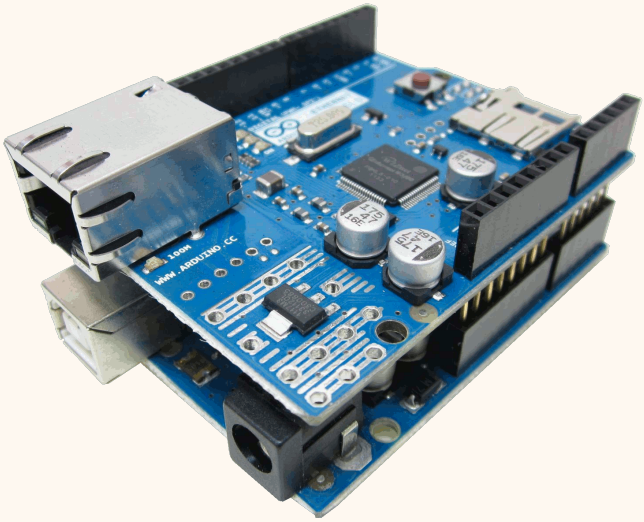
Travaux personnels encadrés

du

Lycée François Rabelais de Chinon
Baccalauréat scientifique, option sciences de l'ingénieur



Thème du TPE : « Avancée scientifique et réalisations techniques »

Comment relever des données météorologiques et les transmettre sur un espace accessible par tous ?

ARDUINO
OPEN-SOURCE
COMMUNITY

Soutenu le 25 mars 2015

Encadré par
M. Guibert : professeur de sciences de l'ingénieur
M. Labourdette : professeur de sciences de l'ingénieur

Présenté par
Téofil Adamski
Remi Bruyère
Sammy Plat
Clément Bidault

Résumé & remerciements

Depuis quelques années maintenant, nous assistons à un réel développement de la domotique. De nombreux produits sont disponibles pour l'utilisateur afin de rendre sa maison plus « intelligente ».

La *domotique* est l'ensemble des techniques utilisés pour nous rendre le quotidien meilleur. Ce domaine utilise quasiment tout les techniques existantes : nous pouvons trouver aussi bien de l'électronique que de la physique ou encore de l'informatique. Ces dispositifs perment, entre autre, de contrôler notre maison depuis l'extérieur : fermeture et ouverture des volets, déclenchement du chauffage, sécuriser notre habitat, ...Mais il peut également servir à augmenter notre comfort et embellir notre quotidien comme les home-cinéma, la musique multi-pièce, ...

Aujourd'hui, un des rôles très important de la domotique est de réduire les coûts financiers et les impacts sur l'environnement. Ainsi, des objets technologiques comme des thermostats intelligents ou des capteurs de température se développent.

Par la biais de nos travaux personnels encadrés, nous avons décidés de nous diriger vers ce domaine. Tout au long de ce dossier, nous nous questionnerons sur

> *Comment relever des données météorologiques et les transmettre sur un espace accessible par tous ?*

Ainsi, pour combler cette problèmatique, nous nous sommes penchés sur le développement et la fabrication d'un releveur de données météorologiques qui est capable de transmettre ses données sur Internet. Notre dispositif permettra alors de contrôler la température et la pression d'une pièce dans le but de réguler le chauffage. Celui-ci peut aussi faire office de mini station météorologique qui deliverai, en temps réel sur le Web, la température et la pression.

Notre dossier s'organisera en plusieurs parties. Dans un premier temps, nous allons vous présenter notre projet avec son analyse fonctionnelle. Ensuite, nous expliquerons en détaiils notre démarche pour créer cet objet avec la partie électronique et site Web. Enfin, nous présenterons les résultats que nous aurons obtenus. À la fin de ce dossier, vous pourrez retrouver en annexe les codes complets utilisés.

Nous pouvons remercier nos deux professeurs de sciences de l'ingénieur M. Guibert et M. Labourdette qui nous ont conseillés tout au long de nos recherches.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Présentation du projet

1.1 Idée de départ

L'idée directrice de notre TPE est la suivante : la capture de donnée météorologique et le transfère de ces données sur un site Web accessible à tous. Nous voulions créer un système qui permettent à l'utilsateur de connaître la température ainsi que la pression d'un environnement à distance. Ce type de produit existe déjà sur le marché de la domotique mais ils stockent les données sur une carte SD (comme ceux de la marque Testo, qui coûte plus d'une centaine d'euros).

FIGURE 1.1 – Le testo 175 T1 (prix : 150€ TTC)

Nous avons voulu innover en stockant sur une base de données en ligne. De plus, notre projet relève non seulement la température mais aussi la pression. Nous avons aussi essayer de réduire les coûts

Nous avons donc penser à utiliser un microcontrôleur relier à un capteur qui enverra les données vers une base de donnée. Ensuite, un site Web ira chercher les données sur cette base. Nous présenterons rapidement les éléments citées dans la synoptique et aprofondirons les sujets dans les différentes parties du dossier

1.2 Analyse fonctionnelle

Nous nous somme donc penché sur les caractéristique de nous devions mettre en œuvre pour arriver à mener à bien notre projet.

1.2.1 Méthode APTE

Dans un premier lieu, nous nous somme demandé pourquoi, comment et pour qui cela pourrait servir. Nous avons donc utiliser la méthode dite « méthode APTE »qui à pour but d'analyser les besoins et les problème que l'on peut rencontrer lors de notre projet. Au début nous avons crée le diagramme bête à corne suivant.

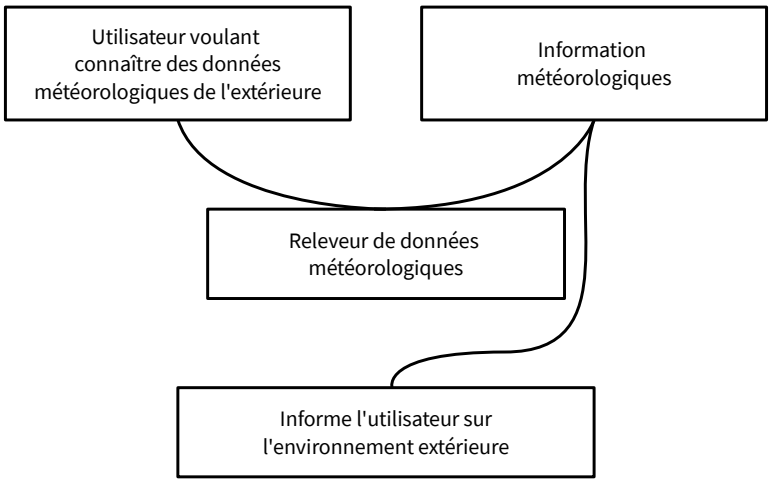


FIGURE 1.2 – Diagramme bête à corne de notre projet

Nous pouvons donc penser que les données météorologique que nous recueillerons servirons à l'utilisateur. Celui-ci pourra alors être informé de la température et de la pression atmosphérique d'une pièce ou extérieure en fonction de l'utilisation faite par celui qui s'en servira.

