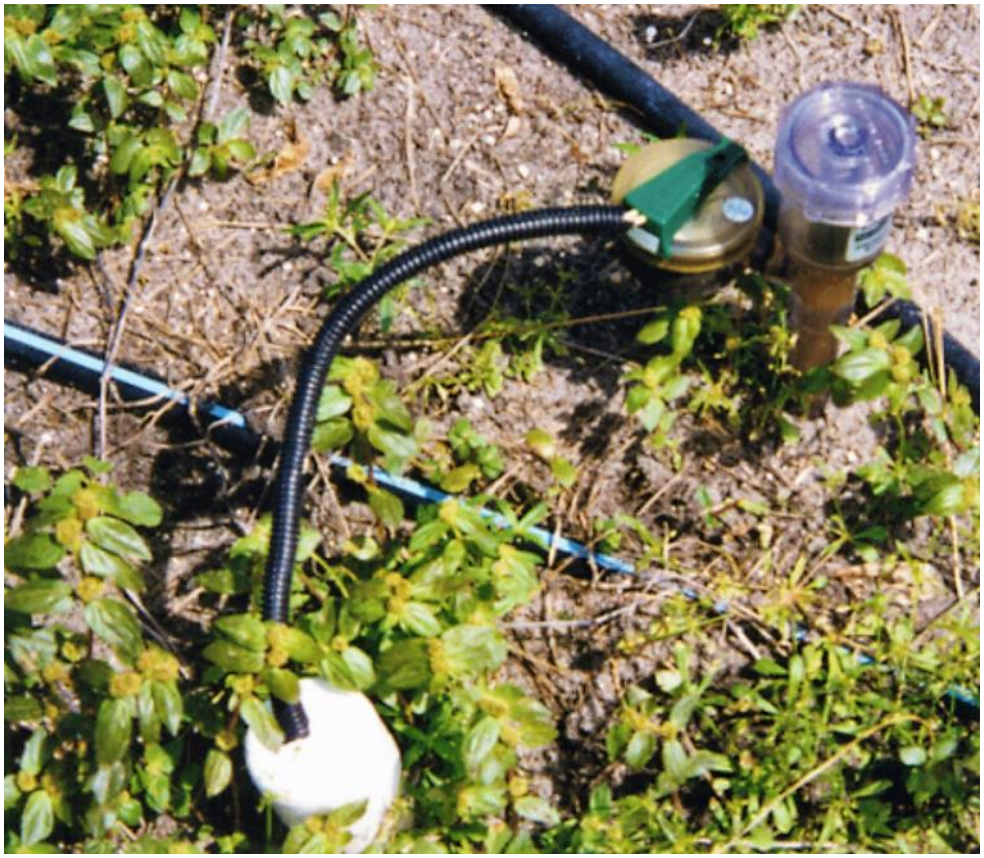
المدرسة الوطنية للعلوم التطبيقية
ⵜⵉⵎⵙⵙⵓⵜ ⵜⴰⵏⴰⵎⵓⵔⵜ ⵏ ⵜⵙⴻⵏⵙⵉⵡⵉⵏ ⵜⵉⵜⵜⴰⵏⵉⵜⵉⵏ
Ecole Nationale des Sciences Appliquées de Fès

جامعة سيدي محمد بن عبد الله
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵙⵉⴷⵉ ⵎⵓⵃⴰⵎⴻⴷ ⴱⵏ ⵄⴱⴷⵍⵍⴰⵀ
Université Sidi Mohamed Ben Abdellah

Génie des Systèmes Embarqués et Informatique Industrielle

Projet de Fin d'Année



Smart Irrigation

Réalisé par :

Ismaili El yazid

Dadaou Abderrahmane

Encadré par :

Ezzouhairi Abdellatif

Année Universitaire 2020-2021

Table de matière

Introduction

Chaque jour, la consommation d'eau extérieure à elle seule représente des dizaines de milliards de litres d'eau, principalement pour l'irrigation du paysage. Jusqu'à 50% de cette eau est gaspillée en raison de l'arrosage supérieur causé par des inefficacités dans les méthodes et les systèmes d'irrigation traditionnels. **La technologie d'*irrigation intelligente* est la réponse.**

Dans la perspective d'une troisième révolution verte, l'application des technologies modernes de l'information et de la communication à l'agriculture revêt une grande importance. L'agriculture intelligente consiste à gérer l'eau, la lumière, l'humidité et la température dans le but de renforcer la qualité et la quantité de la production.

