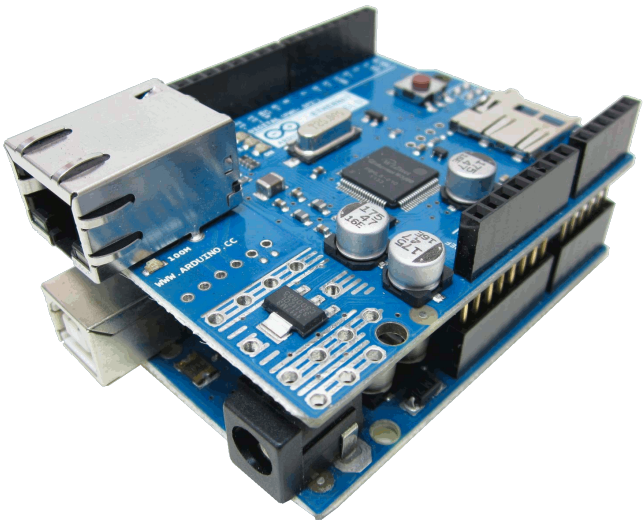
Travaux personnels encadrés

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 12 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 utilisées pour nous rendre le quotidien meilleur. Ce domaine utilise quasiment toutes 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 ils peuvent également augmenter notre confort 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 problématique, 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 délivrerai, 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 détails notre démarche pour créer cet objet avec les parties sur l'électronique, la base de données, le site Web et la mise en place de notre serveur. 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 données météorologiques et le transfert de ces données sur un site Web accessible à tous. Nous voulions créer un système qui permet à l'utilisateur de connaître la température ainsi que la pression d'un environnement à distance. Ce type de produits 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 les relevés sur une base de données en ligne. De plus, notre projet relève non seulement la température mais aussi la pression. Nous avons aussi essayé de réduire les coûts.

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

1.2 Analyse fonctionnelle

Nous nous sommes donc penchés sur les caractéristiques que nous devions mettre en œuvre pour arriver à mener à bien notre projet.

1.2.1 Méthode APTE

Dans un premier lieu, nous nous sommes demandé à quoi cela pourrait servir et pour qui ce serait utile. Nous avons donc utilisé la méthode dite « méthode APTE » qui permet de déterminer l'utilité de notre projet. Au début, nous avons créé le diagramme bête à cornes suivant.

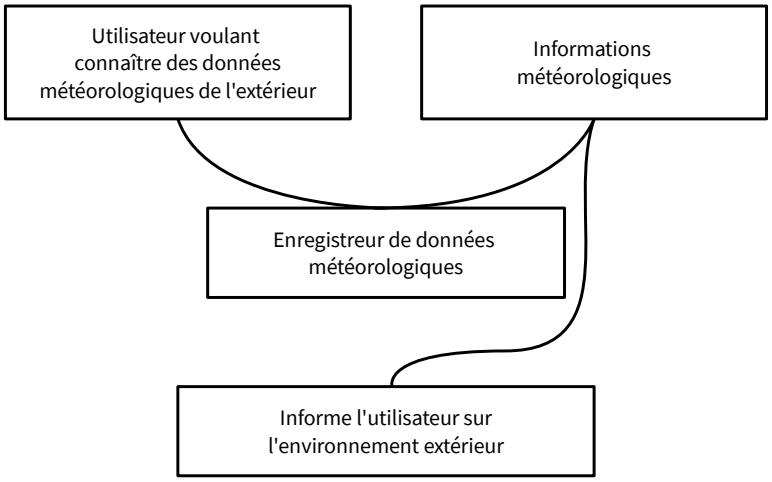


FIGURE 1.2 – Diagramme bête à cornes de notre projet

Nous pouvons donc penser que les données météorologiques que nous recueillerons serviront à l'utilisateur. Celui-ci pourra alors être informé de la température et de la pression atmosphérique d'une pièce ou d'un extérieur en fonction de son utilisation.

Ensuite, il a fallu nous poser quelques questions sur ce qui pourrait conduire à la disparition de ce projet. Il pourrait, par exemple, naître un nouveau projet qui puisse capter des données plus précises ou plus fidèles, qui puisse stocker plus de données, ... Mais pour l'instant, nous n'avons pas vu d'autre produit comme le notre !

1.2.2 Diagramme des interacteurs

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

Ainsi, sept interacteurs ont été trouvés 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 regroupé toutes nos caractéristiques et nos fonctions pour former un cahier des charges fonctionnel (voir tableau 1.1).

Nous avons donc déterminé, en fonction de nos attentes, 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 flexibilité d'un niveau est faible, moins on peut le modifier. Nous pouvons donc voir, par exemple, que pour la température, on autorise une flexibilité de 2 ce qui veut dire que l'on autorise une précision un peu plus faible. Autre exemple, pour l'esthétique, la flexibilité est au maximum car cela n'empêchera pas les performances ou les fonctionnalités de l'objet.

	Fonctions	Critères	Niveaux	Fléxibilité
FP_1	Doit pouvoir communiquer des informations sur le réseau Intranet/Internet	Prise RJ45 et accès au réseau de lycée 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 visibles par l'utilisateur	Site Web avec langage de programmation	Espace inférieur à 10 Mo	F_2
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 d'un transformateur à courant continu	Tension de 5 V à 12 V	F_0

TABLE 1.1 – Le cahier des charges fonctionnel

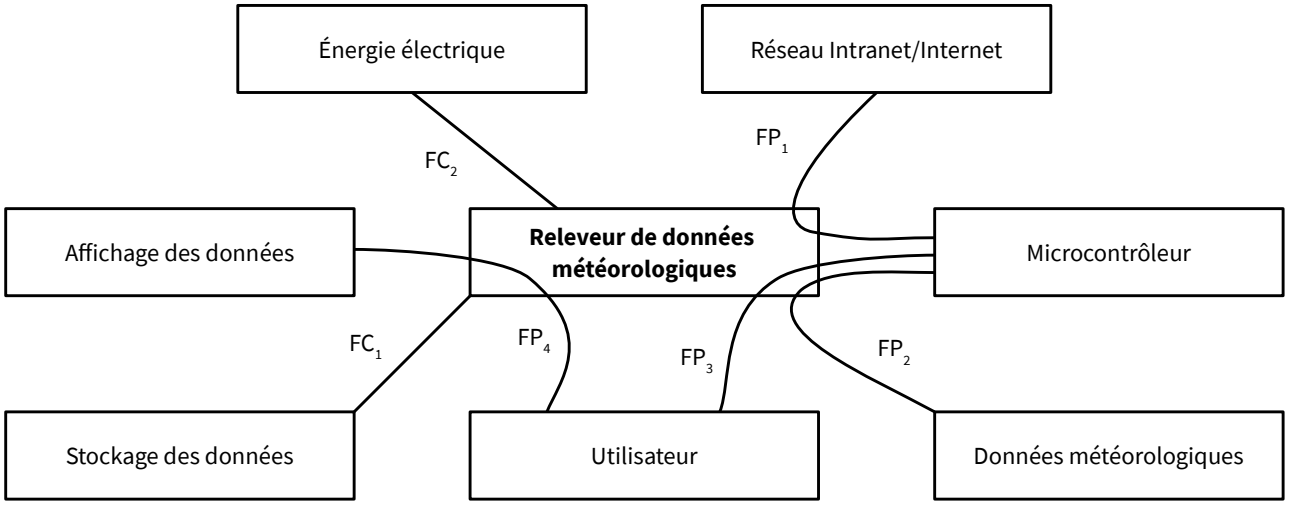


FIGURE 1.3 – Diagramme des interacteurs

1.2.4 Synoptique et chaîne d'informations

Nous avons commencé à rentrer dans les détails de notre TPE en choisissant notre capteur, qui sera un capteur de température et de pression BMP180 et notre microcontrôleur sera 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 données qui se situera sur un serveur qui hébergera aussi le site Web. Un *switch* sera installé entre l'Arduino et l'ordinateur, nous pourrons également brancher un deuxième ordinateur en tant que client (l'utilisateur final). Pour notre base de données, nous allons utiliser MySQL et, pour notre serveur Web, nous allons utiliser Apache. Le langage pour le site Web sera PHP. Tous ces logiciels sont présentés dans la section 5.1.

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 déterminé notre chaîne d'informations (voir figure 1.5), de l'acquisition de la température à l'insertion dans la base de données. La partie site Web ne figure pas car elle est indépendante de cette partie bien que le site communique avec la base.