1.2.2 Diagramme des interacteurs

Suite à cela, nous devons déterminer toutes les fonctions que réalisera notre projet et ses interactions avec l'extérieure. Ainsi, nous avons réaliser le diagramme des interacteurs ou diagramme « pieuvre »suivant.

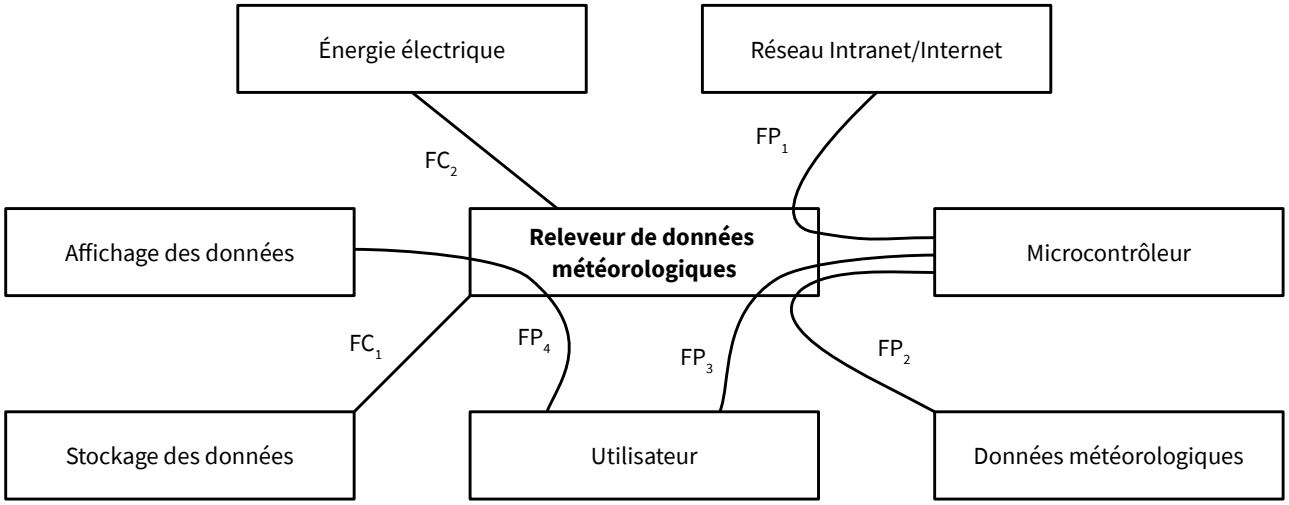


FIGURE 1.3 – Diagramme des interacteurs

Ainsi, sept interacteurs ont été trouvé et il en découle quatre fonctions principales et deux fonctions contraintes.

1.2.3 Cahier des charges fonctionnel

Enfin, pour terminer notre analyse fonctionnelle, nous avons regrouper toute nos caractéristiques et nos fonctions pour former un cahier de charges fonctionnel (voir tableau 1.1).

Nous avons donc déterminer, en fonction de nos attente, les critères pour pouvoir réaliser le projet. Le niveau correspond à la valeur du critère, par exemple, le capteur de température doit être au moins précis à 1 °C. Ensuite, plus la fléxibilité d'un niveau est faible, moins on peut le modifier. On peut donc voir, par exemple, que pour la température, on autorise une flexibilité de 2 ce qui veut dire que

	Fonctions	Critères	Niveaux	Flexibilité
FP_1	Doit pouvoir communiquer des informations sur le réseau Intranet/Internet	Prise RJ45 et accès au réseau de lycée (admission d'adresse IP) ou réseau local	aucun	aucun
FP_2	Doit pouvoir capter des données météorologiques	Capteur de température Capteurs de pression	Précision à 1 °C Précision à 10 hPa	F_2
FP_3	Doit être facile d'installation et d'utilisation, être sécurisé pour l'utilisateur	Boîtier de protection Indicateur de mise en marche Raccord au réseau facile	Étanche Voyant (LED) Câble et prise RJ45 (connexion filaire)	F_0 F_1 F_0
FP_4	Données doivent être visible par l'utilisateur	Site Web avec langage de programmation		
FC_1	Doit pouvoir transférer les données sur un moyen de stockage	Base de données	Espace inférieur à 10 Mo	F_2
FC_2	Doit utiliser l'électricité pour s'alimenter en énergie	Électricité courante suivie transformateur courant continue	Tension de 5 V à 12 V	F_0

TABLE 1.1 – Le cahier des charges fonctionnel

l'on autorise un précision un peu plus petite. Autre exemple, pour l'esthétique, la fléxibilité est au maximum car cela n'empechera pas les performances ou les fonctionnalités de l'objet.

1.2.4 Synoptique et chaîne d'information

Nous avons commencé à rentrer dans le détail de notre TPE en choisissant notre capteur, qui sera un capteur de température et de pression BMP180 et notre microcontrôleur qui sera par conséquent un Arduino Uno pour pouvoir contrôler le capteur. En plus de ce microcontrôleur, nous mettrons un Arduino Ethernet V2.0 qui nous permettra d'envoyer les données sur la base de donnée qui ce situera sur un serveur qui hébergera aussi le site Web. Un *switch* sera installer en l'Arduino et l'ordinateur pour pouvoir brancher d'un deuxième ordinateur en tant que client (l'utilisateur final). Pour notre base de donnée nous allons utiliser MySQL et pour notre serveur Web nous allons utilisé Apache. Tous ces logiciels sont présentées dans le chapitre [référence nécessaire].

Nous avons donc représenté les interactions entre les différents composants sur un synoptique (voir figure 1.4).