Les capteurs et les actionneurs peuvent être installés dans les exploitations agricoles afin de garantir une production agricole plus productive et durable.
L'automatisation et la robotique agricoles rendent les exploitations agricoles intelligentes.

Contrairement aux systèmes d'irrigation traditionnels qui fonctionnent selon un calendrier et des minuteries programmés prédéfinis. Dans le cadre de ce projet d'irrigation intelligente, le système contrôle la consommation d'eau et d'électricité d'une vanne, à travers deux débitmètres (pour calculer la variation du débit d'eau entre deux points distants) et un capteur de courant. En addition, un capteur de température et d'humidité est intégré pour la surveillance de la température et de l'humidité. Toutes ces grandeurs sont visualisées sur des graphiques en courbes pour fournir une distinction visuelle claire entre les jeux de données. La vanne est contrôlée à partir d'une interface web depuis n'importe quel navigateur internet dans n'importe quel dispositif informatique.

Ce rapport présente dans 3 chapitres, le contexte général de l'application ainsi sa conception et modélisation aux deux niveaux, matériel et logiciel, sans oublier la phase du teste des éléments disponibles construisant le projet.

L'application s'inscrit sous le projet de fin d'année, encadré par Mr. Ezzouhairi Abdellatif, dans l'Ecole Nationale des Sciences Appliquées de Fès.

Chapitre 1

Contexte générale du projet

Contexte

Une application ou un système avec une architecture client-serveur signifie qu'une ou plusieurs machines appelées clientes qui sont à la disposition des utilisateurs fonctionnent à partir d'un contacte ou une interrogation d'un serveur qui est généralement une carte ou un ordinateur plus ou moins puissant, contenant tous les codes, les services et les ressources nécessaires pour l'exécution de la tâche voulue par le client.

Cette solution présente des gains importants pour le niveau de la fiabilité et la sécurité de l'application, étant donné que les ressources sont centralisées dans le serveur qui garde un contrôle sur l'ensemble des permissions d'accès aux différents services et codes présenté pour chaque type de client, ainsi que cette architecture est modulaire de faite que la capacité du serveur et la capacité du client peuvent être changées d'une manière indépendante. Bien évidemment, ce modèle a besoin d'un réseau pour se fonctionner, car les clients connaissent le serveur à partir de son adresse IP et vis-versa.

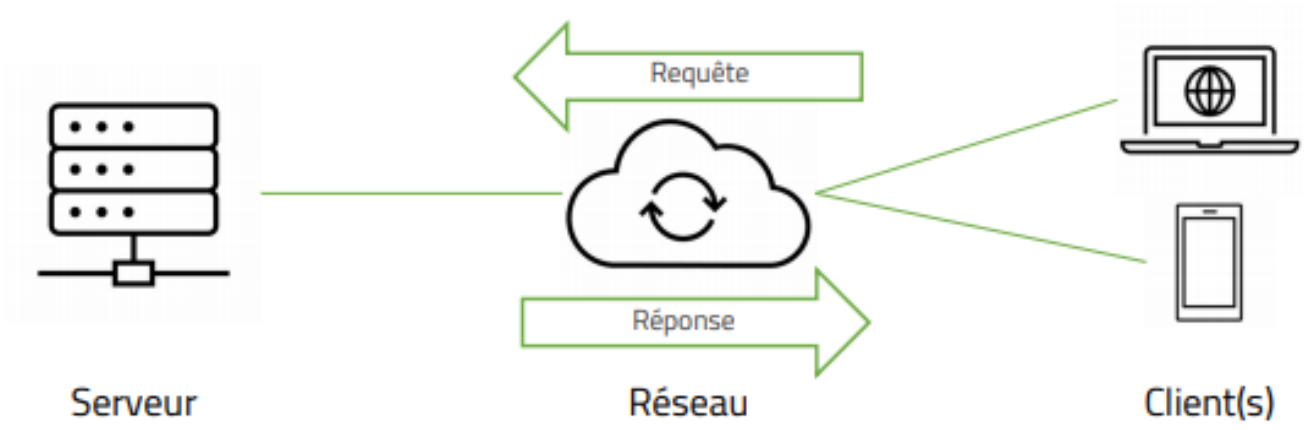


Figure 1 : Architecture Serveur-Client

On utilise le protocole **SSH** pour la communication au sein du réseau.

