation finale le jeudi 2 mai 2019 (durée de 30 min).

Les livrables sont donc un rapport écrit, un PowerPoint de présentation et la station météo en fonctionnement. Les jalons ont été définis en fonction des soutenances blanches et de la présentation finale. La dead-line du projet est le mardi 30 avril 2019 avec une présentation au jury le jeudi 2 mai 2019.

* + 1. Planification du projet

Le projet a été découpé en 4 phases qui définissent :

* les tâches principales avec les sous-tâches associées
* par qui les tâches ou sous-tâches doivent être réalisées
* les livrables à présenter
* la validation des jalons.

Pour cela, nous nous sommes appuyés sur une WBS (Work Breakdown Structure) qui nous permettra de définir les différentes tâches à accomplir durant ce projet avec l’établissement d’une liste des livrables.

Figure 1 : WBS (Work Breakdown Structure)

* + 1. Diagramme de Gantt

Après avoir défini la WBS, nous avons réalisé le planning du déroulement du projet sous forme de diagramme de Gantt et évalué le temps nécessaire pour chaque tâche.

Cela nous a permis de renseigner et situer dans le temps les phases, les tâches et ressources du projet. En ligne, nous avons listé les tâches et en colonne les jours. Les tâches sont représentées par des barres dont la longueur est proportionnelle à la durée estimée.

Les livrables sont représentés par des losanges dans ce diagramme de Gantt :



Figure 2 : Diagramme de Gantt

Nous avons choisi de gérer certaines phases du projet en mode **AGILE SCRUM** pour une meilleure gestion de la productivité. Le rôle de Scrum Master a été occupé par Alexia.

Chaque jour, nous avons organisé en début de journée une réunion d’avancement (1/4h), aussi appelée Scrum ou Mêlée, afin de s’assurer que les objectifs seraient tenus. Elle nous permettait de présenter ce qui avait été réalisé la veille et de parler des éventuels problèmes rencontrés, de savoir ce qui allait être fait dans la journée. Ça nous a permis de coordonner notre travail et de nous aider mutuellement sur les difficultés et de prévoir de possibles évolutions du projet.

Pour structurer la planification des livrables du projet, nous avons utilisé un autre moyen de l’Agile Scrum qui est un story mapping. Ça consiste en une organisation en deux dimensions des user stories : l'axe horizontal matérialise la succession des usages de l'utilisateur, l'axe vertical matérialise les fonctionnalités.

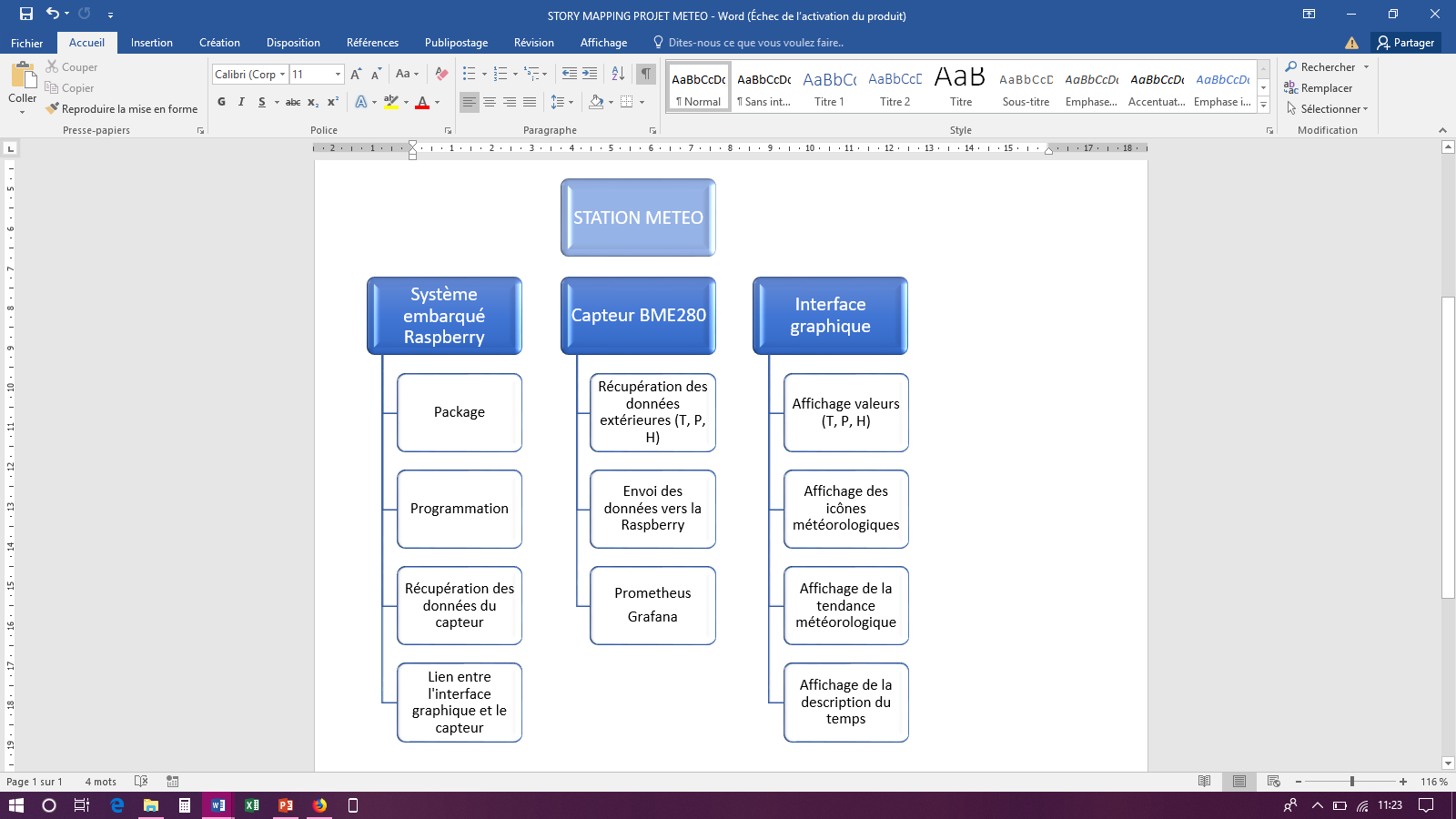


Figure 3 : Story Mapping des user stories

Pour gérer et suivre l’évolution des différentes phases du projet, nous avons utilisé une deuxième méthode Agile qui est le KANBAN intégré directement dans le GitHub de notre projet.