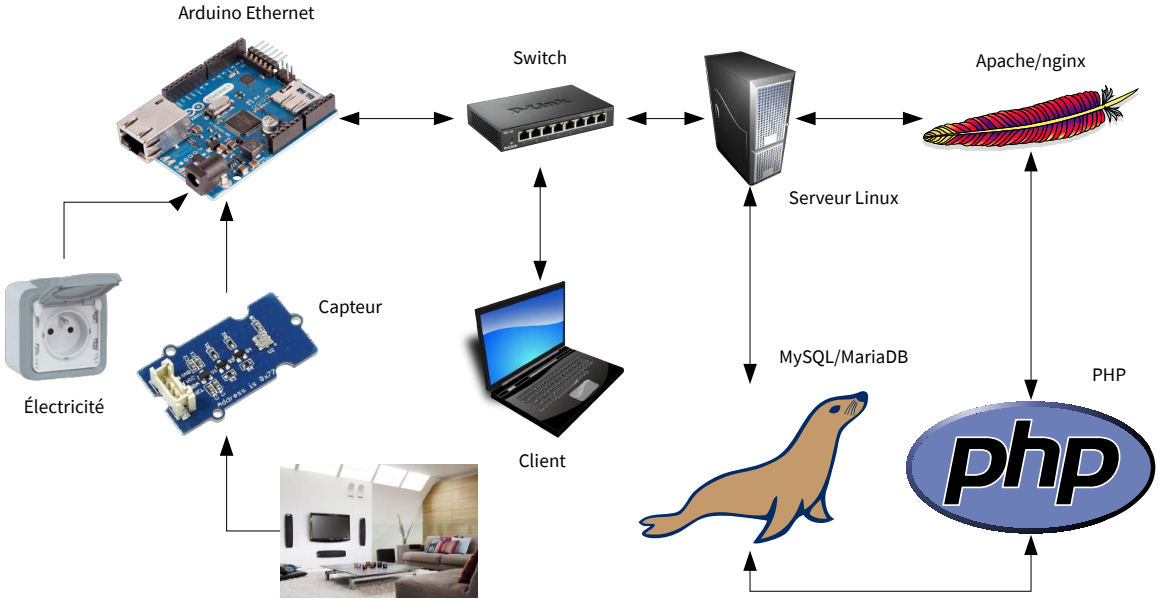


FIGURE 1.4 – Synoptique récapitulant tous les composants

Enfin, nous avons déterminer notre chaîne d'information (voir figure 1.5), de l'acquisition de la température à l'insertion de la base de données. La partie site Web ne figure pas car elle est indépendente de cette partie bien qu'il communique avec la base.

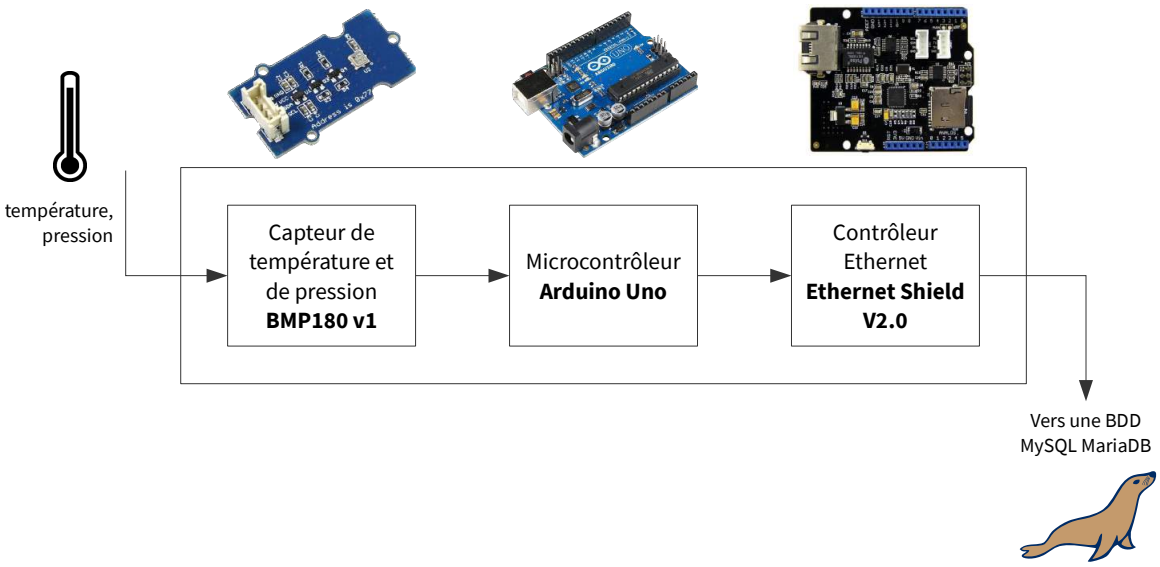


FIGURE 1.5 – Chaîne d'information de la partie éléctronique

Chapitre 2

Partie électronique

Dans cette partie, nous nous interréserons à la partie électronique de notre projet. Elle s'acompagnera d'explications théoriques sur les différents composants que nous utilisons sur notre réalisation.

2.1 Une plateforme pour reçevoir les données

Comme vu dans la partie précédente, notre projet consiste à capter, au départ, des données météorologiques. Pour faire cela, il nous faut un plateforme qui puisse communiquer avec des capteurs.

Naturellement, nous nous sommes tournés vers un *Arduino*.

2.1.1 Qu'est-qu'un Arduino ?

Un Arduino est une plateforme possédant des entrées et des sorties. Il est organisé autour d'un microcontrôleur Atmel. Celui-ci est capable des contrôler différents récepteur comme une LED, un moteur ou encore un autre microcontrôleur. Il est capable également de recevoir et traiter des données d'un capteur. C'est dans ce cas que nous allons l'utiliser.

Les entrées et les sorties sont commandés à partie d'un code compilé sur un ordinateur puis téléversé sur un microcontrôleur.

Il existe une multitude de carte Arduino. Pour nos besoins, nous allons utiliser un *Arduino Uno*. C'est une carte qui comporte 13 entrées/sorties numériques et 6 analogiques.

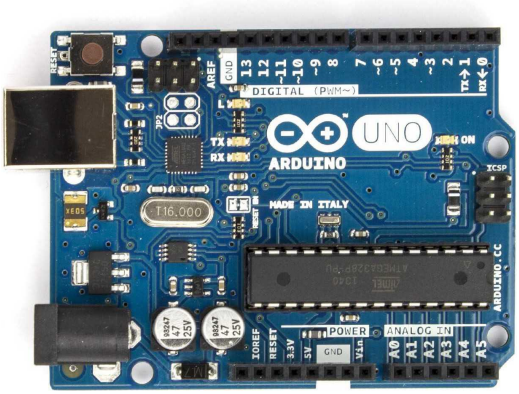


FIGURE 2.1 – Un Arduino Uno R3

2.1.2 Programmation d'un Arduino

Pour programmer un Arduino, il faut utiliser un ordinateur avec l'IDE [1] du constructeur. Après l'avoir installé et configuré pour qu'il programme la bonne carte, nous pouvons y insérer du code.

1. Disponible sur http://arduino.cc/en/Main/Software.

La langage pour programmer un Arduino est très proche du C++, il s'agit en fait d'une surcouche. De ce fait, ce langage intègre la programmation orientée objet qui est très utile quand on veut développer : cela permet d'écrire du code plus compréhensible. Nous avons appris à nous en servir grâce au tutoriel [1] sur Zeste de Savoir.

La structure d'un code Arduino se divise essentiellement en 2 parties : la première pour initialiser les différents composants, la seconde contient le code qui doit s'exécuter en boucle. Ainsi, pour faire clignoter une LED raccordée à la sortie 13, on peut téléverser le code suivant à l'Arduino.

```
#define LED 13 // On stocke l'entrée dans un constante.

/* Phase d'initialisation */
void setup()
{
  pinMode(LED, OUTPUT); // On dit que la LED est une sortie.
}

/* Boucle infinie */
void loop()
{
  digitalWrite(LED, HIGH); // On éteint la LED.
  delay(1000);             // On attend 1000 ms = 1 s.
  digitalWrite(LED, LOW);  // On allume la LED.
  delay(1000);             // On attend 1 s.
}
```

Comme vous pouvez le voir, le code est relativement simple. C'est pourquoi les plateformes Arduino sont conseillés pour les débutants (comme nous) en électronique. Dans ce code, il est possible d'ajouter des conditions avec `if .. elseif ... else ...` ou des boucles `for` ou `while` pour faire des exemples plus complets.