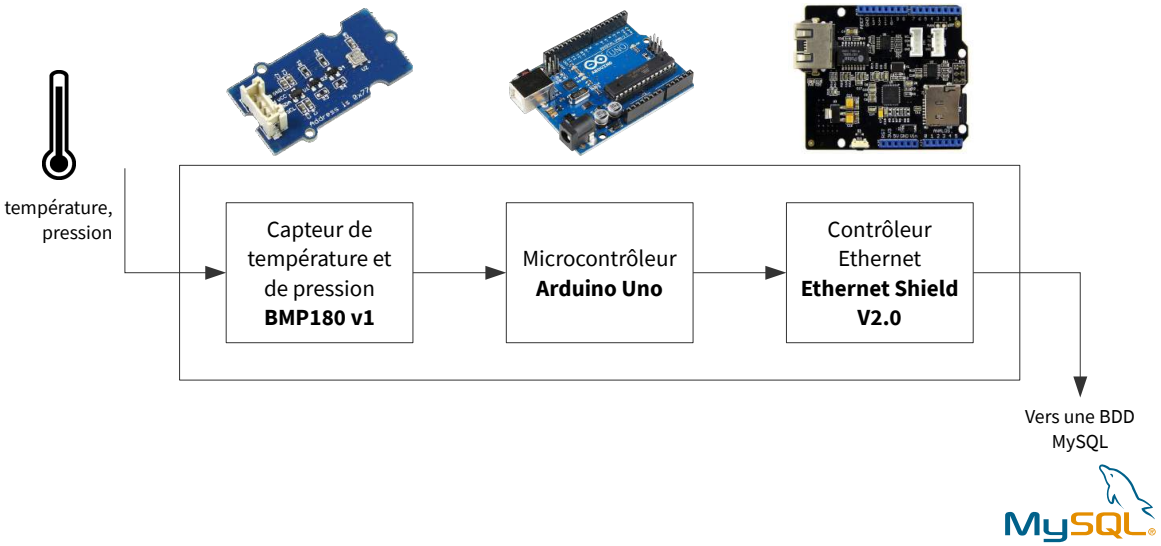


FIGURE 1.5 – Chaîne d'informations de la partie électronique

Chapitre 2

Partie électronique

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

2.1 Une plateforme pour recevoir 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 de contrôler différents récepteurs 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 commandées à l'aide d'un code compilé sur un ordinateur puis téléversé sur le microcontrôleur.

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

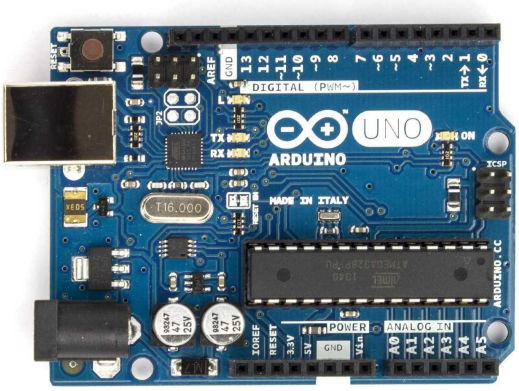


FIGURE 2.1 – Un Arduino Uno R3

2.1.2 Programmation d'un Arduino

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

Le 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, nous pouvons 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 conseillées pour les débutants (comme nous) en électronique. Dans ce code, il est possible d'ajouter des conditions avec `if .. else if ... 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, hydraulique, ... Les capteurs sont utilisés quasiment partout et il en existe beaucoup : certains peuvent capter 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 de 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

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

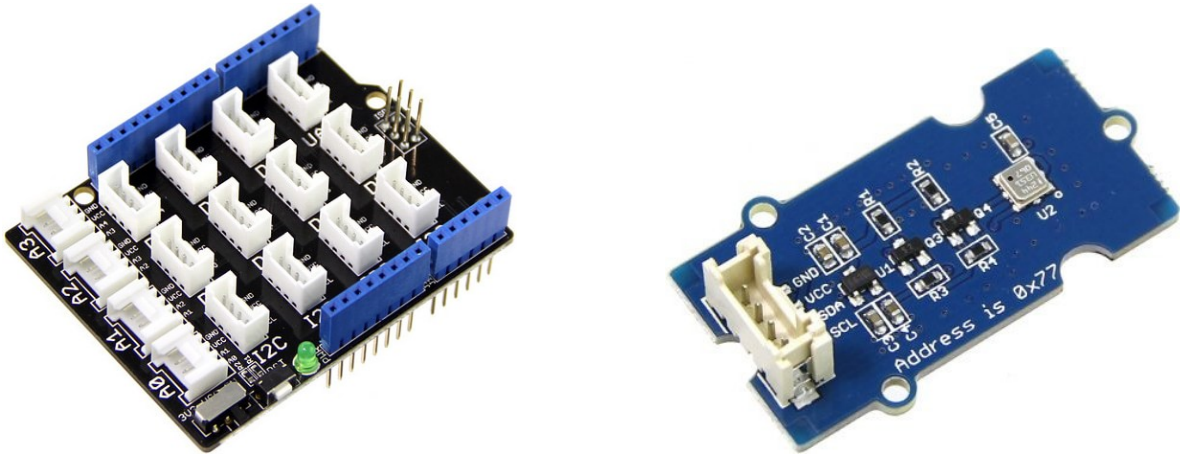


FIGURE 2.2 – Le Base Shield V2 et le BMP180

Caractéristiques Pour distinguer différents capteurs, il existe des caractéristiques :
- l'étendue : c'est l'écart entre la plus grande et la plus petite valeur mesurable,
- la résolution : c'est la plus petite variation que le capteur est capable de mesurer. 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 le 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 une valeur juste, c'est l'écart entre le résultat moyen et la valeur vraie ;
- 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 la température ainsi que la pression. Au lycée, nos deux professeurs nous ont orienté 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 câblage 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 créé 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 bibliothèque [2] qui va nous faciliter le travail. Il suffit alors de décompresser l'archive dans le dossier des bibliothèques Arduino.

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

```
#include <Barometer.h> // On inclut 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 quarts d'heure, 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 extrêmes et augmentera la fidélité des relevés. Pour ce faire, nous utilisons une 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.

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

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

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

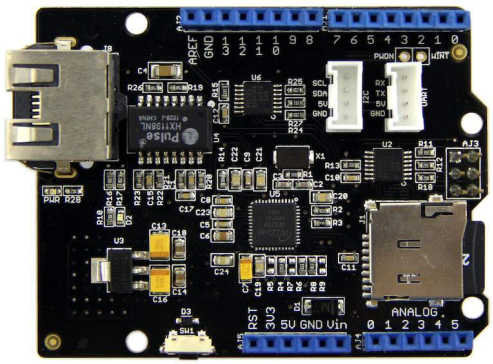


FIGURE 2.3 – L'Ethernet Shield V2.0

```
9
10  moyenneTemperature /= 10;
11  // ...
```

2.3 Connexion de l'Arduino au réseau Ethernet

Une des fonctionnalités essentielles 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 lancer des requêtes TCP/UDP sur un réseau Ethernet. Il faut bien évidemment 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 la retourner.

```
1   #include <SPI.h>
2   #include <EthernetV2_0.h> // On inclut la bibliothèque.
3
4   #define W5200_CS 10
5   #define SDCARD_CS 4
6
7   byte MAC[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED}; // On donne une adresse MAC de
     ↪  l'Arduino.
8
9   char serveur[] = "google.fr"; // Serveur vers lequel on va faire la requête.
10
11  EthernetClient client; // On crée un client Ethernet.
12
13  void setup()
14  {
15    Serial.begin(9600);
16
17    /* Désactivation de la carte SD */
18    pinMode(SDCARD_CS, OUTPUT);
19    digitalWrite(SDCARD_CS, HIGH);
20
21    /* On commence la connexion Ethernet. */
22    if (Ethernet.begin(MAC) == 0)
```

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