SSH, ou Secure Shell, est un protocole d'administration à distance qui permet aux utilisateurs de contrôler et de modifier leurs serveurs distants sur Internet. Le service a été créé en remplacement sécurisé du telnet non chiffré et utilise des techniques de chiffrement pour garantir que toutes les communications vers et depuis le serveur distant se produisent de manière chiffrée. Il fournit un mécanisme d'authentification d'un utilisateur distant, de transfert d'entrées du client vers l'hôte et de relais de la sortie vers le client.

L'avantage significatif offert par SSH par rapport à ses prédécesseurs est l'utilisation du cryptage pour assurer un transfert sécurisé des informations entre l'hôte et le client. Hôte fait référence au serveur distant auquel vous essayez d'accéder, tandis que le client est l'ordinateur que vous utilisez pour accéder à l'hôte.

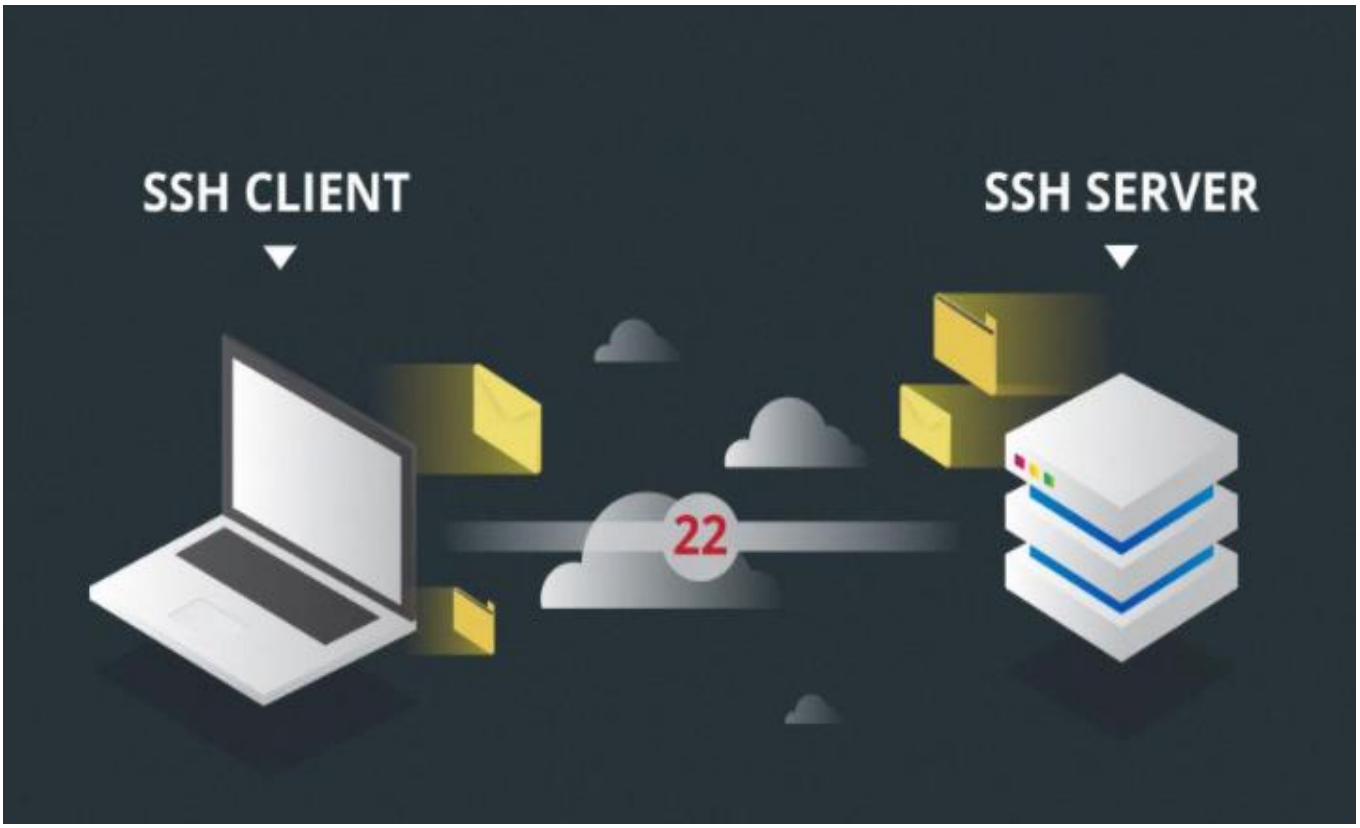


Figure 2 : SSH Secure SHell Serveur-Client

Le projet de contrôle d'une vanne d'eau adopte l'architecture client/serveur pour plusieurs raison, notamment la nécessité d'une interface homme machine assez développée pour faciliter le contrôle de la vanne. Ainsi, ce contrôle peut être fait à partir de plusieurs périphériques connectés au réseau, et non pas une interface figée dans un emplacement fixe.

De plus, les fonctionnalités à implémenter dans le système nécessite un périphérique puissant qui peut exécuter plusieurs codes en même temps, et qui nécessite des ressources matérielles importantes, donc le recourt à utiliser un serveur dans ce projet est une nécessité.

Cahier de charge

Le système, basé sur le modèle client-serveur, doit respecter le cahier de charge suivant:

• Pour la partie matérielle liée au serveur, le système fait l'acquisition des données de la variation du débit de l'eau, quantité de courant, températures et humidité, ainsi que le transfert des fichiers JSON au serveur. Comme il fait les traitements des données et la gestion de base de données.

• Pour la partie logicielle liée au client, l'interface graphique est une page Web accessible depuis un navigateur internet, qui donne au client une visualisation de la variation ou l'erreur du débit et de l'énergie électrique consommée, plus la température et l'humidité, sur des graphiques en courbes pour fournir une distinction visuelle claire entre les jeux de données. Le client peut aussi éteindre ou démarrer la vanne a tout moment à travers une deuxième page web.