2.2 Le choix du capteur

Afin de relever nos données météorologiques, il nous faut un capteur.

2.2.1 Caractéristiques d'un capteur

Un capteur est un composant permettant de traduire une grandeur physique en un courant électrique, hydrolique, ... Les capteurs sont utilisés quasiment partout et il en existe beaucoup : certains peuvent capteur le courant, le vent, la température ou encore la lumière. Il s'agit en fait d'une interface entre le monde extérieur et un circuit électronique.

Type de sortie Tout d'abord, il existe principalement deux types des capteurs qui se classent en fonction du leur sortie. Les capteurs analogiques sortent une tension ou une intensité qui se traduit par la valeur que capte ce composant.

Nous pouvons trouver également des capteurs numériques qui ne sortent pas un courant mais des états logiques : 0 ou 1. Nous allons nous tourner vers ce type de capteur

Caractéristiques Pour distinguer différents capteurs, il existe des caractéristiques :

- l'étendue : c'est l'écart entre la plus grande et la plus petit valeur mesurable,
- la résolution : c'est la plus petite variation que la capteur est capable de mesurées. Par exemple, un capteur de température capable de distinguer au maximum 20,0 °C de 20,1 °C à une résolution de 0,1 °C ;
- la sensibilité : c'est la rapport entre la variation du signal d'entrée et la variation du signal du sortie ;
- l'exactitude : elle indique le pourcentage d'erreurs commises ;
- la justesse : c'est l'aptitude à donner un valeur juste, c'est l'écart entre le résultat moyen et la valeur vraie ;

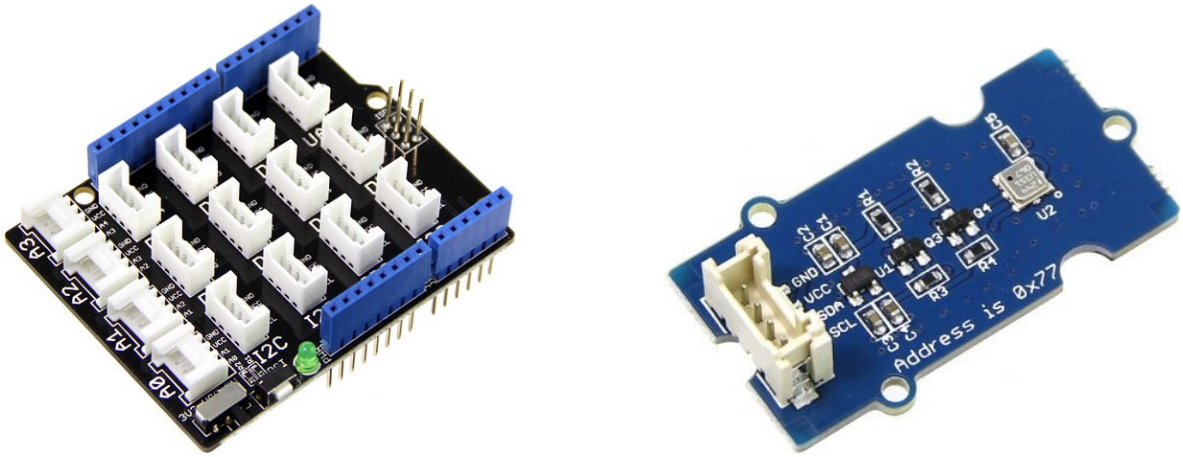


FIGURE 2.2 – Le Base Shield V2 et le BMP180

— la fidélité : elle définit la dispersion des valeurs relevées.

En fonction de ces différentes caractéristiques, nous pouvons choisir un capteur.

2.2.2 Capteur utilisé

Comme vu dans le premier chapitre, notre capteur se doit de capter le température ainsi que la pression. Au lycée, nos deux professeurs nous ont orientées vers le capteur BMP180 en particulier celui de la marque Seeed Studio. Celui-ci a une particularité, il a besoin d'un *shield* pour pouvoir être branché à un Arduino.

Ce capteur se relie alors à une borne `I2C` et le *shield* se place sur l'Arduino. L'avantage de celui-ci est de permettre un cablage facile sans rajout de composants électroniques.

D'après le wiki de Seeed Studio [3], l'étendue de mesure de ce capteur va de $-40\,°C$ à $85\,°C$ et il est précis à $2\,°C$ près : cela ne satisfait pas totalement notre cahier des charges initial mais suffira pour notre utilisation.

Protocole I²C

Le capteur BMP180 se pilote à l'aide du protocole I²C. Ce protocole est un bus série crée par Philips. Les échanges d'informations se font toujours entre un seul maître et un seul esclave (ici, l'Arduino et le capteur respectivement). La connexion des composants I²C se fait par l'intermédiaire de trois fils :

— la SDA (*Serial Data Line*) : c'est la ligne de données ;

— la SCL (*Serial Clock Line*) : c'est la ligne d'horloge ;

— la masse.

Pour un Arduino Uno, le fils pour la SDA doit se relier au *pin* A4 et la SCL au *pin* A5 mais cela nous est inutile grâce au *shield*.

Sa programmation

Pour communiquer avec le capteur, il va falloir programmer l'Arduino. Pour cela, le constructeur du capteur nous met à disposition une bibliothéque [2] qui va nous faciliter le travail. Il suffit alors de décompresser l'archive dans le dossier des bibliothéques Arduino.

Dans notre code, il faut inclure le fichier `Barometer.h` qui correspond à l'en-tête de la bibliothéque. Ensuite, il faut déclarer le capteur puis relever ses données. Voici un exemple de code qui permet de capter la température et la pression.