```
23    {
24      Serial.println("Impossible de configuer la connexion Ethernet.");
25
26      while (true); // On arrête alors le programme.
27    }
28
29    delay(1000);
30
31    Serial.println("Connexion.");
32
33    /* On se connecte au port 80 du serveur. */
34    if (client.connect(server, 80))
35    {
36      Serial.println("Connecté.");
37
38      client.println("GET / HTTP/1.0"); // On lance une requête HTTP.
39      client.println();
40    }
41    else
42    {
43      Serial.println("Erreur de connexion.");
44    }
45  }
46
47  void loop()
48  {
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 nous le faisons 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 a des ports d'ouvert, le port pour le serveur Web, le logiciel qui délivre le site, est le 80. Ainsi, à la ligne 34, nous essayons de nous connecter à `serveur` en écoutant le port 80. Ensuite, nous lançons 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 la section 4.2. Ce script se présente sous la forme suivante.

```
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 sera ajoutée dans la base. Le premier argument sert pour sécuriser son accès.

Le code complet se trouve en annexe à la page 31. En téléversant ce code sur l'Arduino, il va capter une température et une pression toutes 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.

Cependant, il subsiste un problème : l'Arduino charge la page demandée uniquement lorsqu'il est connecté à un ordinateur par USB. Nous n'avons pas réussi à régler ce problème...

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 consomme beaucoup de ressources, 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 précédent, 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) ainsi que les requêtes basiques mais peuvent êtres complexes suivant les besoins. 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 base de données avec un champs pour le temps, la température et la pression.

Pour notre site, nous utilisons MySQL, un Système 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 la section 5.1.

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), nous rentrerons 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 celui-ci 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

Afin de réaliser notre projet, nous avons dû créer un site Web pour accueillir toutes les données que l'Arduino aura obtenu grâce au capteur.

4.1 Deux technologies : HTML/CSS et PHP

Pour ce faire, il nous a fallu coder ce site avec le langage HTML pour la partie fixe du site et le langage PHP pour la partie du qui viendra à changer au cours du temps.

4.1.1 Partie statique : HTML et CSS

Pour créer un site Web, il faut le coder à la main[1]. Pour cela, il existe deux langages que sont le HTML et le CSS. À l'heure actuelle, le HTML est en version 5 et le CSS en version 3. Le HTML et le CSS sont complémentaires. En effet, le premier nous sert pour le fond, c'est-à-dire le contenu ; le second pour la forme, l'apparence.

Le HTML est un langage balistique : il utilise des balises de la forme `<balise>` et `</balise>`. Il est donc très compréhensible. Voici un exemple de page HTML :

```
<!DOCTYPE html><!-- Spécification de la version du HTML -->
<html>
    <head><!-- En-tête : toutes les informations non-visible par l'utilisateur -->
        <meta charset="utf-8">
        <title>Exemple de page HTML</title><!-- Titre de la page -->
    </head>
    <body><!-- Corps de la page -->
        <h1>Un titre</h1>

        <p>Un paragraphe avec un <a href="http://localhost">lien</a>.</p>
    </body>
</html>
```

D'un autre côté, le CSS s'organise en propriétés, en classes et en identificateurs. Il est capable de mettre en forme le HTML. Pour associer une classe ou un identificateur à une balise, il suffit d'utiliser respectivement les arguments `class="ma-class"` et `id="mon-identificateur"` sur une balise quelconque. Contrairement à une classe, un identificateur ne peut s'utiliser qu'une fois dans une page. Dans le fichier CSS, les classes sont précédées d'un point, les identificateurs d'un croisillon `#` et leurs propriétés sont entre accolades.

Supposons un fichier CSS nommé `style.css` avec le contenu suivant.

1. Ou en passant par un logiciel spécifique.

```css
.important { /* Définition d'une classe */
    color: red;
    font-weight: bold;
}

h1 { /* Redéfinition d'un style de balise ; ici, le titre h1 */
    color: green;
    text-decoration: underline;
}
```

À partir de cela, nous pouvons créer un fichier HTML utilisant ces classes.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Un exemple avec de CSS</title>
        <link href="style.css" rel="stylesheet" type="text/css"><!-- Inclusion du
        ↪ fichier CSS -->
    </head>
    <body>
        <h1>Un titre</h1>

        <p>Un mot <span class="important">très important</span> !</p>
    </body>
</html>
```

Ainsi, nous pouvons voir le résultat sur la figure suivante.

FIGURE 4.1 – Exemple de CSS

Notre site est alors constitué de ces deux langages. Vous pouvez en voir un aperçu sur la figure suivante ou en ligne sur http://tpe.teguad.ovh.

FIGURE 4.2 – Notre site sans et avec le CSS

4.1.2 Partie dynamique : PHP

Mais notre site a besoin d'être dynamique, de changer au cours du temps. Pour cela, nous utilisons le langage PHP. Ce langage s'exécute sur le serveur qui nous avons mis en place (voir chapitre 5) à l'inverse du HTML et du CSS qui s'exécute du côté client.

Un code PHP se trouve entre `<?php` et `?>`. À l'intérieur, nous pouvons mettre des variables, des structures conditionnelles, des boucles, etc. Le code suivant affiche un simple texte.

```php
<?php echo "Un texte"; ?>
```

Le PHP sert à générer du code HTML statique. Ce langage va nous servir à ajouter et à récupérer les différentes données dans la base MySQL.

Exécuter des requêtes SQL avec PHP

Pour communiquer avec une base MySQL, nous utilisons un module PHP nommé PDO. Celui-ci facilite les requêtes. Le code suivant liste les données de la table `datalog_meteo` présentée dans la partie précédente.

```php
<?php

/* On essaie de se connecter à la base. */
try
{
    /* Connexion
       - tpe : nom de la base
       - localhost : serveur (ici, en local)
       - root : utilisateur
       - root : mot de passe */
    $bdd = new PDO("mysql:dbname=tpe;host=localhost", "root", "root",
     ↪  array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING));
}
catch(PDOException $e)
{
    die("Impossible de se connecter à MySQL : ".$e->getMessage()); // Sinon, on affiche
     ↪  un message d'erreur
}

$requete = $bdd->query("SELECT * FROM `datalog_meteo` ORDER BY id DESC LIMIT 5"); // On
 ↪  lance une requête sur la table datalog_meteo : les 5 derniers enregistrements.
$donnees = $requete->fetchAll(PDO::FETCH_ASSOC); // Génération du tableau contenant les
 ↪  données

foreach($donnees as $donnee) // Parcours des données ($donnees : tous le résultat de la
 ↪  requête ; $donnee : un enregistrement)
{
    $temps = preg_replace("#(\d{4})-(\d{2})-(\d{2}) (\d{2}:\d{2}:\d{2})#", "$3/$2/$1 à
     ↪  $4", $donnee["temps"]); // Conversion du temps

    /* Affichage des donneés */
    echo $temps." ; ".$donnee["temperature"]." °C ; ".$donnee["pression"]." hPa\r\n"; //
     ↪  \r\n : saut de ligne
}
```

Le résultat généré est alors visible dans un navigateur.

```
18/02/2015 à 12:07:14 ; 24.3 °C ; 1036.03 hPa
18/02/2015 à 12:07:10 ; 24.3 °C ; 1036.06 hPa
18/02/2015 à 12:07:07 ; 24.3 °C ; 1036.05 hPa
18/02/2015 à 12:07:03 ; 24.3 °C ; 1036.07 hPa
18/02/2015 à 12:07:00 ; 24.3 °C ; 1036.07 hPa
```

4.2 Ajout des données par l'Arduino

Afin que l'Arduino puisse transmettre ces données à la base, il va falloir créer un script qui va être chargé par celui-ci (voir section 2.4). Ce script va recevoir les arguments passés dans

l'URL. Il y aura trois arguments :

— `temperature` contiendra la température ;
— `pression` contiendra la pression ;
— et `mdp` contiendra le mot de passe pour sécuriser l'accès à la base (ici, `123`).

Ainsi, nous créons un fichier `arduino.php` à la racine du site. Le fichier `sql.php` qui contient tout le code pour accéder à la base MySQL est disponible en annexe.

```php
<?php

require 'sql.php'; // Inclusion du fichier contenant les commandes pour accéder à la base

$data = array(); // Création d'un tableau contenant les données

if (isset($_GET['pass']) && $_GET['pass'] == '123') // On test si le mot de passe est bon.
{
    $data['temps'] = 'NOW()'; // Le temps est maintenant.
    /* Si il y a une température donnée, alors on la convertit. Sinon, rien. */
    $data['temperature'] = (isset($_GET['temperature'])) ?
                                htmlspecialchars($_GET['temperature']) : NULL;
    /* Idem pour la pression */
    $data['pression'] = (isset($_GET['pression'])) ?
                                htmlspecialchars($_GET['pression']) : NULL;

    insert($data); // Et on insert les données dans la base.
}
```

4.3 Récupération des données

L'objectif principal du site est de proposer un tableau de données. À l'aide de PHP, il faut coder une fonction qui crée le tableau recueillant les données venant la base avec un nombre limité de valeurs pour éviter un tableau trop long qui serait illisible.