Voici le Kanban final du projet :

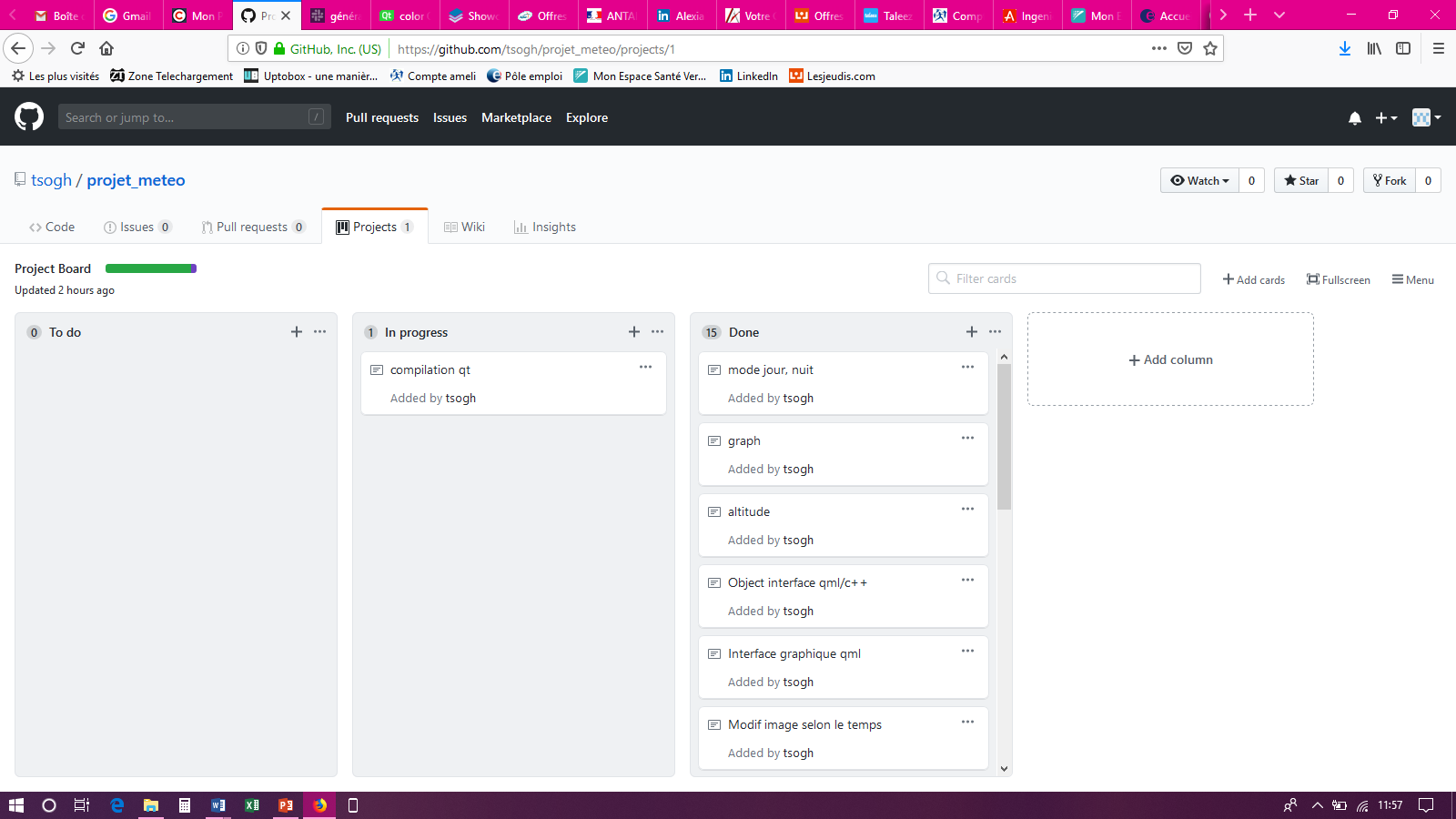


Figure 4 : Kanban (GitHub)

* + 1. Indicateurs de pilotage

Pour suivre l’avancement correct de notre projet, le chef de projet/scrum master a mis en place un tableau de bord ou « scoreboards » afin de regrouper les indicateurs de pilotage.

Le choix des indicateurs doit respecter les critères SMART, c’est-à-dire qu’ils doivent être Spécifiques, Mesurables, Acceptables, Réalistes et Temporellement définis.

Voici donc les indicateurs de pilotage du projet que nous avons choisis :



Figure 5 : Tableau des indicateurs de pilotage du projet

* + 1. Quality Assurance

Dans le cadre du développement de produits et de services, l'assurance qualité (ou QA pour Quality Assurance) désigne tout processus systématique visant à vérifier qu'un produit ou un service en cours de développement est conforme à des exigences précises.

Concernant notre projet, la chargée de projet/scrum master a décidé en relation avec le développeur d’effectuer deux tests unitaires. Ce choix s’est fait en tenant compte de la durée du projet. Etant de 3 semaines, si nous voulions amener le projet à terme en respectant la contrainte temps, nous avons priorisé le livrable.

Les deux tests unitaires ont concerné le calcul de prévision qui utilise l’algorithme de Zambretti. Pour le calcul de la prévision, nous pouvons utiliser l’une des 2 méthodes de calcul possibles. Il était nécessaire de les tester afin de vérifier les résultats en fonction de la méthode utilisée. Cela nous a permis aussi de définir quelle méthode était la plus précise mais nous avons quand même décidé de donner le choix au client d’utiliser l’une ou l’autre méthode.

# **Cahier Des Charges Fonctionnels**

* + 1. Contexte

Les ordinateurs embarqués fonctionnant sous le système d’exploitation Linux sont massivement présents dans les technologies modernes (transports, multimédia, téléphonie mobile, appareils photos …).

Les systèmes embarqués sous Linux disposent généralement de peu de RAM et utilisent fréquemment de la mémoire flash plutôt qu'un disque dur. Comme ils sont souvent dédiés à un nombre de tâches réduites sur une cible matérielle bien définie, ils utilisent plutôt des versions du noyau Linux optimisées pour des contextes précis.

L’objectif de ce projet est de réaliser une station météo avec un Raspberry Pi 3 B+ et un capteur BME280 qui permettra de mesurer la température, la pression atmosphérique et l’humidité.