2. Disponible sur leur dépôt GitHub : https://github.com/Seeed-Studio/Grove_Barometer_Sensor.

```ino
#include <Barometer.h> // On inclus la bibliothèque du constructeur.
#include <Wire.h>

float temperature(0), pression(0); // On déclare les variables.

Barometer capteur; // On déclare le capteur.

void setup()
{
  Serial.begin(9600); // On initialise le communication série
  capteur.init();     // et le capteur.
}

void loop()
{
  /* On stocke la température et la pression. */
  temperature = capteur.bmp085GetTemperature(capteur.bmp085ReadUT());
  pression = capteur.bmp085GetPressure(capteur.bmp085ReadUP());

  /* On affiche les données. */
  Serial.print("Température : ");
  Serial.print(temperature);
  Serial.println(" °C");
  Serial.print("Pression : ");
  Serial.print(pression);
  Serial.println(" Pa");
  Serial.println();

  delay(1000);
}
```

Comme nous allons envoyer les données sur le serveur tous les quartes d'heures, nous préférons prendre une température et une pression toutes les 90 s et faire la moyenne de toutes les valeurs reçues au bout de 15 min. Cela évitera les valeurs extrèmes et augmentera la fidélité des relevés. Pour se faire, nous utilisons un boucle `for` allant de 0 à 9 (ce qui fait 10 relevés car $10 \times 90\,\text{s} = 15\,\text{min}$). À l'intérieur, nous faisons la somme des relevés. Puis nous divisons par 10 cette somme. Voici un exemple uniquement pour la température.

```ino
// ...
float moyenneTemperature(0);

for (int i(0) ; i < 10 ; i++)
{
  moyenneTemperature += capteur.bmp085GetTemperature(capteur.bmp085ReadUT());
  delay(90000);
}

moyenneTemperature /= NB_RELEVES;
// ...
```

2.3 Connexion de l'Arduino au réseau Ethernet

Un des côtés essentiels de notre projet est de pouvoir communiquer les données sur Internet. Pour faire communiquer l'Arduino à Internet, il est possible de lui ajouter un *shield* comportant une carte réseau. Pour notre projet, nous utilisons le shield Ethernet V2.0 de Seeed Studio.

Ce *shield* est composé d'une puce W5200 qui permet de lancée des requêtes TCP/UDP sur un réseau Ethernet. Il faut bien évidement le relier à un *switch* ou à une *box* par un câble Ethernet.

Pour programmer ce *shield*, il faut aussi utiliser la bibliothèque[3] proposée par Seeed Studio. Le code suivant permet de charger une page (ici, `google.fr`) et de nous retourner cette page.

3. Disponible sur leur dépôt GitHub : https://github.com/Seeed-Studio/Ethernet_Shield_W5200.

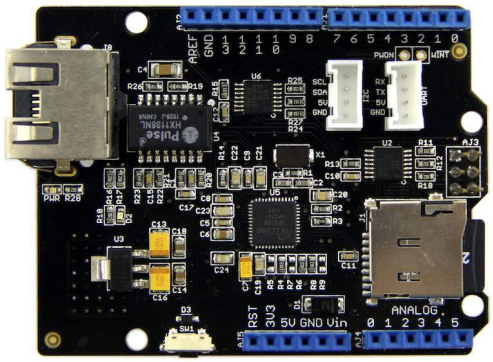


FIGURE 2.3 – L'Ethernet Shield V2.0

```
#include <SPI.h>
#include <EthernetV2_0.h> // On inclus la bibliothèque.

#define W5200_CS 10
#define SDCARD_CS 4

byte mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}; // On déterminer l'adresse MAC de l'Arduino.

char serveur[] = "google.fr"; // Serveur vers lequel on va faire la requête.

EthernetClient client; // On crée un client Ethernet.

void setup()
{
  Serial.begin(9600);

  /* Désactivation de la carte SD */
  pinMode(SDCARD_CS,OUTPUT);
  digitalWrite(SDCARD_CS,HIGH);

  /* On commence la connexion Ethernet. */
  if (Ethernet.begin(mac) == 0)
  {
    Serial.println("Impossible de configuer la connexion Ethernet.");

    while (true); // On arrête alors le programme.
  }

  delay(1000);

  Serial.println("Connexion.");

  /* On se connecte au port 80 du serveur. */
  if (client.connect(server, 80))
  {
    Serial.println("Connecté.");

    client.println("GET / HTTP/1.0"); // On lance une requête HTTP.
    client.println();
  }
  else
  {
    Serial.println("Erreur de connexion.");
  }
}

void loop()
{
```

```
49    /* On affiche le résultat de la requête s'il existe. */
50    if (client.available())
51    {
52      char c = client.read();
53      Serial.print(c);
54    }
55
56    /* Quand le client n'est pas connecté, on arrête le client. */
57    if (!client.connected())
58    {
59      Serial.println("Déconnexion.");
60      client.stop();
61
62      while (true);
63    }
64  }
```

Pour initialiser l'Ethernet V2.0, il faut lui assigner une adresse MAC unique comme on le fait à la ligne 7. Une adresse MAC est un identifiant unique stocké dans la carte réseau.

Revenons sur le code allant de la ligne 34 à 44. Tout d'abord, il faut savoir que chaque serveur ont des ports d'ouvert, le port pour le serveur Web, le logiciel qui délivre le site, est le 80. Ainsi, à la ligne 34, on se connecte à `serveur` en écoutant le port 80. Ensuite, on essaye de faire un requête HTTP pour avoir le contenu de la racine du site.

2.4 Mise en place du programme final

Ainsi, pour notre projet, nous devons associer les deux codes : celui pour la capteur et celui pour l'Ethernet. Pour pouvoir transférer les données à la base de données, nous allons appeler un script PHP situé sur le serveur que nous détaillerons dans le chapitre [référence nécessaire]. Ce script se présente sous la forme suivant.

```
1  http://192.168.1.2/arduino.php?mdp=123&temperature=21&pression=10079
```

Ici, par cet appel, une entrée avec le temps courant, la température de 21 °C et la pression de 10 079 hPa seront ajouter dans la base. Le premier argument sert pour sécuriser son accès.

Le code complet se trouve en annexe à la page 20. En téléversant ce code sur l'Arduino, il va capter une température et une pression toute les 90 s. Au bout de 15 min, il fait la moyenne de 10 valeurs. Enfin, il fait une requête et charge la page qui ajoute les données dans la base.

Chapitre 3

Base de données

3.1 Qu'est-ce qu'un base de données ?

Une *base de données* est un moyen de stocker, d'organiser et de hiérarchiser des données. Une base de données peut être rapprochée d'un classeur Calc ou Microsoft Excel :

— chaque fichier est une base de données ;
— chaque onglet (feuille) correspond à une table ;
— chaque colonne (ou ligne) est un champ spécifiant une caractéristique des données ;
— chaque ligne (ou colonne) est un enregistrement, une donnée.

Il existe plusieurs moyens de stocker des données.

— Le format texte (`.txt`) est un choix assez évident pour les néophytes, il ne nécessite pas de logiciels supplémentaires. Mais il est peu efficace, un fichier devient vite encombrant, il utilise beaucoup de ressources à la lecture et à l'écriture, et chaque table nécessite un fichier séparé qui est ouvert tour à tour.
— Le format texte optimisé pour les données (`.csv`) ne nécessite pas de logiciels supplémentaires. C'est un moyen efficace de stocker les données quand il est bien utilisé. Mais, il a les mêmes inconvénients que le format texte.
— Le classeur Excel ou Calc (`.xls`, `.xlsx` ou `.ods`) est équivalent à une base de données. Mais le fichier devient encombrant et est peu pratique à utiliser, il utilise beaucoup de ressources pour l'utilisation, il n'est pas optimisé pour être utilisé sur un serveur et il doit être créé par un logiciel externe.
— Le format Microsoft DataBase (`.mdb`) est un format spécial base de données, il est entièrement fonctionnel sous Windows. Mais il est impossible à utiliser avec des outils gratuits et doit être créé par un logiciel externe.
— Le format OpenDocument Database (`.odb`) est un format de bases de données libre. Mais tout comme le précedent, il doit être créé par un logiciel extérieur
— Enfin, le format SQL (*Structured Query Language* en anglais, `.sql`) est gratuit et libre. Il est issu d'un consensus entre les différentes technologies pour unifier leur utilisation. Sa connexion est simple (selon les systèmes) et les requêtes basiques sont simples mais peuvent êtres complexes suivant l'utilisation. Mais il nécessite un Système de Gestion de Base de Données (ou SGBD) comme PostrgeSQL, MySQL ou ORACLE qu'il faut installer.