Il a donc fallu d'abord créer l'en-tête du tableau puis demander à la base d'aller chercher les valeurs. Nous avons donc créé une fonction que nous appellerons sur la page adéquate. Encore une fois, le fichier `connect.php` qui permet la connexion à la base se trouve en annexe.

```php
<?php

require 'connect.php'; // Inclusion du fichier pour la connexion à la base

/* Fonction qui crée et affiche le tableau
   - $limit le nombre de donneés que le tableau doit afficher */
function table($limit)
{
    $db = $GLOBALS['db'];
?>
    <table>
        <thead>
            <tr>
                <td>Date et heure</td>
                <td>Température (&deg;C)</td>
                <td>Pression (hPa)</td>
            </tr>
        </thead>
        <tbody>
```

```php
<?php
    $query = $db->query('SELECT temps, temperature, pression FROM `datalog_meteo` ORDER BY id DESC LIMIT '.$limit); // Requête SQL avec LIMIT
    $meteo = $query->fetchAll(PDO::FETCH_ASSOC);

    foreach($meteo as $data) // Parcours des données
    {
        $temps = preg_replace('#(\d{4})-(\d{2})-(\d{2}) (\d{2}:\d{2}:\d{2})#', '$3/$2/$1 à $4', $data['temps']); // Conversion du temps
        /* Nombre décimal avec une virgule */
        $temperature = preg_replace('/\./', ',', $data['temperature']);
        $pression = preg_replace('/\./', ',', $data['pression']);

        /* Affichage des donneés : création d'une ligne */
        echo '<tr>';
            echo '<td>'.$temps.'</td>';
            echo '<td>'.$temperature.'</td>';
            echo '<td>'.$pression.'</td>';
        echo "</tr>\r\n";
    }
?>
        </tbody>
    </table>
<?php
}
```

Ainsi, les valeurs s'ajoutent petit à petit dans les cases avec l'heure à laquelle elle ont été prises. Par exemple, en créant une page avec la fonction `afficher_tableau(20)`, nous pourrons voir un tableau des 20 derniers relevés.

4.4 Le graphique de données

Enfin, pour que les données soient plus lisibles, nous avons décidé de créer un graphique. Sur Internet, nous avons repéré Highchart : c'est une bibliothèque JavaScript générant des graphiques vectoriels en HTML.

À partir des démonstrations proposées sur le site de Highchart [2], nous pouvons créer un graphique pour les températures. Afin de faire apparaître ces courbes, il faut d'abord dessiner les axes (heures en abscisses, température en ordonnées), les graduer et d'afficher la courbe.

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Graphique avec Highcharts</title>
        <!-- Inclusion des bibliothèques -->
        <script src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
        <script src="http://code.highcharts.com/highcharts.js"></script>
        <script src="http://code.highcharts.com/modules/exporting.js"></script>
    </head>
    <body>
        <div id="graphique"></div><!-- Bloc pour le graphique -->

        <script>
            $(function() {
                $('#graphique').highcharts({
```

2. Voir http://www.highcharts.com/demo.

```
                    chart: {
                        type: 'line'
                    },
                    title: {
                        text: 'Graphique des températures'
                    },
                    xAxis: { // Abscisses
                        categories: ['14 h', '14 h 15', '14 h 30', '14 h 45', '15 h',
                        ↪  '15 h 15', '15 h 30', '15 h 35', '16 h', '16 h 15', '16 h
                        ↪  30', '16 h 45']
                    },
                    yAxis: { // Ordonnées
                        title: {
                            text: 'Température (°C)'
                        }
                    },
                    plotOptions: {
                        line: {
                            dataLabels: {
                                enabled: true // Affichage des valeurs à côté des points
                            }
                        }
                    },
                    series: { // Valeurs
                        name: 'Lycée',
                        data: [21, 23, 22, 21.5, 21.6, 22, 23.3, 22.1, 21.5, 22.6]
                    }
                });
            });
        </script>
    </body>
</html>
```

Nous pouvons ainsi observer le résultat suivant.

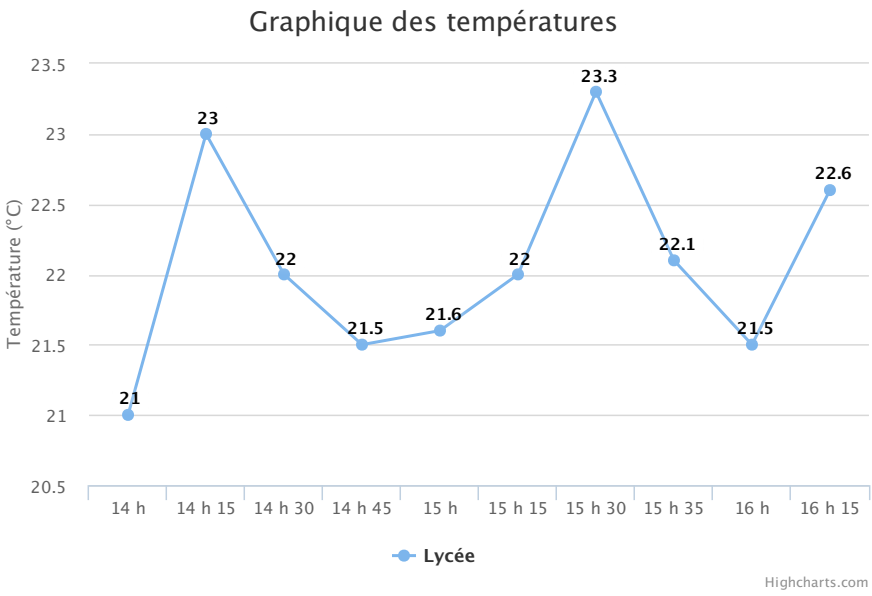


FIGURE 4.3 – Graphique avec Highchart

Il faut donc que nous codions une fonction PHP qui récupère les données et qui les adapte au code pour Highchart. Ce graphique mettra en image l'évolution de la température et de la pression en fonction du temps.

Tout le code du site, un peu long, est disponible en annexe à la page 34.

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 informations 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, Apache, MySQL et PHP. 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, d'en faire un serveur Web. Ainsi, si « serveur HTTP » désigne toujours un logiciel, « serveur Web » peut aussi bien désigner le logiciel, en l'occurence Apache. Le SGBD [1] MySQL sert à gérer les bases de données. Enfin, PHP nous sert pour le côté dynamique du site Web.

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

Avec Linux, l'installation des logiciels est très facile. Nous utilisons le logiciel XAMPP [2] : il s'agit d'un ensemble de logiciels avec Apache, MySQL, PHP et Perl (nous n'utiliserons pas ce dernier). Après l'avoir téléchargé, il suffit d'exécuter le `.run` pour l'installer. Ensuite, on peut utiliser les commandes suivantes 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 permissions du superutilisateur et transférer tous nos fichiers à l'intérieur.

```
cd /opt/lampp/htdocs # On se déplace dans le dossier du serveur Web.
sudo 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 accessibles par un navigateur à l'adresse `http://localhost/tpe`.

1. Système de gestion de bases de données.
2. Téléchargeable sur leur site : https://www.apachefriends.org/download.html.

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 différents appareils communicants entre eux. Il nous permet ici, grâce à l'Arduino Ethernet Shield, d'interconnecter nos différentes parties avec des câbles Ethernet. L'Arduino est donc relier au *switch* tout comme l'ordinateur qui héberge le serveur Web.

Pour connecter les appareils, il faut utiliser des adresses IP pour que nous puissions les reconnaître. 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 sommes 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.

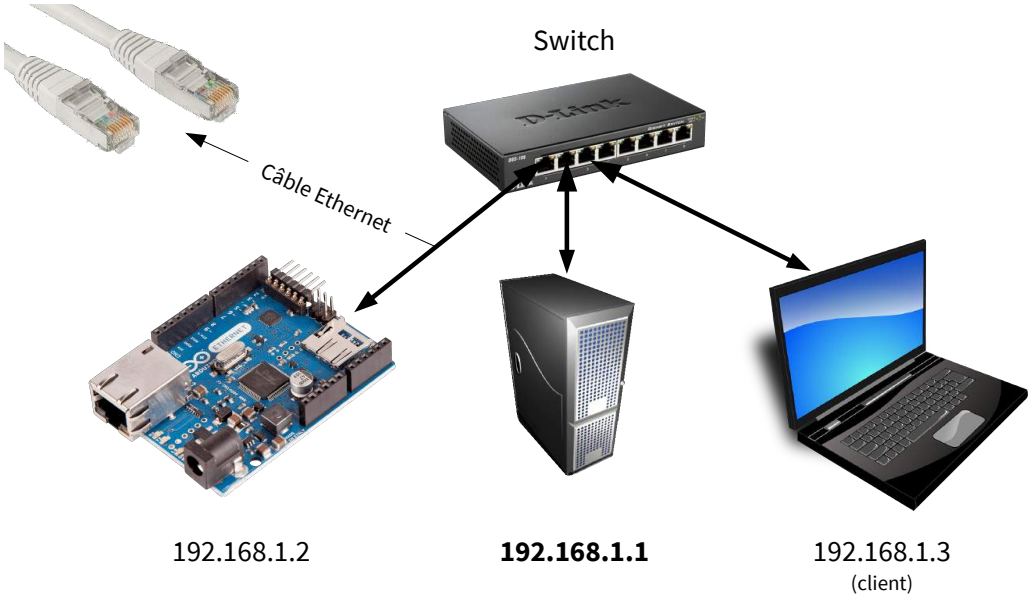


FIGURE 5.1 – Schéma de notre réseau local

Il nous reste plus qu'à configurer le PC et l'Arduino avec les bonnes adresses IP. Concernant l'Arduino, il suffit d'invoquer la commande `Ethernet.begin(MAC, IP)` avec une adresse MAC quelconque et l'IP donnée. Pour notre serveur, il faut aller dans « Paramètres système » puis cliquer sur « Réseau ». Ensuite, il faut aller dans la partie « Filaire » puis cliquer sur le bouton « Option… ». Enfin, dans l'onglet « Paramètre IPv4 », nous pouvons insérer la configuration en sélectionnant « Manuel » puis en ajoutant l'IP en cliquant sur « Ajouter » comme le montre la figure 5.2.