* + 1. Produit

Les moyens mis en œuvre pour réaliser ce projet sont le produit et ses composants suivants :

* Le Raspberry PI 3.B + est une carte électronique comprenant un processeur intégré Quad-core ARM Cortex-A53 1.2 GHz (Broadcom BCM2837), RAM : 1024 Mo, GPU Dual Core VideoCore IV MultiMedia Co-Processor, lecteur de cartes Micro SD, ports : HDMI, 4x USB, RJ45, jack 3.5 mm, connecteurs pour APN et écran tactile, Wi-Fi b/g/n et Bluetooth 4.1.

Il est destiné à être piloté par un système Linux. Nous avons utilisé la distribution Raspbian qui est une version Debian avril 2019 avec un kernel version 4.14. Cette distribution Linux est installée sur une carte SD de 16 Go.

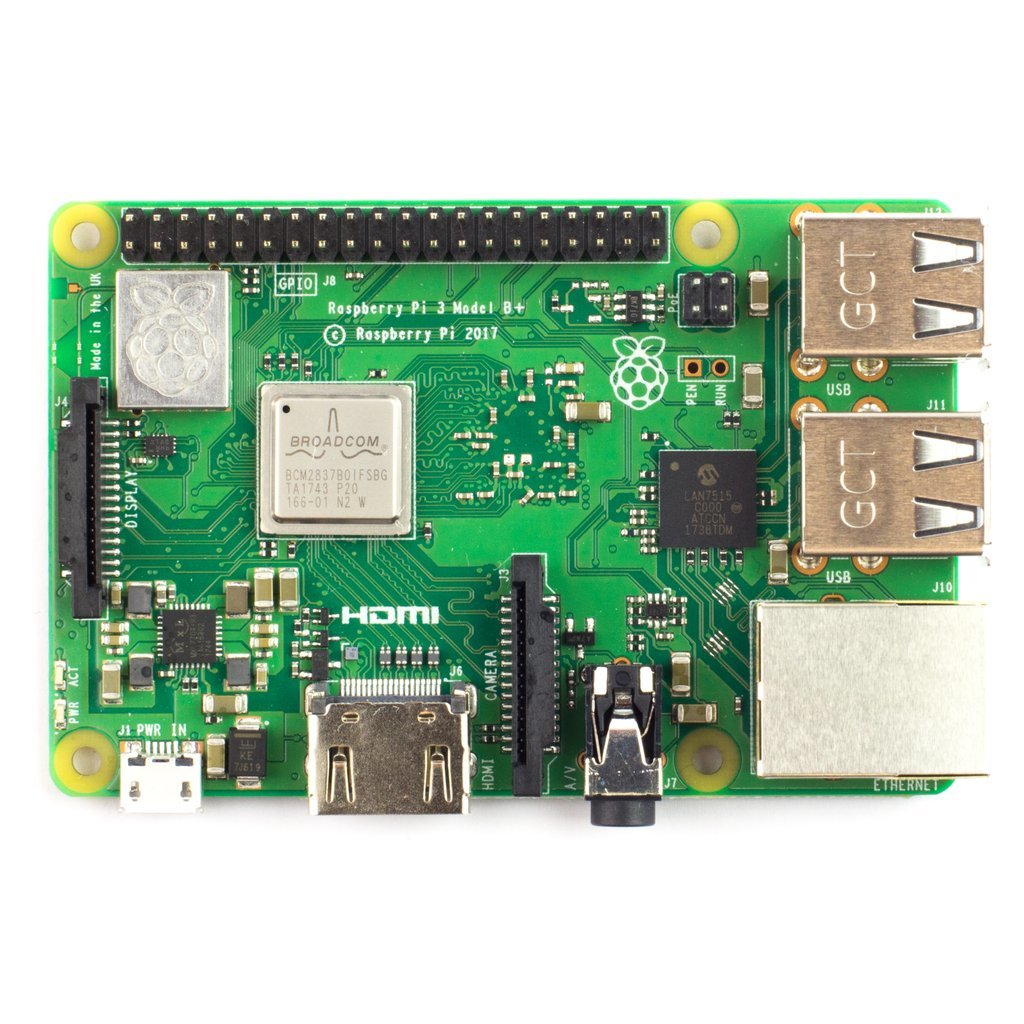


Figure 6 : Vue du dessus d’un Raspberry Pi 3 B+

* Le capteur BME280 est un circuit qui concentre un thermomètre, un mesureur de pression atmosphérique et un mesureur d’humidité. Il peut mesurer la température entre -40°C et 85°C avec une précision d’un degré, la pression entre 300hPa et 1100hPa à +/- 1hPa et l’humidité entre 0 et 100%.



Figure 7 : Vue du dessus d’un capteur BME280

Il peut se connecter en SPI ou en I2C, accepte une tension d’alimentation jusqu’à 3.6V et consomme très peu de courant. Les connections seront les suivantes pour l’I2C :

|  |  |
| --- | --- |
| **BME280** | **Raspberry** |
| VCC | 3.3V : Pin 1 ou 17 (surtout pas le 5V) |
| GND | Masse : Pin 6, 9, 14, 30, 34 ou 39 |
| SDI | SDA : Pin 3 |
| SCK | SCL : Pin 5 |
| CSB | 3.3V Pin 1 ou 17 |
| SDO | Masse : adresse 0x76 ou 3.3V : adresse 0x77 |

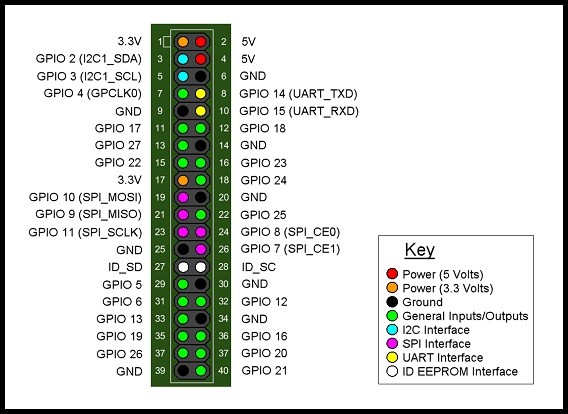
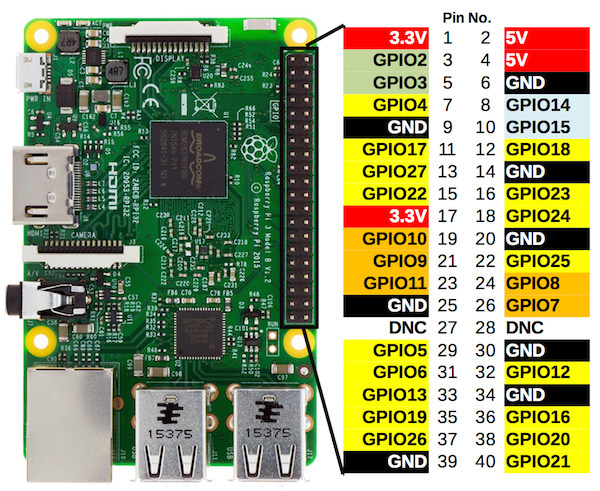