3.2 Notre utilisation

Afin de stocker les différentes données acquises par la partie électronique, nous devons utiliser une basse de données avec un champs pour le temps, la température et la pression.

Pour notre site, nous utilisons MySQL, un Syème de Gestion de Base de Données Relationnel (SGBDR) qui peut créer et gérer des relations entre différentes tables, couplé avec le module PDO de PHP pour se connecter à la base.

Pour pouvoir utiliser une base de données, il faut d'abord installer le serveur MySQL, dont l'installation sera détaillée dans le chapitre [référénce nécessaire].

Après l'installation, nous devons créer la base (que l'on nommera `tpe`). Pour ce faire, dans la console MySQL ou *via* phpMyAdmin (administation pour MySQL en ligne), on rentrera le requête suivante :

```
CREATE DATABASE tpe
```

La base est alors créée mais encore inutilisable car ne contenant pas de tables. Nous créons donc une table nommée `datalog_meteo` contenant plusieurs champs avec la requête :

```
CREATE TABLE datalog_meteo (
    id int(10) unsigned NOT NULL AUTO_INCREMENT,
    temps timestamp NULL DEFAULT CURRENT_TIMESTAMP,
    temperature float DEFAULT NULL,
    pression float DEFAULT NULL,
    PRIAMRY KEY (id)
) ENGINE=InnoDB;
```

La requête crée une table avec les champs `id` (une valeur qui s'auto-incrémente pour différencier les enregistrements), `temps` (une valeur de temps), `temperature` (un nombre décimal) et `pression` (un nombre décimal également).

Table	Action	Lignes	Type	Interclassement	Taille	Perte
datalog_meteo	Afficher Structure Rechercher Insérer Vider Supprimer	~28	InnoDB	utf8_general_ci	16 Kio	-
1 table	Somme	28	InnoDB	utf8_general_ci	16 Kio	0 o

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	id	int(10)		UNSIGNED	Non	*Aucune*	AUTO_INCREMENT
2	temps	timestamp			Oui	CURRENT_TIMESTAMP	
3	temperature	float			Oui	*NULL*	
4	pression	float			Oui	*NULL*	

FIGURE 3.1 – Structure de notre base

La table est bien créée et utilisable mais est vide. Pour la remplir, nous utilisons la requête :

```
--- Valeurs pour tester la table ---
INSERT INTO `datalog_meteo` (`temps`, `temperature`, `pression`) VALUES
    ('2014-12-10 21:00:43', 105, 100);

--- Valeurs réelles ---
INSERT INTO `datalog_meteo` (`temps`, `temperature`, `pression`) VALUES
    ('2015-02-15 18:11:09', 21.09, 1010.46),
    ('2015-02-15 18:12:38', 21.06, 1010.52),
    ('2015-02-15 18:12:40', 21.07, 1010.51),
    ('2015-02-15 18:12:43', 21.09, 1010.58),
    ('2015-02-15 18:12:46', 21.1, 1010.52),
    ('2015-02-15 18:12:48', 21.1, 1010.52),
    ('2015-02-15 18:12:51', 21.1, 1010.54),
    ('2015-02-15 18:12:54', 21.1, 1010.54),
    ('2015-02-15 18:12:56', 21.09, 1010.49),
    ('2015-02-15 18:12:59', 21.1, 1010.54),
    ('2015-02-15 18:46:37', 21.1, 1011.14),
    ('2015-02-15 18:46:40', 21.1, 1011.1),
    ('2015-02-15 18:46:44', 21.1, 1011.07),
    ('2015-02-15 18:46:48', 21.1, 1011.11),
    ('2015-02-15 18:46:54', 21.1, 1011.19),
    ('2015-02-15 18:46:56', 21.1, 1011.1),
    ('2015-02-15 18:46:59', 21.1, 1011.15),
    ('2015-02-15 18:47:02', 21.1, 1011.16),
    ('2015-02-15 18:47:05', 21.1, 1011.1),
    ('2015-02-15 18:47:07', 21.1, 1011.12),
```

```
    ('2015-02-15 18:47:10', 21.1, 1011.1),
    ('2015-02-15 18:47:13', 21.1, 1011.1),
    ('2015-02-15 18:47:15', 21.1, 1011.13),
    ('2015-02-15 18:47:18', 21.1, 1011.09),
    ('2015-02-15 18:47:21', 21.1, 1011.11),
    ('2015-02-15 18:47:23', 21.1, 1011.06),
    ('2015-02-15 18:47:26', 21.1, 1011.15);
```

La requête `INSERT INTO (table) (champs)` indique à MySQL qu'il doit insérer dans les champs indiqués de la table les valeurs suivant le mot-clé `VALUES`.

Nous pouvons maintenant afficher les valeurs contenues dans la table grâce à la requête :

```
SELECT * FROM datalog_meteo;
```

Cette requête affiche tout le contenu de la table `datalog_meteo` sans exception. Nous pouvons cependant restreindre ceci en donnant les noms des champs nous étant utile, et, par exemple, un intervalle de temps ou une limite (une limite de x valeurs retournera au plus x valeurs).

Nous utiliserons par défaut les 20 dernières valeurs, la requête la plus utilisée est alors

```
SELECT temps, temperature, pression FROM datalog_meteo ORDER BY temps DESC LIMIT 20;
```

Nous demandons au serveur SQL de renvoyer les valeurs des champs `temps`, `temperature` et `pression` dans l'ordre de temps décroissant et dans la limite de 20 valeurs. Le serveur renvoie alors un ensemble de données ressemblant à la figure suivante (dépendant des données).