À l'avenir, si nous utilisons 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. Par conséquent, ils n'auraient plus les mêmes adresses IP.

Nous avons mis en place notre site sur un VPS (*Virtual Private Server*). Nous avons installé les différents logiciels sauf que pour le serveur Web, nous avons pris nginx et pour la base, MariaDB (une alternative libre à MySQL). Ainsi, notre site est disponible à l'adresse `http://tpe.teguad.ovh`.

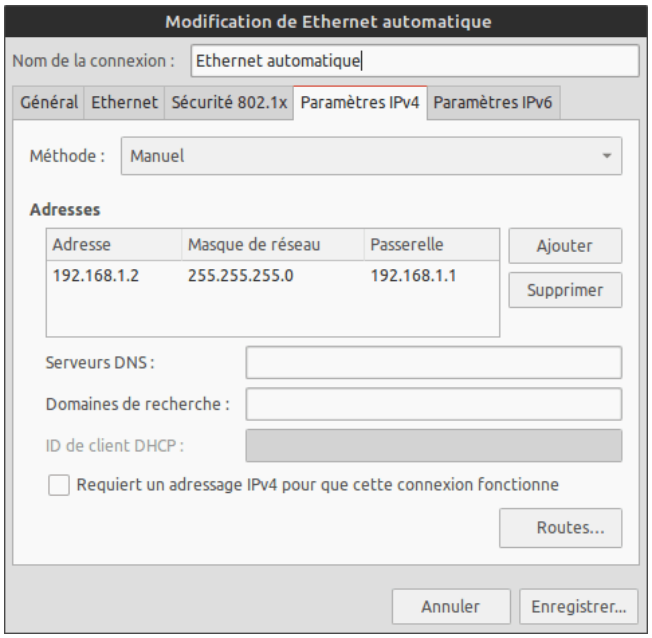


FIGURE 5.2 – Configuration réseau du serveur

Chapitre 6

Résultats finaux

Après quelques essais, nous pensons avoir plutôt bien réussi notre projet. Mais il subsiste quelques défauts : une précision du capteur qui n'est pas parfaite mais satisfaisante pour une utilisation domestique, l'Arduino et son *shield* ne peuvent fonctionner sans la connexion avec l'ordinateur.

Les données obtenues s'affichent très bien sur le site. Par ailleurs, nous pouvons changer la fréquence d'envoi des données.

Rappelons que vous pouvez essayez sur notre site :

— le tableau à l'adresse `http://tpe.teguad.ovh/donnees.php` ;

— et le graphique à l'adresse `http://tpe.teguad.ovh/graphique.php`.

Il est possible de passer un argument à ces deux pages pour limiter le nombre de données (par exemple, `graphique.php?limit=50` pour 50 données).

Date et heure	Température (°C)	Pression (hPa)
18/02/2015 à 12:07:14	24.3	1036.03
18/02/2015 à 12:07:10	24.3	1036.06
18/02/2015 à 12:07:07	24.3	1036.05
18/02/2015 à 12:07:03	24.3	1036.07
18/02/2015 à 12:07:00	24.3	1036.07
18/02/2015 à 12:06:56	24.3	1036.08
18/02/2015 à 12:06:53	24.3	1036.07
18/02/2015 à 12:06:49	24.3	1036.02
18/02/2015 à 12:06:46	24.3	1036.06
18/02/2015 à 12:06:42	24.3	1036.07
18/02/2015 à 12:06:39	24.3	1036.07
18/02/2015 à 12:06:35	24.33	1036.1
18/02/2015 à 12:06:32	24.3	1036.05
18/02/2015 à 12:06:28	24.35	1036.05
18/02/2015 à 12:06:25	24.36	1036.06
18/02/2015 à 12:06:21	24.36	1036.07
18/02/2015 à 12:06:18	24.36	1036.02
18/02/2015 à 12:06:14	24.4	1036.07
18/02/2015 à 12:06:11	24.4	1036.06
18/02/2015 à 12:06:07	24.4	1036.05

FIGURE 6.1 – Aperçu du tableau finalisé

FIGURE 6.2 – Page d'accueil du site

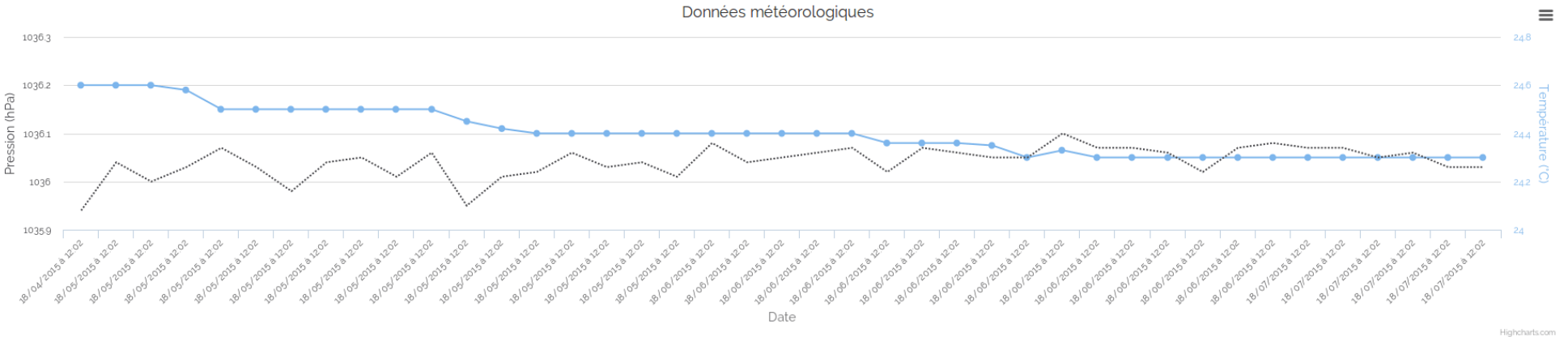


FIGURE 6.3 – Graphique généré avec 40 relevés

Conclusion

Nous avons donc réussi à faire fonctionner et à mettre en relation chacun des éléments composant notre système. Ceci montre que nous avons atteint les objectifs que nous nous étions fixés.

Cependant, nous avons subi divers échecs. Par exemple, notre capteur n'avait pas une bonne fidélité et nous donnait des résultats étranges compte tenu de notre connaissance *a priori* de l'environnement. Nous avons alors décidé de changer de capteur. De même, notre programme Arduino nécessitait — et nécessite toujours — un ordinateur pour fonctionner et effaçait l'intérêt du système

Nous pouvons cependant toujours faire évoluer notre projet en ajoutant des capteurs pour obtenir d'autres données (par exemple, l'humidité ou la luminosité), en étanchéifiant l'Arduino pour lui permettre de résister au milieu extérieur, ou encore en reliant plusieurs Arduino au serveur pour relever les données météorologiques à des endroits différents. Nous pourrions de même effectuer les relevés avec des capteurs plus précis, bien que notre utilisation ne le requiert pas.

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] Mathieu Nebra. *Apprenez à créer votre site Web avec HTML5 et CSS3*. 2015. URL : http://openclassrooms.com/courses/apprenez-a-creer-votre-site-web-avec-html5-et-css3.

[7] Mathieu Nebra. *Concevez votre site avec PHP et MySQL*. 2015. URL : http://openclassrooms.com/courses/concevez-votre-site-web-avec-php-et-mysql.