Figure 8 : Raspberry Pi 3 B+

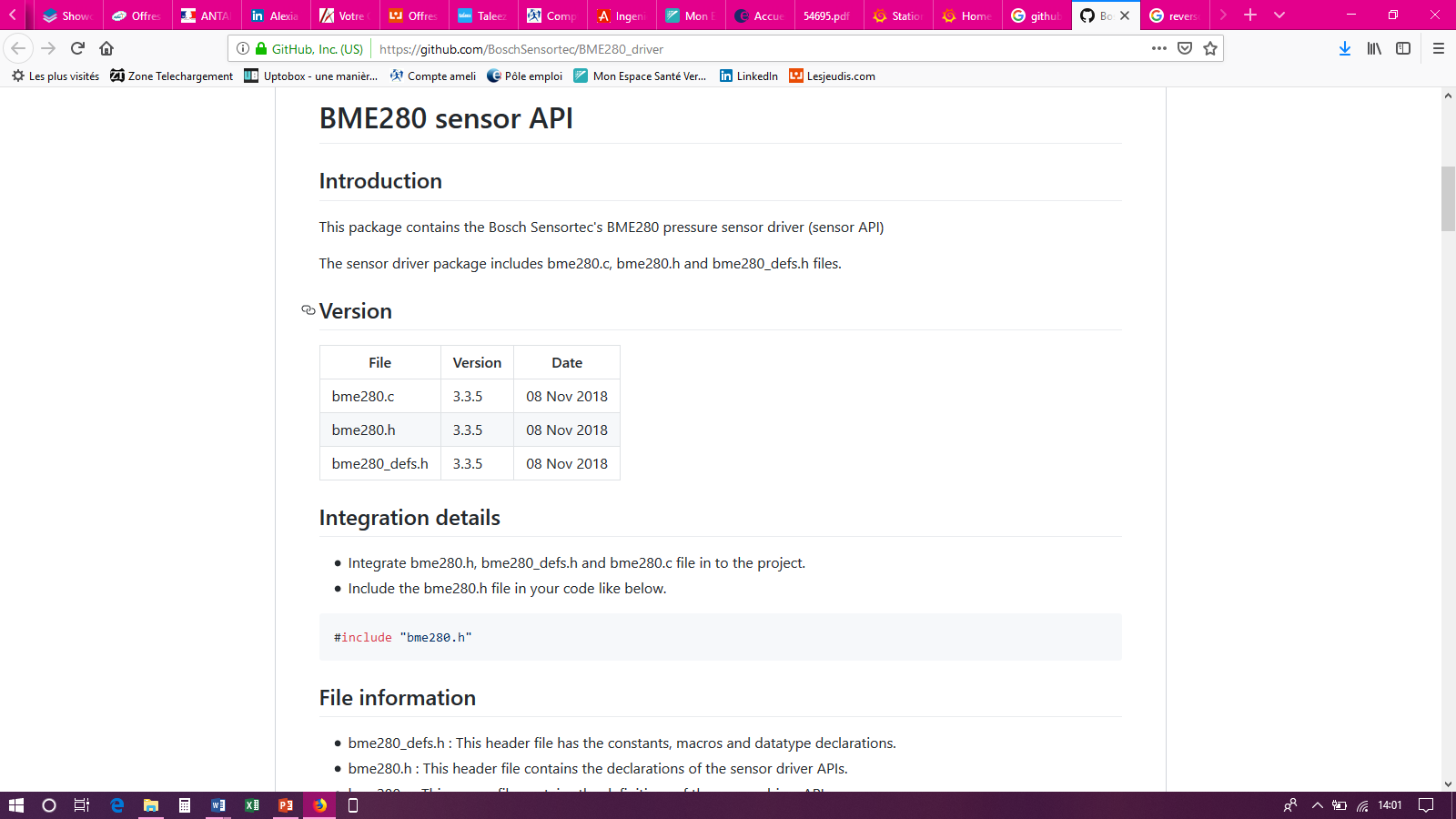
Les méthodes permettant d’avoir la température, la pression et l’humidité avec la compensation mémorisée dans le module sont très complexes.

En pratique, les librairies permettant la gestion du module comprendront une partie commune pour les fonctionnalités et une partie spécifique pour la lecture du module en I2C.

La partie commune sera une classe abstraite qui ne sera pas utilisable seule, elle devra obligatoirement être étendue par le module I2C qui contiendra les fonctions de lecture et d’écriture dans les registres du module ainsi que les constructeurs.

Le package de pilotes du capteur comprend les fichiers bme280.c, bme280.h et bme280\_defs.h.

**Version**



**Détails de l’intégration**

Intégrer les fichiers bme280.h, bme280\_defs.h et bme280.c dans le projet.

Inclure le fichier bme280.h dans votre code comme ci-dessous.

#include "bme280.h"

Voici les informations contenues dans les fichiers :

* bme280\_defs.h: ce fichier d'en-tête contient les déclarations de constantes, de macros et de types de données
* bme280.h: ce fichier d'en-tête contient les déclarations des API du pilote de capteur
* bme280.c: ce fichier source contient les définitions des API de pilote de capteur.
  + 1. Environnement logiciels

Avant de commencer l’implémentation de notre application, nous avons mis en place l’environnement de développement par l’installation des outils suivants :

* Qt/QML
* Grafana/ Prometheus
* Configuration de la Raspberry

Pour commencer, nous avons mis à jour la distribution Raspbian.

Nous avons installé les dépendance Qt et QML nécessaire à notre projet :

sudo apt-get install qt5-default qt5-qmake qtdeclarative5-dev qtdeclarative5-dev-tools qml-module-qtquick2 qml-module-qtquick-controls

Nous avons installé l’utilitaire de compilation Qmake :

sudo apt-get install Qmake

Nous avons utilisé Prometheus et Grafana pour la visualisation des données :

apt-get install -f prometheus grafana libcurl4-gnutls-dev zlib1g-dev

On lance les deux services au démarrage de la machine :

systemctl enable grafana-server

systemctl enable prometheus

* + 1. Fonctionnalités



Figure 9 : Modélisation des fonctionnalités

Pour le capteur **BME280** mis en place, il a deux fonctions :

* la fonction d’initialisation du capteur
* et la fonction de lecture de données.