id	temps	temperature	pression
1	2014-12-10 22:00:43	105	100
275	2015-02-15 19:11:09	21.09	1010.46
276	2015-02-15 19:12:38	21.06	1010.52
277	2015-02-15 19:12:40	21.07	1010.51
278	2015-02-15 19:12:43	21.09	1010.58
279	2015-02-15 19:12:46	21.1	1010.52
280	2015-02-15 19:12:48	21.1	1010.52
281	2015-02-15 19:12:51	21.1	1010.54
282	2015-02-15 19:12:54	21.1	1010.54
283	2015-02-15 19:12:56	21.09	1010.49
284	2015-02-15 19:12:59	21.1	1010.54
285	2015-02-15 19:46:37	21.1	1011.14
286	2015-02-15 19:46:40	21.1	1011.1
287	2015-02-15 19:46:44	21.1	1011.07
288	2015-02-15 19:46:48	21.1	1011.11
289	2015-02-15 19:46:54	21.1	1011.19
290	2015-02-15 19:46:56	21.1	1011.1
291	2015-02-15 19:46:59	21.1	1011.15
292	2015-02-15 19:47:02	21.1	1011.16
293	2015-02-15 19:47:05	21.1	1011.1
294	2015-02-15 19:47:07	21.1	1011.12
295	2015-02-15 19:47:10	21.1	1011.1
296	2015-02-15 19:47:13	21.1	1011.1
297	2015-02-15 19:47:15	21.1	1011.13
298	2015-02-15 19:47:18	21.1	1011.09
299	2015-02-15 19:47:21	21.1	1011.11
300	2015-02-15 19:47:23	21.1	1011.06
301	2015-02-15 19:47:26	21.1	1011.15

id	temps	temperature	pression
301	2015-02-15 19:47:26	21.1	1011.15
300	2015-02-15 19:47:23	21.1	1011.06
299	2015-02-15 19:47:21	21.1	1011.11
298	2015-02-15 19:47:18	21.1	1011.09
297	2015-02-15 19:47:15	21.1	1011.13
296	2015-02-15 19:47:13	21.1	1011.1
295	2015-02-15 19:47:10	21.1	1011.1
294	2015-02-15 19:47:07	21.1	1011.12
293	2015-02-15 19:47:05	21.1	1011.1
292	2015-02-15 19:47:02	21.1	1011.16
291	2015-02-15 19:46:59	21.1	1011.15
290	2015-02-15 19:46:56	21.1	1011.1
289	2015-02-15 19:46:54	21.1	1011.19
288	2015-02-15 19:46:48	21.1	1011.11
287	2015-02-15 19:46:44	21.1	1011.07
286	2015-02-15 19:46:40	21.1	1011.1
285	2015-02-15 19:46:37	21.1	1011.14
284	2015-02-15 19:12:59	21.1	1010.54
283	2015-02-15 19:12:56	21.09	1010.49
282	2015-02-15 19:12:54	21.1	1010.54

FIGURE 3.2 – Données de la base sans et avec `LIMIT`

Chapitre 4

Partie site Web

Chapitre 5

Installation du serveur

Pour finaliser notre installation, nous avons mis en place un serveur. Un serveur est une machine informatique permettant de réceptionner des information et de les envoyer par la même occasion à un client.

5.1 Logiciels utilisés

Pour créer notre serveur, nous utilisons un ordinateur où nous avons installé Linux et Apache. Le système d'exploitation Linux permet la mise en place d'un serveur très facilement, nous avons pris la distribution Linux Mint. Apache est un logiciel appelé « serveur HTTP », il permet, lorsqu'il est installé sur un ordinateur, dans faire un serveur Web. Ainsi, si « serveur HTTP »désigne toujours un logiciel, « serveur Web » peut aussi bien désigner le logiciel, en l'occurence Apache.

Dans notre TPE, l'utilisation d'un serveur HTTP nous permet de faire fonctionner la base de données vu précédemment avec le site Web pour en ressortir les données.

Avec Linux, l'installation des logiciels est très facile. Nous utilisons le logiciel XAMPP [1] : il s'agit d'un ensemble de logiciel avec Apache, MySQL, PHP et Perl (nous n'utiliserons pas ce dernier). Après l'avoir télécharger, il suffit d'exécuter le `.run` pour l'installer. Ensuite, on peut utiliser les commandes suivante dans un terminal pour démarrer et arrêter le serveur Apache et MySQL.

```
sudo /opt/lampp/lampp start # Pour démarrer
sudo /opt/lampp/lampp stop  # Pour arrêter
```

Ensuite, par un navigateur Web, nous pouvons y accéder avec l'adresse `http://localhost`. Nous pouvons voir la page d'accueil. Il nous faut maintenant ajouter les fichiers HTML/CSS et PHP du site. Pour cela, il faut créer une nouveau dossier dans `/opt/lampp/htdocs` en ayant les permission du superutilisateurs et transférer tous nos fichiers à l'intérieur.

```
cd /opt/lampp/htdocs # On se déplace dans le dossier du serveur Web.
mkdir tpe # On crée un dossier tpe.
cd tpe
sudo cp -r nos-fichier/ /opt/lampp/htdocs/tpe # On copie nos-fichier dans tpe.
```

Nous pouvions également le faire en utilisant l'interface graphique. Ensuite, les fichiers sont accessible par le navigateur à l'adresse `http://localhost/tpe`.

5.2 Mise en place du réseau

Pour relier ce serveur Web avec notre Arduino, nous devons utiliser un *switch*. Un *switch* désigne un commutateur réseau, il s'agit d'un équipement ou un appareil qui permet l'interconnexion de différent appareils communicants entre eux. Il nous permet ici, grâce à l'Arduino Ethernet Shield,

1. Téléchargeable sur leur site : https://www.apachefriends.org/download.html.

d'interconnecter nos différente partie avec des câbles Ethernet. L'Arduino est donc relier au *switch* tout comme l'ordinateur qui héberge le serveur Web.

Pour connecter différent appareil, il faut utiliser des adresses IP pour que nous puissions reconnaître les appareils. Il existe des adresses IP de version 4 (sur 32 bits, soit 4 octets) et de version 6 (sur 128 bits, soit 16 octets). La version 4 est actuellement la plus utilisée : elle est généralement représentée en notation décimale avec quatre nombres compris entre 0 et 255, séparés par des points, ce qui donne par exemple 212.85.150.134.

Nous somme en réseau local donc nous devons utiliser l'adresse IP suivante 192.168.1.X en remplaçant le X par 1, 2 ou 3 en fonction de la configuration du PC et de l'Arduino. Dans notre cas, 192.168.1.1 va représenter le serveur, 192.168.1.2 l'Arduino et 192.168.1.3 le client. Tout cela peut être représenté par la figure suivante.