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

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

Annexe A

Code complet pour l'Arduino

Note : Ce code est configuré pour fonctionner en réseau local.

```
/* 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 15 * 60 * 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

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
```

```
43      Serial.print("Mon adresse IP : ");
44      Serial.println(Ethernet.localIP());
45    }
46  }
47
48  void loop()
49  {
50    /* Affichage des résultats des requêtes */
51    if (DEBUG)
52    {
53      while (client.available())
54      {
55        char c(client.read());
56        Serial.print(c);
57      }
58    }
59
60    if (!client.connected() && dernierConnecte)
61    {
62      Serial.println("Déconnexion.");
63
64      client.stop();
65    }
66
67    /* moyennes : tableau de 2 cases
68      - 1re : moyenne de la température
69      - 2de : moyenne de la pression */
70    float moyennes[2] = {};
71
72    /* Relevés des données */
73    for (int i(0) ; i < NB_RELEVES ; i++) // On fait la somme de 10 relevés de température
      ↪  et de pression.
74    {
75      moyennes[0] += capteur.bmp085GetTemperature(capteur.bmp085ReadUT()); // Température
76      moyennes[1] += capteur.bmp085GetPressure(capteur.bmp085ReadUP()); // Pression
77
78      delay(TEMPS);
79    }
80
81    moyennes[0] = moyennes[0] / NB_RELEVES;
82    moyennes[1] = moyennes[1] / NB_RELEVES / 100; // Division par 100 pour convertir les
      ↪  Pa en hPa
83
84    /* Affichage des valeurs */
85    if (DEBUG)
86    {
87      Serial.print("Moyennes : ");
88      Serial.print(moyennes[0]);
89      Serial.print(" °C et ");
90      Serial.print(moyennes[1]);
91      Serial.println(" hPa");
92    }
93
94    if (!client.connected() && (millis() - tempsDerniereConnexion > INTERVALLE))
95    {
96      if (client.connect(serveur, 80)) // Connexion au port 80
97      {
98        if (DEBUG)
99          Serial.println("Connexion.");
```

```
100
101       /* Envoi de la requête */
102       client.print("GET /arduino.php?pass=123&temperature=");
103       client.print(moyennes[0]);
104       client.print("&pression=");
105       client.print(moyennes[1]);
106       client.println(" HTTP/1.1");
107       client.println("Host: 192.168.1.2");
108       client.println("User-Agent: Arduino Ethernet");
109       client.println("Connection: close");
110       client.println();
111
112       tempsDerniereConnexion = millis();
113     }
114     else
115     {
116       if (DEBUG)
117       {
118         Serial.println("Erreur de connexion.");
119         Serial.println("Déconnexion.");
120       }
121
122       client.stop();
123     }
124   }
125
126   dernierConnecte = client.connected();
127 }
```

Annexe B

Code complet du site Web

Le code complet du site Web se trouve ci-dessous. Il est également téléchargeable sur notre dépôt GitHub :

http://github.com/TPE-Datalogger-Arduino/Site

B.1 Fichiers d'inclusion

`includes/settings.php`

C'est le fichier comportant les paramètres principaux du site comme le serveur de la base, le mot de passe, ...

```php
<?php
/* Fichier de configuration */

define("SERVEUR_BDD", "localhost");
define("BDD", "tpe");
define("LOGIN", "root");
define("MDP", "root");
```

`includes/connect.php`

C'est le fichier qui permet de se connecter à la base.

```php
<?php

require 'settings.php';

try
{
    $GLOBALS['db'] = new PDO('mysql:dbname='.BDD.';host='.SERVEUR_BDD, LOGIN, MDP,
    ↪  array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING));
}
catch(PDOException $e)
{
    die('Impossible de se connecter à MySQL : '.$e->getMessage());
}
```

`includes/sql.php`

C'est le fichier qui contient toutes les fonctions pour extraire les données de la base. (La fonction pour exporter en CSV n'a pas été expliquée mais elle est relativement simple.)

```php
<?php
/* Fichier contenant tous les fonctions pour accéder à la base de données */

require 'connect.php'; // Inclusion du fichier pour la connexion à la base

/* Fonction qui crée et affiche le tableau
   - $limit le nombre de donneés que le tableau doit afficher */
function table($limit)
{
    $db = $GLOBALS['db'];
?>
    <table>
        <thead>
            <tr>
                <td>Date et heure</td>
                <td>Température (&deg;C)</td>
                <td>Pression (hPa)</td>
            </tr>
        </thead>
        <tbody>
<?php
    $query = $db->query('SELECT temps, temperature, pression FROM `datalog_meteo` ORDER BY id DESC LIMIT '.$limit); // Requête SQL avec LIMIT
    $meteo = $query->fetchAll(PDO::FETCH_ASSOC);

    foreach($meteo as $data) // Parcours des données
    {
        $temps = preg_replace('#(\d{4})-(\d{2})-(\d{2}) (\d{2}:\d{2}:\d{2})#', '$3/$2/$1 à $4', $data['temps']); // Conversion du temps
        /* Nombre décimal avec une virgule */
        $temperature = preg_replace('/\./', ',', $data['temperature']);
        $pression = preg_replace('/\./', ',', $data['pression']);

        /* Affichage des donneés : création d'une ligne */
        echo '<tr>';
            echo '<td>'.$temps.'</td>';
            echo '<td>'.$temperature.'</td>';
            echo '<td>'.$pression.'</td>';
        echo "</tr>\r\n";
    }
?>
        </tbody>
    </table>
<?php
}

/* Fonction qui retourne la dernière valeur */
function lastData()
{
    $db = $GLOBALS['db'];

    $query = $db->query('SELECT * FROM `datalog_meteo` ORDER BY id DESC LIMIT 1');
    $meteo = $query->fetch(PDO::FETCH_ASSOC);

    return array(
        'temps' => preg_replace('#(\d{4})-(\d{2})-(\d{2}) (\d{2}:\d{2}:\d{2})#', '$3/$2/$1<br>$4', $meteo['temps']),
        'temperature' => preg_replace('/\./', ',', $meteo['temperature']),
        'pression' => preg_replace('/\./', ',', $meteo['pression'])
```

```php
    );
}

/* Fonction qui exporte les données au format CSV
   - $limit le nombre de donneés que le tableau doit afficher */
function exportCSV($limit)
{
    $db = $GLOBALS['db'];

    $query = $db->query('SELECT * FROM (SELECT id, temps, temperature, pression FROM `datalog_meteo` ORDER BY id DESC LIMIT '.$limit.') AS sq ORDER BY id ASC'); // Requête SQL
    $meteo = $query->fetchAll(PDO::FETCH_ASSOC);

    /* Type du fichier */
    header('Content-Type: text/csv; charset=utf-8');
    header('Content-Disposition: attachment; filename=Donnees.csv');

    echo "Temps;Température;Pression\r\n"; // Affichage des données

    foreach($meteo as $data) // Parcours des données
    {
        $temps = preg_replace('#(\d{4})-(\d{2})-(\d{2}) (\d{2}:\d{2}:\d{2})#', '$3/$2/$1 $4', $data['temps']);
        $temperature = preg_replace('/\./', ',', $data['temperature']);
        $pression = preg_replace('/\./', ',', $data['pression']);

        echo $temps.";".$temperature.";".$pression."\r\n"; // Affichage des données
    }

    echo "\r\n";
}

/* Insère les données $data dans la table datalog_meteo */
function insert($data)
{
    $db = $GLOBALS['db'];
    $query = $db->prepare('INSERT INTO datalog_meteo(temps, temperature, pression) VALUES(NOW(), ?, ?)');
    $query->execute(array(
        $data['temperature'],
        $data['pression']
    ));
}

/* Fonction qui génère le script pour le graphique */
function chart($limit)
{
    /* Pour convertir le temps */
    function formatTime($value)
    {
        return preg_replace('#(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})#', 'Date.UTC($1, $2, $3, $4, $5, $6)', $value);
    }

    $db = $GLOBALS['db'];
    $temps = array();
    $temperature = array();
    $pression = array();
```

```
111
112     $query = $db->query('SELECT * FROM (SELECT id, temps, temperature, pression FROM
        ↪  `datalog_meteo` ORDER BY id DESC LIMIT '.$limit.') AS sq ORDER BY id ASC'); //
        ↪  Requête SQL
113     $meteo = $query->fetchAll (PDO::FETCH_ASSOC);
114
115     /* Association des variables avec les valeurs de la base */
116     foreach($meteo as $data)
117     {
118         $temps[] = $data['temps'];
119         $temperature[] = $data['temperature'];
120         $pression[] = $data['pression'];
121     }
122
123     $max = sizeof($temps);
124 ?>
125     <script>
126         $(function () {
127             Highcharts.setOptions({
128                 lang: {
129                     decimalPoint: ',' // Nombre décimal avec une virgule
130                 }
131             });
132             $('#graphique').highcharts({
133                 chart: { // Style du graphique
134                     zoomType: 'xy',
135                     backgroundColor: '#fcfcfc',
136                     style: {
137                         font: 'inherit'
138                     }
139                 },
140                 title: { // Titre
141                     text: 'Données météorologiques'
142                 },
143                 xAxis: { // Abscisses
144                     categories: [
145                         <?php
146                         for ($i = 0 ; $i < $max ; $i++)
147                         {
148                             echo formatTime($temps[$i]).", ";
149                         }
150                         echo formatTime(end($temps));
151                         ?>
152                     ],
153                     title: {
154                         text: 'Date'
155                     },
156                     type: 'datetime',
157                     labels: {
158                         formatter: function() {
159                             return Highcharts.dateFormat('%d/%m/%Y à %H:%M:%S',
                                ↪  this.value);
160                         }
161                     }
162                 },
163                 yAxis: [ // Ordonnées
164                     { // Températures
165                         labels: {
166                             style: {
```

```
                                color: Highcharts.getOptions().colors[0]
                            }
                        },
                        title: {
                            text: 'Température (\u00B0C)',
                            style: {
                                color: Highcharts.getOptions().colors[0]
                            }
                        },
                        opposite: true
                    }, { // Pressions
                        gridLineWidth: 0,
                        title: {
                            text: 'Pression (hPa)',
                            style: {
                                color: Highcharts.getOptions().colors[1]
                            }
                        },
                        labels: {
                            style: {
                                color: Highcharts.getOptions().colors[1]
                            }
                        }
                    }
                ],
                tooltip: { // Boîte pour voir les informations
                    shared: true,
                    formatter: function() {
                        return Highcharts.dateFormat('%d/%m/%Y à %H:%M:%S', this.x) +
                         ↪  '<br>' +
                            '<span style="color: ' + Highcharts.getOptions().colors[0] +
                             ↪  '">\u25CF</span> Température : ' + this.points[0].y + '
                             ↪  \u00B0C<br>' +
                            '<span style="color: ' + Highcharts.getOptions().colors[1] +
                             ↪  '">\u25C6</span> Pression : ' + this.points[1].y + '
                             ↪  hPa';
                    },
                    style: {
                        fontSize: '10pt'
                    }
                },
                legend: {
                    enabled: false
                },
                series: [ // Données
                    { // Températures
                        name: 'Température',
                        yAxis: 0,
                        data: [
                            <?php
                            for ($i = 0 ; $i < $max ; $i++)
                            {
                                echo $temperature[$i].", ";
                            }
                            echo end($temperature);
                            ?>
                        ]
                    }, { // Pression
                        name: 'Pression',
```

```
                        yAxis: 1,
                        marker: {
                            enabled: false
                        },
                        dashStyle: 'shortdot',
                        data: [
                            <?php
                            for ($i = 0 ; $i < $max ; $i++)
                            {
                                echo $pression[$i].", ";
                            }
                            echo end($pression);
                            ?>
                        ]
                    }
                ]
            });
        });
    </script>
<?php
}
```