La phase d’initialisation est essentielle pour pouvoir paramétrer le capteur, notamment le délai d’attente nécessaire entre deux mesures qui est de 70ms minimum pour pouvoir récupérer des données viables. Nous initialisons le capteur lors de la création de notre objet. La fonction de lecture permet de lire les données et de l’envoyer à notre objet.

Pour l’interface graphique, nous avons utilisé Qt Designer qui est une série d'outils intégrés dans Qt Creator permettant de créer graphiquement des interfaces graphiques d’une application. Qt Designer a l’avantage de permettre de modéliser des fenêtres graphiques sans compiler le projet et d’ajouter les éléments directement grâce à son interface. Mais plus encore, Qt Designer permet aussi de modifier les propriétés des widgets, d'utiliser des layouts et d'effectuer la connexion entre signaux et slots.



Légende :

* 1 : Partie propriété des widgets sélectionnés
* 2 : Fenêtre de modélisation de la fenêtre
* 3 : Éditeur de signaux/slots et éditeur d'actions
* 4 : Library : liste des widgets QML/ Qt Quick disponibles à placer dans notre fenêtre

Figure 10 : Fenêtre de Qt Designer

Pour les calculs des données, une fois les données métriques récupérées, nous avons dû modifier certaines métriques telles que la pression atmosphérique. Pour notre algorithme de prévisions, nous devons utiliser la pression atmosphérique au niveau de la mer. Hors le capteur nous renvoie la pression à l’altitude actuelle. Pour corriger la pression, nous utilisons la formule internationale du nivellement barométrique :



Figure 11 : Formule internationale du nivellement barométrique

avec :

* 1013,25 hPa la pression moyenne au niveau de la mer
* 0,0065 représente le gradient de température, ici 0,65 K pour 100m
* h l’altitude et 288,15 la température moyenne en Kelvin
* l’exposant est calculé par le rapport Mg/Ra avec M la masse molaire moyenne de l’atmosphère (0,028 96 kg mol-1), g l'accélération de la pesanteur (9,807 m s-2), R la constante universelle des gaz parfaits (8,314 J K-1 mol-1).

Cette formule permet le calcul de la pression à une certaine altitude, sans avoir besoin de connaître la température ou le gradient vertical de température.

Pour améliorer la pression de cette équation, nous l’avons modifié en utilisant la température fournie par le capteur à la place de la température moyenne de 15°C.

* + 1. Algorithme de ZAMBRETTI

Cet instrument date de 1915 est créé par deux météorologistes Negretti et Zambra (Zambretti) qui a été conçu à l’origine pour l’hémisphère Nord et le Royaume-Uni.

Le Zambretti Forecaster se compose de trois disques en plastique de rayon descendant, disposés concentriquement et fixés ensemble au moyen d’un rivet. Le grand disque extérieur A a une échelle courte calibrée dans la direction du vent. Le disque moyen B a un indice pour être aligné de manière appropriée avec l’échelle de direction du vent sur A et lui-même porte une échelle calibrée en unités de pression barométrique (pouces de mercure ou millibars.) Le disque intérieur C a un index réglable à la pression atmosphérique sur B et trois fenêtres qui permettent le choix d’une lecture basée sur la hausse ou la chute du baromètre en hiver ou en été. Par la fenêtre appropriée, une lettre alphabétique est visible (imprimée sur le disque A) et cette lettre est la clé de la prévision qui peut être lue sur le côté facial de l’instrument.

L’implémentation de cette algorithme se décompose en trois parties :

1. Le calcul de la tendance avec la pression
2. L’utilisation de de cette tendance et de la pression pour le calcul du numéro de Zambretti
3. L’utilisation de ce numéro dans la recherche de la prévision correspondante



Figure 12 : Zambretti forecaster

Il y a 26 numéros de Zambretti donc 26 prévisions possibles :

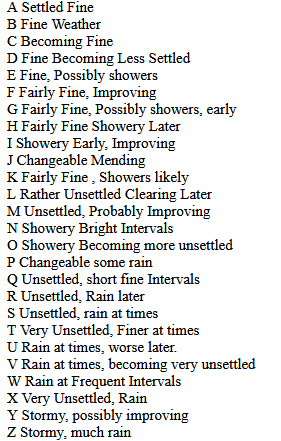


Figure 13 : Numéros de zambretti

Les prévisions vont du beau temps à l’orage avec beaucoup de pluie. Nous utilisons donc cet éventail de prévisions dans notre application pour la prévision.

Zambretti fournit trois tables selon les tendances qui sont calculées en comparant la pression calculée et une pression antérieure (une historisation sur 4 heures). Pour cette comparaison, nous avons utilisé une historisation des données sur 24 heures. Notre historisation est basée sur les moyennes des pressions. Une heure représente la moyenne des 60 dernières minutes, sachant que chaque minute est la moyenne des 60 dernières secondes. Nous avons obtenu ainsi une moyenne sur plusieurs heures. Cette historisation nous a permis de calculer la tendance.

Nous calculons la pression comme suit :

* < - 0,25 : tendance descendante
* - 0,25 <tendance < 0,25 : tendance stable
* > 0,25 : tendance montante.

En utilisant cette tendance, on peut déterminer la prévision en utilisant les tables correspondantes :

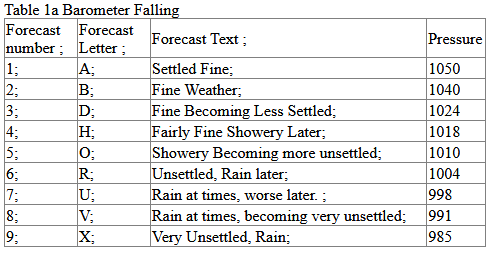


Figure 14 : Tendance descendante

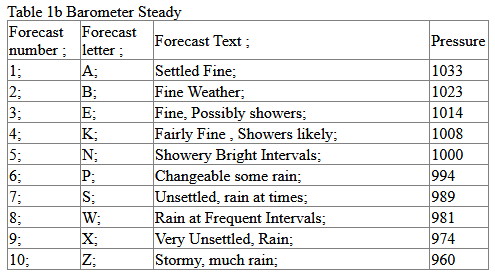


Figure 15 : Tendance stable

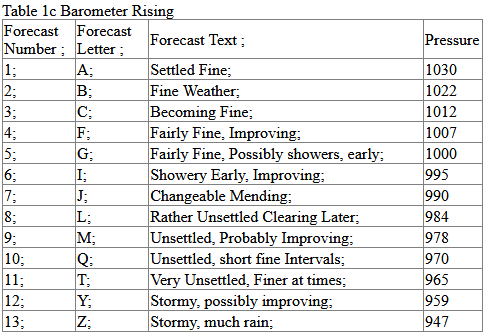


Figure 16 : Tendance montante

Une fois la tendance calculée, on peut choisir la table correspondante et ainsi déterminer la prévision.