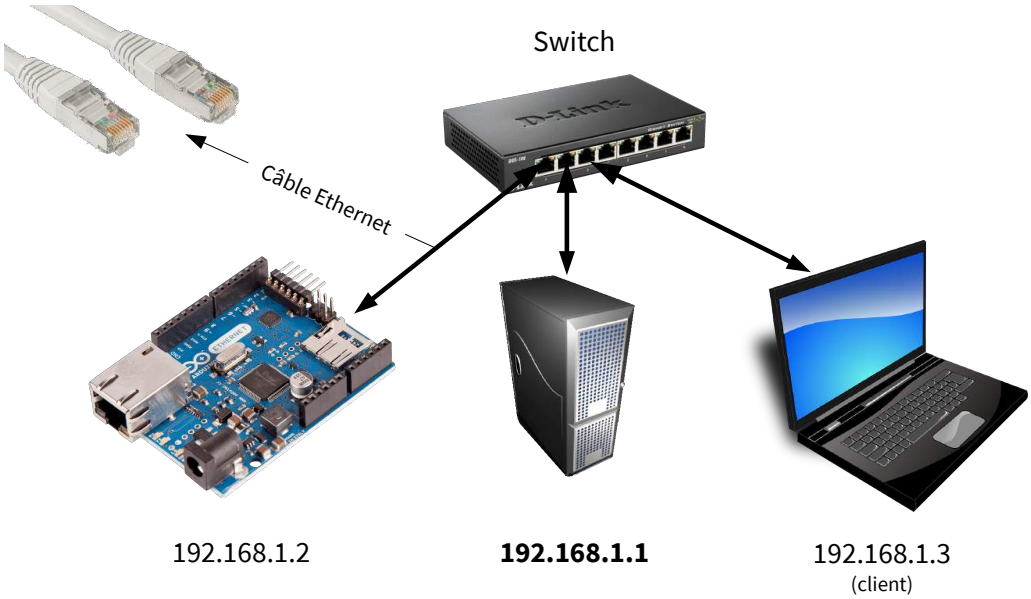


FIGURE 5.1 – Schéma de notre réseau local

À l'avenir, si nous utiliserions réellement notre projet, l'Arduino et le serveur n'iraient plus sur un switch mais seraient branchés sur une *box*, le site et la base de données seraient hébergés dans un *data-center* pour que l'utilisateur final y est accès depuis n'importe quelle connexion Internet. Ils n'auront par conséquent plus les mêmes adresses IP.

Webographie

[1] Eskimon et olyte. *Arduino : Premiers pas en informatique embarquée — Zeste de Savoir*. 2015. URL : https://zestedesavoir.com/tutoriels/537/arduino-premiers-pas-en-informatique-embarquee/.

[2] Wikipédia. *Qualité métrologique des appareils de mesure — Wikipédia, l'encyclopédie libre*. 2015. URL : http://fr.wikipedia.org/wiki/Qualit%C3%A9_m%C3%A9trologique_des_appareils_de_mesure.

[3] Seeed Studio. *Grove - Barometer Sensor (BMP180)*. 2014. URL : http://www.seeedstudio.com/wiki/Grove_-_Barometer_Sensor_%28BMP180%29.

[4] Wikipédia. *I^2C — Wikipédia, l'encyclopédie libre*. 2014. URL : http://fr.wikipedia.org/wiki/I2C.

[5] Seeed Studio. *Ethernet Shield V2.0*. 2014. URL : http://www.seeedstudio.com/wiki/Ethernet_Shield_V2.0.

[6] Wikipédia. *Serveur HTTP — Wikipédia, l'encyclopédie libre*. 2014. URL : http://fr.wikipedia.org/wiki/Serveur_HTTP.

[7] Wikipédia. *Adresse IP — Wikipédia, l'encyclopédie libre*. 2014. URL : http://fr.wikipedia.org/wiki/Adresse_IP.

Annexe A

Code complet pour l'Arduino

```
/* Inclusion des bibliothèques */
#include <Wire.h>
#include <SPI.h>
#include <Barometer.h>
#include <EthernetV2_0.h>

#define DEBUG true // Pour déboguer

#define W5200_CS 10
#define SDCARD_CS 4

#define NB_RELEVES 10 // Nombre de relevés par passage
#define INTERVALLE 1.5 * 1000 // Intervalle entre les envois de données
#define TEMPS_INTERVALLE / NB_RELEVES // Temps entre chaque relevé

Barometer capteur; // Déclaration du capteur

byte MAC[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
IPAddress IP(192, 168, 1, 3); // Adresse IP de l'Arduino
IPAddress DNS(192, 168, 1, 1);

EthernetClient client; // Déclaration du client Ethernet

char serveur[] = "192.168.1.2"; // Adresse IP du serveur

unsigned long tempsDerniereConnexion;
boolean dernierConnecte(false);

void setup()
{
  capteur.init(); // Initialisation du BMP180

  /* Désactivation de la carte SD */
  pinMode(SDCARD_CS, OUTPUT);
  digitalWrite(SDCARD_CS, HIGH);

  delay(1000);

  Ethernet.begin(MAC, IP); // Démarrage de la connexion Ethernet

  if (DEBUG)
  {
    Serial.begin(9600); // Démarrage de la liaison série
    Serial.print("Mon adresse IP : ");
    Serial.println(Ethernet.localIP());
  }
}

```

```
49  void loop()
50  {
51    /* Affichage des résultats des requêtes */
52    if (DEBUG)
53    {
54      while (client.available())
55      {
56        char c(client.read());
57        Serial.print(c);
58      }
59    }
60
61    if (!client.connected() && dernierConnecte)
62    {
63      Serial.println("Déconnexion.");
64
65      client.stop();
66    }
67
68    /* moyennes : tableau de 2 cases
69      - 1re : moyenne de la température
70      - 2de : moyenne de la pression */
71    float moyennes[2] = {};
72
73    /* Relevés des données */
74    for (int i(0) ; i < NB_RELEVES ; i++) // On fait la somme de 10 relevés de température et de pression.
75    {
76      moyennes[0] += capteur.bmp085GetTemperature(capteur.bmp085ReadUT()); // Température
77      moyennes[1] += capteur.bmp085GetPressure(capteur.bmp085ReadUP()); // Pression
78
79      delay(TEMPS);
80    }
81
82    moyennes[0] = moyennes[0] / NB_RELEVES;
83    moyennes[1] = moyennes[1] / NB_RELEVES / 100; // Division par 100 pour convertir les Pa en hPa
84
85    /* Affichage des valeurs */
86    if (DEBUG)
87    {
88      Serial.print("Moyennes : ");
89      Serial.print(moyennes[0]);
90      Serial.print(" °C et ");
91      Serial.print(moyennes[1]);
92      Serial.println(" hPa");
93    }
94
95    if (!client.connected() && (millis() - tempsDerniereConnexion > INTERVALLE))
96    {
97      if (client.connect(serveur, 80)) // Connexion au port 80
98      {
99        if (DEBUG)
100         Serial.println("Connexion.");
101
102       /* Envoi de la requête */
103       client.print("GET /arduino.php?pass=123&temperature=");
104       client.print(moyennes[0]);
105       client.print("&pression=");
106       client.print(moyennes[1]);
107       client.println(" HTTP/1.1");
108       client.println("Host: 192.168.1.2");
109       client.println("User-Agent: Arduino Ethernet");
110       client.println("Connection: close");
111       client.println();
112
113       tempsDerniereConnexion = millis();
```

```
114       }
115       else
116       {
117         if (DEBUG)
118         {
119           Serial.println("Erreur de connexion.");
120           Serial.println("Déconnexion.");
121         }
122
123         client.stop();
124       }
125     }
126
127     dernierConnecte = client.connected();
128   }
```