`includes/head.php`

C'est l'en-tête HTML de toutes les pages.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title><?php echo $title; ?> &lsaquo; TPE &mdash; Releveur de données météorologiques</title>
        <meta name="viewport" content="width=device-width, minimum-scale=1, initial-scale=1">
        <link href="http://fonts.googleapis.com/css?family=Ubuntu+Mono:400%7cRaleway:400,300,700" rel="stylesheet" type="text/css">
        <link href="/statics/styles/main.css" rel="stylesheet" type="text/css">
        <link href="http://maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css" rel="stylesheet">
        <script src="/statics/scripts/mobile-menu.js" type="application/javascript"></script>
    </head>

    <body>
        <div id="page">
            <header id="bandeau">
                <div id="nom-site">
                    <a href="/">TPE &mdash; Releveur de données météorologiques</a>
                </div>

                <div id="bouton">
                    <div class="barre"></div>
                    <div class="barre"></div>
                    <div class="barre"></div>
                </div>

                <nav id="navigation">
```

```
                    <ul>
                        <li><a href="/donnees.php">Données brutes</a></li>
                        <li><a href="/graphique.php">Graphique</a></li>
                        <li><a href="/dossier.php">Dossier</a></li>
                    </ul>
                </nav>
            </header>

            <section>
```

`includes/footer.php`

De-même, le pied de page.

```
            </section>

            <footer>
                Téofil <span class="nom">Adamski</span>,
                Rémi <span class="nom">Bruyère</span>,
                Clément <span class="nom">Bidault</span>
                et Sammy <span class="nom">Plat</span> &mdash; 2015
                <a href="https://github.com/TPE-Datalogger-Arduino"
                 ↪  class="logo-github"><i class="fa fa-github"></i></a>
            </footer>
        </div>
    </body>
</html>
```

B.2 Style et script pour le site

`statics/styles/main.css`

C'est le fichier CSS principale régissant toute l'apparence du site.

```
/* Feuille de style du site web du TPE
   Auteurs : Téofil Adamski, Rémi Bruyère, Clément Bidault et Sammy Plat

   Jeu de couleurs provenant de
 ↪  https://color.adobe.com/fr/Copie-de-Flat-design-colors-1-color-theme-4079805/ */

/* Style général
   ============= */

/* Reset */
*
{
    margin: 0;
    padding: 0;
    border: 0;
}

/* Corps du document */
body
{
    font: 11pt "Raleway", sans-serif;
    color: #222;
}

```

```
/* Page entière */
#page
{
    background-color: #fcfcfc;
    width: 100%;
}

/* Bandeau */
#bandeau
{
    position: fixed;
    left: 0;
    right: 0;
    top: 0;
    padding: 1rem 3rem;
    background-color: #efc94c;
    box-shadow: 0px -3px 12px #efc94c;
}

#bandeau #nom-site
{
    float: left;
    width: 50%;
    font-size: 1.5rem;
    font-weight: 300;
}

#bandeau #navigation
{
    float: right;
    line-height: 2em;
}

#bandeau a
{
    color: inherit;
    text-decoration: none;
}

#bandeau a:hover
{
    color: #334d5c;
}

#bandeau ul
{
    list-style-type: none;
}

#bandeau ul li
{
    float: left;
    padding-left: 2rem;
}

/* Section */
section
{
    margin-top: 4.5em;
```

```
}

section header
{
    font-size: 1.5rem;
    border-bottom: 1px solid #eee;
    padding: 1rem 10%; /* 10 % = 1/2 × (100 - width(#contenu)) */
    margin-bottom: 2rem;
}

/* Contenu */
#contenu
{
    width: 80%;
    margin: auto
}

/* Pied-de-page */
footer
{
    padding: .5rem;
    margin-top: 2rem;
    border-top: 1px dotted #efc94c;
    font-style: italic;
    text-align: right;
}

.logo-github
{
    padding-left: 1rem;
    vertical-align: middle;
    color: initial;
    font-size: 1.5em;
}

/* Paragraphes */
section p
{
    margin-bottom: 1ex;
    text-align: justify;
    line-height: 1.4rem;
}

/* Liens */
a
{
    color: #334d5c;
}

a:hover
{
    text-decoration: none;
    color: #df5a49;
}

/* Titres */
section header h1
{
    font-weight: normal;
```

```
142 }
143
144 section h2, section h3, section h4
145 {
146     margin: 1em 0 1ex;
147 }
148
149 /* Liste */
150 section ul
151 {
152     margin-left: 2rem;
153     list-style-type: circle;
154 }
155
156 section li
157 {
158     margin-bottom: .25rem;
159 }
160
161 /* Codes */
162 pre, code
163 {
164     font-family: "Ubuntu Mono", Courier, monospace;
165 }
166
167 /* Citations */
168 blockquote
169 {
170     margin: 1rem;
171     padding: 0 1rem;
172     border-left: 5px solid #45b29d;
173     border-right: 5px solid #45b29d;
174     border-radius: 10px;
175 }
176
177 /* Tableaux */
178 section table
179 {
180     margin: 1rem auto;
181     border-spacing: 0;
182     text-align: center;
183     border-top: 1px solid #efc94c;
184     border-bottom: 1px solid #efc94c;
185 }
186
187 section thead
188 {
189     background-color: #efc94c;
190 }
191
192 section table td
193 {
194     padding: 5px 10px;
195 }
196
197 table td:first-child, table th:first-child
198 {
199     border-left: 1px solid #efc94c;
200 }
```

```
201
202 table td, table th
203 {
204     border-right: 1px solid #efc94c;
205 }
206
207 table tbody tr:nth-child(2n+1)
208 {
209     background-color: rgba(238, 223, 39, .1);
210 }
211
212 /* Images */
213 section img
214 {
215     margin: 1rem auto;
216     max-width: 100%;
217 }
218
219 /* Graphique */
220 #graphique
221 {
222     width: 120%;
223     margin-left: -10%;
224 }
225
226 /* Éléments des formulaires */
227 form
228 {
229     margin: 2rem 0;
230 }
231
232 button, input[type=text], input[type=number], select
233 {
234     padding: 5px;
235     font: inherit;
236     border-radius: 5px;
237 }
238
239 button
240 {
241     background-color: #df5a49;
242     color: white;
243 }
244
245 button:hover
246 {
247     cursor: pointer;
248     background-color: #334d5c;
249 }
250
251 input[type=text], select
252 {
253     background-color: transparent;
254 }
255
256
257 input[type=text], input[type=number]
258 {
259     border: 1px solid #df5a49;
```