On peut utiliser deux méthodes :

* 1ère méthode : en utilisant la pression correspondante de la table. Si on prend la table 1b Barometer Steady et que la pression est >= 1033 hPa alors la prévision est beau temps.
* 2ème méthode : en calculant le numéro de Zambretti déterminé par des fonctions de régression linéaire déduites des observations météorologiques. Pour le même exemple, pour avoir du beau temps, le numéro de Zambretti doit être 1.

Nous avons implémenté les deux méthodes pour nous permettre de pouvoir comparer et ainsi choisir la méthode la plus précise. Nous avons choisi la méthode avec les régressions linéaires qui est plus proche de la technique utilisée par les créateurs de l’algorithme et qui nous semble plus précise.

Pour notre implémentation, nous avons utilisé les régressions linéaires suivantes :

**Régression linéaire :**

Falling :

Steady :

Rising :

On utilise la pression actuelle pour le calcul du nombre de Zambretti. Une fois le numéro de Zambretti déterminé, on peut trouver la pression correspondante dans la table.

Ceci est l’algorithme généralisé de Zambretti mais les auteurs ont apporté des ajustements sur la mesure selon la saison. On effectue un ajustement pour les mesures d'hiver ou d'été. Une tendance en chute en hiver entraîne généralement une valeur Zambretti inférieure de 1 unité par rapport à une tendance en baisse en été. De même, une tendance en hausse en été améliore les perspectives de 1 unité par rapport à une tendance en hausse hivernale.

* + 1. Code

On peut résumer l’organisation de notre code avec le graphique des dépendances de notre projet :

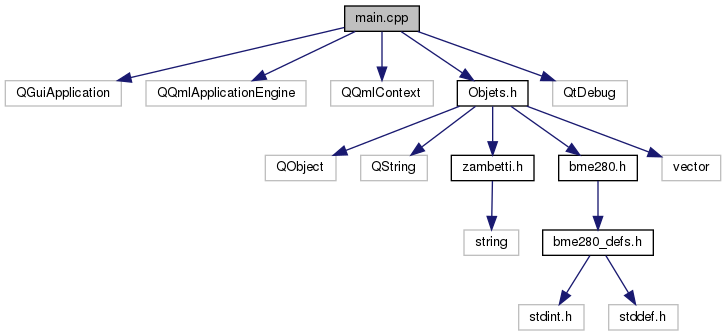


Figure 17 : Graphique des dépendances du main.ccp

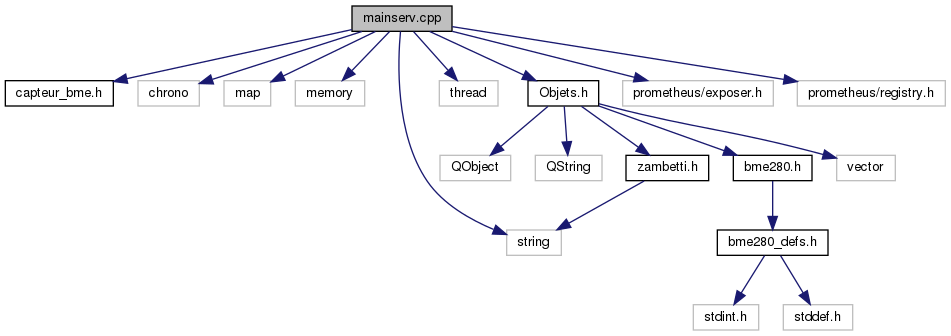


Figure 18 : Graphique des dépendances du mainserv.cpp

La seule différence entre notre « mainserv.cpp » et notre « main.cpp » est que pour le serveur il n’y a pas d’interface graphique, on ne fait que renvoyer les métriques à Prometheus.

Nous avons découpé notre code commun aux deux applications en une classe et plusieurs fonctions :

* La classe « Objets » qui interface le QML et nos fonctions en C++ :

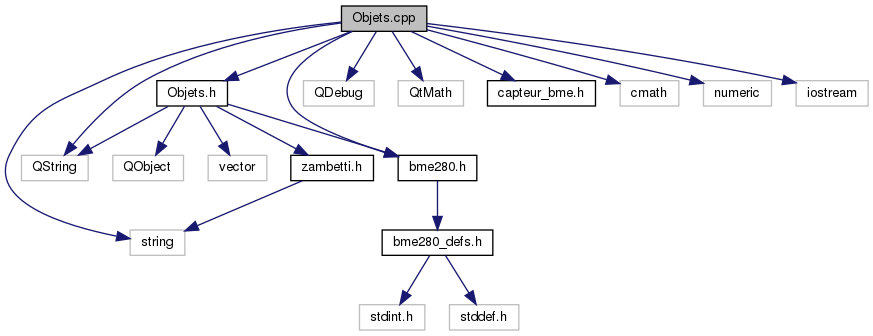


Figure 19 : Graphique des dépendances de la classe "objets"

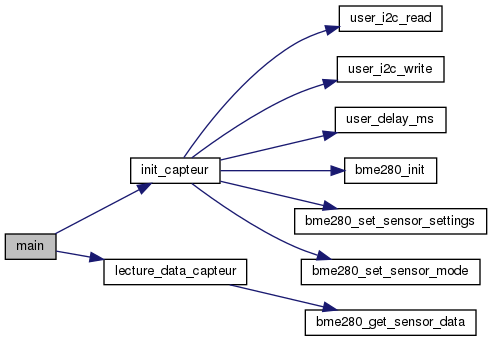


Figure 20 : Diagramme des fonctions

Dans notre « main », on initialise le capteur et on appelle la fonction « lecture\_data\_capteur » pour récupérer les métriques.

Le diagramme de classe « Objets » est le suivant:

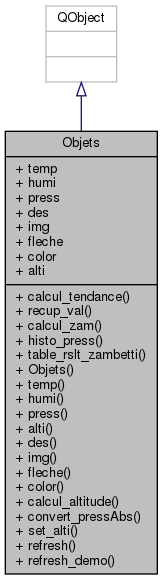


Figure 21 : Diagramme de classe "objets"

On initialise plusieurs variables telles que la température, l’humidité et la pression mais aussi l’altitude qui va nous servir pour le calcul de la pression atmosphérique au niveau de la mer. Les variables description, image, couleur et flèche vont nous servir pour l’affichage dans le QML.

Pour interfacer le QML et le code C++, nous avons utilisé la macro « Q\_Property ». Cette macro est utilisée pour déclarer des propriétés dans des classes qui héritent de « QObject ». Les propriétés se comportent comme des membres de données de classe. Dans notre « Main », nous avons passé l’objet à QML qui va accéder à notre membre de la classe et permet ainsi de modifier ou envoyer les données de la classe.

Q\_PROPERTY (type name | (READ getFunction | setFunction | NOTIFY notifySignal)

* La fonction « capteur\_bme » rassemble les fonctions qui interfacent notre code et le capteur :

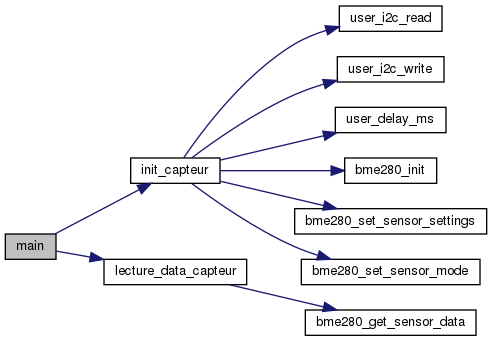


Figure 22 : Diagramme des fonctions

Les données sont renvoyées par le capteur dans une structure du type :



Figure 23 : Structure du "bme280\_data"

* La classe « Zambretti » contient le code de prévision :

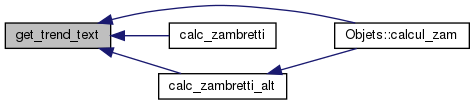


Figure 24 : Graphique des dépendances des fonctions

La fonction « cal\_zambretti » est la fonction calculant la prévision via la pression et « cal\_zambretti\_alt » calcule les prévisions via les régressions linéaires.

On voit que les deux fonctions appellent la fonction « get\_trend\_text » qui retourne la tendance.

* + 1. Interface de visualisation des données

Pour la visualisation de l’historique des données métriques, nous avons décidé d’utiliser Prometheus et Grafana ainsi que la librairie « promotheus-cpp ».

Prometheus est un projet logiciel open-source écrit en Go utilisé pour enregistrer des métriques en temps réel dans une base de données chronologiques construite à l'aide d'un modèle d'extraction HTTP, avec des requêtes flexibles et des alertes en temps réel.

Grafana est un logiciel libre sous licence Apache 2.0 qui permet la visualisation et la mise en forme de données métriques. Il permet de réaliser des tableaux de bord et des graphiques depuis plusieurs sources dont des bases de données de série temporelle comme Graphite, InfluxDB et OpenTSDB.

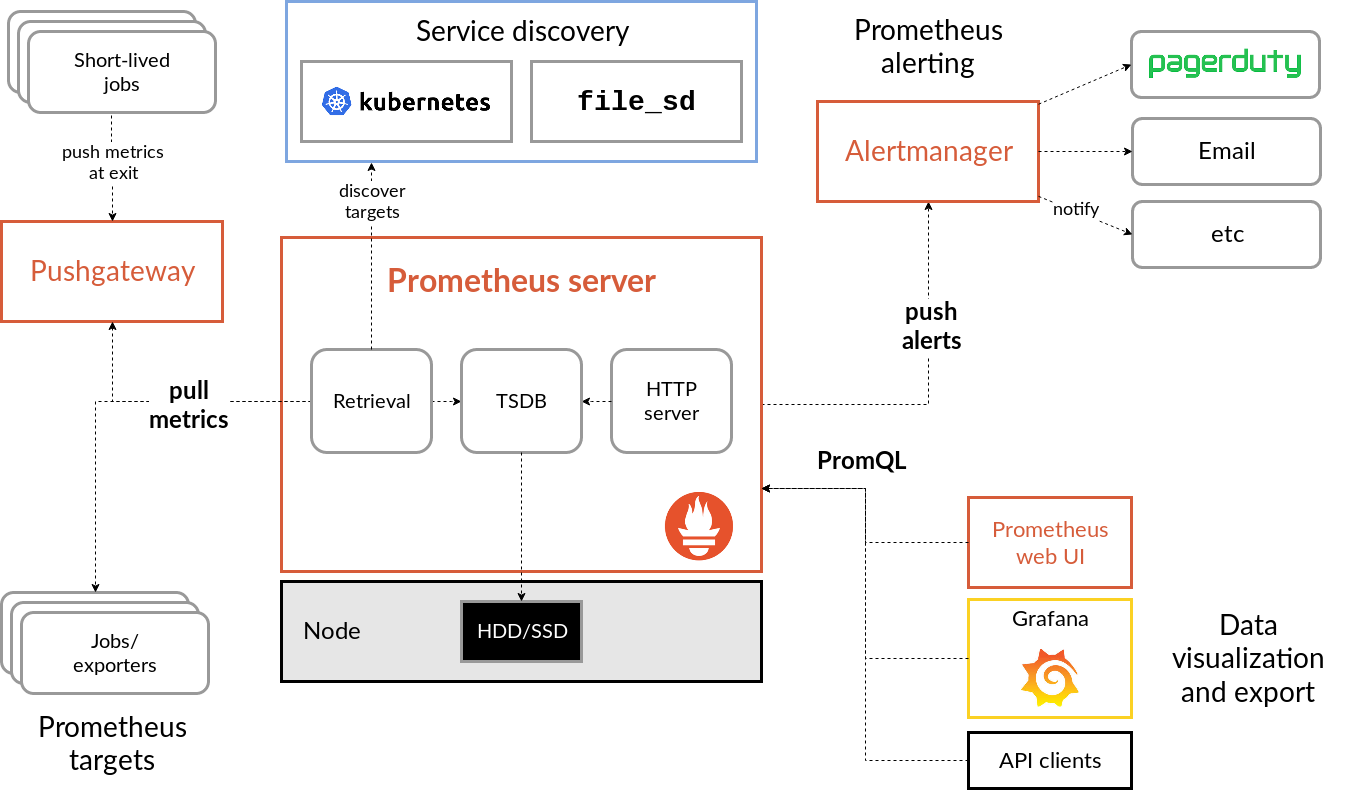


Figure 25 : Graphique d'utilisation entre prometheus et grafana

Nous avons mis en place Grafana pour l’affichage des données en temps réel. Les métriques sont envoyées par Prometheus qui les récupère via le port 8080 de notre serveur utilisant la libraire « prometheus-cpp ».