```
260 }
261
262 /* Noms de famille */
263 .nom
264 {
265     font-variant: small-caps;
266 }
267
268 /* Barre d'avancement */
269 .avancement
270 {
271     width: 100%;
272     height: 1.5rem;
273     overflow: hidden;
274     border-radius: 5em;
275     background-color: #eee;
276 }
277
278 .avancement span
279 {
280     display: block;
281     background-color: #df5a49;
282     height: inherit;
283 }
284
285 .avancement span em
286 {
287     display: block;
288     padding-top: .25em;
289     text-align: center;
290     font-style: normal;
291     color: white;
292 }
293
294 /* Dernier relevé */
295 .temps, .temperature, .pression
296 {
297     display: inline-block;
298     width: 33%;
299     text-align: center;
300 }
301
302 .valeur
303 {
304     display: block;
305     font-size: 2rem;
306 }
307
308 /* Media-queries
309    ============== */
310
311 /* Bouton pour le menu mobile */
312 #bouton
313 {
314     display: none;
315     float: right;
316     margin: .5em;
317     cursor: pointer;
318 }
```

```
319
320 .barre
321 {
322     width: 18px;
323     height: 3px;
324     background-color: #222;
325 }
326
327 .barre + .barre
328 {
329     margin-top: 3px;
330 }
331
332 /* Version petit écran */
333 @media screen and (max-width: 979px)
334 {
335     #bandeau
336     {
337         padding: 1rem;
338     }
339
340     #bandeau #nom-site
341     {
342         width: calc(100% - 2em);
343         overflow: hidden;
344         white-space: nowrap;
345         text-overflow: ellipsis;
346     }
347
348     #bandeau #navigation
349     {
350         display: none;
351     }
352
353     #bandeau a:hover
354     {
355         padding: 0;
356         border: none;
357     }
358
359     .affiche
360     {
361         display: block !important;
362     }
363
364     #bouton
365     {
366         display: inline-block;
367     }
368
369     #bandeau #navigation
370     {
371         float: left;
372         width: 100%;
373         line-height: initial;
374         margin-top: .5rem
375     }
376
377     #bandeau ul li
```

```
    {
        width: 100%;
        padding: .5rem 0;
    }

    #bandeau #navigation a
    {
        display: block;
        width: 100%;
    }

    .temps, .temperature, .pression
    {
        width: 100%;
        margin-bottom: 1rem;
    }
}

/* Version mobile */
@media screen and (max-width: 768px)
{
    #contenu
    {
        width: 90%;
    }

    section header
    {
        padding: 1rem 5%;
    }
}
```

`statics/scripts/mobile-menu.js`

C'est un petit fichier JavaScript permettant de créer un menu déroulant lorsqu'on accède au site depuis un *smartphone*.

```
/* Script pour afficher le menu mobile */
window.onload = function () {
    "use strict";

    var bouton = document.getElementById("bouton"),
        nav = document.getElementById("navigation");

    bouton.onclick = function () {
        if (nav.className === "affiche") {
            nav.removeAttribute("class");
        } else {
            nav.className = "affiche";
        }
    };
};
```

B.3 Pages pour ajouter ou visualiser les données

`arduino.php`

C'est le fichier qu'appele Arduino pour ajouter les données.

```php
<?php

require 'includes/sql.php'; // Inclusion du fichier contenant les commandes pour accéder à la base

$data = array(); // Création d'un tableau contenant les données

if (isset($_GET['pass']) && $_GET['pass'] == '123') // On test si le mot de passe est bon.
{
    $data['temps'] = 'NOW()'; // Le temps est maintenant.
    /* Si il y a une température donnée, alors on la convertit. Sinon, rien. */
    $data['temperature'] = (isset($_GET['temperature'])) ?
                                htmlspecialchars($_GET['temperature']) : NULL;
    /* Idem pour la pression */
    $data['pression'] = (isset($_GET['pression'])) ?
                                htmlspecialchars($_GET['pression']) : NULL;

    insert($data); // Et on insert les données dans la base.
}
else
{
    echo 'Mauvais mot de passe !';
}
```

`index.php`

C'est la page d'accueil. Elle affiche le dernier relevé.

```php
<?php

$title = 'Accueil';

include 'includes/sql.php';
include 'includes/head.php';

$lastData = lastData();

?>
    <header>
            <h1>Bienvenue sur notre site</h1>
    </header>

    <div id="contenu">
        <p>Notre projet a pour but de créer un releveur de données météorologiques qui puisse transmettre les données sur un site Web. Ce site tourne sous nginx avec une base de données MariaDB le tout codé en PHP.</p>

        <p>Vous pouvez lire le dossier <a href="dossier.php">ici</a>.</p>

        <h2>Dernier relevé</h2>

        <div class="dernier-releve">
            <div class="temps">
                Date
                <span class="valeur"><?php  echo $lastData['temps'];?></span>
            </div>

            <div class="temperature">
```

```
                Température
                <span class="valeur"><?php echo $lastData['temperature']; ?>
                ↪ &deg;C</span>
            </div>

            <div class="pression">
                Pression
                <span class="valeur"><?php echo $lastData['pression']; ?> hPa</span>
            </div>
        </div>
    </div>
<?php include 'includes/footer.php';
```

`donnees.php`

C'est la page pour afficher le tableau de données.

```
<?php

$title = 'Tableau';

include 'includes/sql.php';
include 'includes/head.php';

/* Valeur limite passé dans l'URL ; si elle est négative ou n'existe pas, on la met à 20
↪ */
$limit = isset($_GET['limite']) && $_GET['limite'] > 0 ?
↪ htmlspecialchars($_GET['limite']) : 20;

?>
    <header>
        <h1>Tableau des <?php echo $limit; ?> derniers relevés</h1>
    </header>

    <div id="contenu">
        <!-- Formulaire pour le nombre de relevé à afficher -->
        <form method="get" action="donnees.php">
            Quantité de données : <input type="number" name="limite" value="<?php echo
            ↪ $limit; ?>" pattern="\d+">
            <button type="submit">Go !</button>
        </form>

        <!-- Formulaire pour exporter au format CSV -->
        <form method="get" action="exporter-csv.php">
            Vous pouvez également exporter les données au format CSV pour les traiter
            ↪ dans un tableur. <input type="number" name="limite" value="<?php echo
            ↪ $limit; ?>" pattern="\d+">
            <button type="submit">Exporter</button>
        </form>

        <?php // Affichage du tableau
        table($limit); ?>
    </div>
<?php include 'includes/footer.php';
```

`exporter-csv.php`

C'est le fichier qui permet d'exporter les données au format CSV.

```php
<?php

$title = 'Graphique';

include 'includes/sql.php';

$limit = isset($_GET['limite']) && $_GET['limite'] > 0 ?
 ↪  htmlspecialchars($_GET['limite']) : 20;

exportCSV($limit);
```

`graphique.php`

C'est la page qui affiche le graphique.

```php
<?php

$title = 'Graphique';

include 'includes/sql.php';
include 'includes/head.php';

/* Valeur limite passé dans l'URL ; si elle est négative ou n'existe pas, on la met à 20
 ↪  */
$limit = isset($_GET['limite']) && $_GET['limite'] > 0 ?
 ↪  htmlspecialchars($_GET['limite']) : 20;

?>
    <script
     ↪  src="http://cdnjs.cloudflare.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
    <script
     ↪  src="http://cdnjs.cloudflare.com/ajax/libs/highcharts/4.1.3/highcharts.js"></script>
    <script
     ↪  src="http://cdnjs.cloudflare.com/ajax/libs/highcharts/4.1.3/modules/exporting.js"></script>
    <?php // Insertion du script pour le graphique
    chart($limit); ?>

    <header>
        <h1>Graphique des <?php echo $limit; ?> derniers relevés</h1>
    </header>

    <div id="contenu">
        <!-- Formulaire pour le nombre de relevé à afficher -->
        <form method="get" action="graphique.php">
            Quantité de données : <input type="number" name="limite" value="<?php echo
             ↪  $limit; ?>" pattern="\d+">
            <button type="submit">Go !</button>
        </form>

        <div id="graphique"></div>
    </div>
<?php include 'includes/footer.php';
```

B.4 Autres pages

`403.php` et `404.php`

Ce sont les pages d'erreurs.

```php
<?php

$title = 'Erreur 403';

include 'includes/head.php';

?>
    <header>
        <h1>Accés interdit</h1>
    </header>

    <div id="contenu">
        Vous n'avez pas le droit d'accéder à cette page !
    </div>
<?php include 'includes/footer.php';
```

```php
<?php

$title = 'Erreur 404';

include 'includes/head.php';

?>
    <header>
        <h1>Page non trouvée</h1>
    </header>

    <div id="contenu">
        La page que vous demandez n'a pas été trouvée&hellip;
    </div>
<?php include 'includes/footer.php';
```