Après avoir installé Prometheus et Grafana, il a fallu les configurer pour connecter les deux interfaces. Dans un premier temps, nous avons utilisé une librairie externe pour nous permettre d’envoyer nos métriques depuis notre code vers Prometheus. Pour cela, nous avons utilisé « prometheus-cpp ». Cette bibliothèque vise à activer les Metrics-Driven Development pour le C ++. Il implémente le modèle de données Prometheus qui permet de collecter et exposer des métriques.

Nous avons implémenté une fonction qui récupère les données métriques et qui créée un serveur http sur le port 8080 pour l’envoi des données formatées par la librairie, les données étant attachées à un label qui sera utile pour Grafana. Ensuite, nous avons configuré Prometheus pour qu’il récupère les données sur ces ports :

- job\_name: ServPro

# If prometheus-node-exporter is installed, grab stats about the local

# machine by default.

static\_configs:

- targets: ['localhost:8080']

Ensuite, une fois que Prometheus a récupéré nos données, on a configuré Grafana pour qu’il récupère nos données en indiquant le label dans le Querie et ainsi Grafana peut afficher en temps réel nos données.

* + 1. Qt Designer de Qt Creator

Pour la partie compilation de notre projet, nous avons utilisé « qmake » qui est un utilitaire qui automatise la génération de « makefiles ».

« qmake » a été créé par Trolltech (maintenant la société Qt). Distribué et intégré au framework d’application Qt, il automatise la création des sources « moc » (compilateur de méta-objets) et « rcc » (compilateur de ressources) qui sont utilisées dans le système de méta-objets de Qt et dans l’intégration de ressources binaires (par exemple, des images).

L'outil « qmake » permet de simplifier le processus de création de projets de développement sur différentes plates-formes. Il automatise la génération de « makefiles » de manière à ne nécessiter que quelques lignes d’informations pour créer chaque « makefile ».

# **Retour d’expérience**

* + 1. Suivi des ressources

En terme de ressources, la prévision a été respectée. L’ambiance de travail agréable a favorisé le bon avancement et réalisation du projet.

* + 1. Suivi de la planification du projet
    2. Avantages/inconvénients de l’environnement technique

Au cours du projet, le pilotage va permettre de comparer le réalisé avec le prévisionnel, éventuellement de réviser les plannings et les charges.

Quel que soit l’envergure du projet, chaque responsable ne bénéficie pas du recul et du temps suffisants pour mesurer l’impact de ses décisions, le pilotage permet d’assurer un suivi fiable du projet grâce à l’obtention d’une vue d’ensemble sur le projet, de mesurer précisément l’avancement du projet, de valider les dates jalons et de prendre les bonnes décisions en cas de difficulté.

#### La démarche qualité au cours du projet

La démarche qualité doit s’intégrer à toutes les phases du projet.

**Phases d’études préalables et de définition fonctionnelle du besoin**

Lors des phases d’étude préalable et de définition fonctionnelle des besoins, la qualité se caractérise par la capacité des études produites à adresser les véritables objectifs du projet et de complètement satisfaire les attentes associées. Lors de ces phases, les contrôles qualité reposent d’une part, sur la vérification du respect des principes de la méthodologie mise en œuvre et d’autre part, sur des revues visant à garantir l’adéquation et la cohérence des solutions proposées avec les attentes entre les différentes phases (schéma directeur, puis analyse préalable puis définition fonctionnelle du besoin). Le processus qualité s’attachera également à s’assurer du bon fonctionnement du cycle de validation pour vérifier que tous les principes et solutions proposés ont fait l’objet d’une validation ad hoc.

**Phases de réalisation**

Le contrôle de la qualité porte d’une part, sur la vérification de la méthodologie mise en œuvre et d’autre part, sur des revues de code permettant de garantir les performances et la maintenabilité des applications développées.

**Phases de test**

Les phases de test permettent de vérifier d’une part, le bon fonctionnement intrinsèque des applications livrées et d’autre part, l’adéquation entre les fonctions réalisées par ces applicatifs et les fonctions spécifiées dans les dossiers de définition du besoin. En conséquence, lors de ces phases, le contrôle de qualité va consister à vérifier le bon fonctionnement de ce processus.

**Contrôle**

Contrôle de l’élaboration des fiches de tests (cycle de test, cas de test, description des jeux de test, description des résultats attendus),

TABLE DES ILLUSTRATIONS

[Figure 1 : WBS (Work Breakdown Structure) 5](#_Toc7536248)

[Figure 2 : Diagramme de Gantt 6](#_Toc7536249)

[Figure 3 : Story Mapping des user stories 7](#_Toc7536250)

[Figure 4 : Kanban (GitHub) 8](#_Toc7536251)

[Figure 5 : Tableau des indicateurs de pilotage du projet 9](#_Toc7536252)

[Figure 6 : Vue du dessus d’un Raspberry Pi 3 B+ 10](#_Toc7536253)

[Figure 7 : Vue du dessus d’un capteur BME280 11](#_Toc7536254)

[Figure 8 : Raspberry Pi 3 B+ 12](#_Toc7536255)

[Figure 9 : Modélisation des fonctionnalités 14](#_Toc7536256)

[Figure 10 : Fenêtre de Qt Designer 15](#_Toc7536257)

[Figure 11 : Formule internationale du nivellement barométrique 15](#_Toc7536258)

[Figure 12 : Zambretti forecaster 17](#_Toc7536259)

[Figure 13 : Numéros de zambretti 17](#_Toc7536260)

[Figure 14 : Tendance descendante 18](#_Toc7536261)

[Figure 15 : Tendance stable 19](#_Toc7536262)

[Figure 16 : Tendance montante 20](#_Toc7536263)

[Figure 17 : Graphique des dépendances du main.ccp 21](#_Toc7536264)

[Figure 18 : Graphique des dépendances du mainserv.cpp 21](#_Toc7536265)

[Figure 19 : Graphique des dépendances de la classe "objets" 22](#_Toc7536266)

[Figure 20 : Diagramme des fonctions 22](#_Toc7536267)

[Figure 21 : Diagramme de classe "objets" 23](#_Toc7536268)

[Figure 22 : Diagramme des fonctions 24](#_Toc7536269)

[Figure 23 : Structure du "bme280\_data" 24](#_Toc7536270)

[Figure 24 : Graphique des dépendances des fonctions 24](#_Toc7536271)

[Figure 25 : Graphique d'utilisation entre prometheus et grafana 25](#_Toc7536272)