Méthodologie

1. **L'expression du besoin:** Le système a pour but de faciliter au client le contrôle d'une vanne à l'aide des données acquises par les capteurs. Une partie matérielle va être installée à côté de la vanne, et une partie logicielle va être prête à utiliser dans un dispositif informatique chez le client.

2. **Spécification fonctionnelle:**

Pour la partie Serveur, elle doit gérer les fonctionnalités suivantes:

- Capturer la variation de débit et la quantité de courant, les stocker sur une base de données.

- Capturer la température et l'humidité et les stocker sur une base de données.

- Démarrer ou éteindre une vanne.

Pour la partie Client, elle doit contrôler les options suivantes:

- Voir la variation du débit de l'eau et l'énergie consommée par la vanne, en temps réel.

- Voir la température et l'humidité en temps réel

-Contrôler la vanne.

3. **Spécification technique:**

   Les technologies à employer dans le projet qui se base sur le modèle client-serveur peuvent se résumer en un dispositif informatique assez puissant qui va servir comme serveur, avec une partie qui gère les entrées et les sorties (GPIO), deux débitmètres, ainsi un capteur de courant et un capteur de la température et l'humidité. Pour la partie Client, c'est le choix de l'utilisateur d'accéder à l'interface avec n'importe quel dispositif avec connexion réseau et navigateur internet.

4. **Codage:**

   La partie du codage va être divisée en deux phases. La première est d'écrire les codes dans le Serveur qui vont contrôler l'ensemble des entrées/sorties (capteurs + vanne) disponibles pour le projet.

   Chaque E/S va avoir un code Python.

   La partie Client va être codée en deuxième lieu, après que les codes soient déjà fonctionnels en manière standalone dans le serveur.

5. **Test :**

   La partie du test doit être divisée en deux phases :

   La première est celle des tests unitaires où chaque E/S doit être testée indépendamment des autres composantes. Ainsi, Chaque option liée à cette E/S dans l'interface client va être testée indépendamment des autres options. La deuxième est le test d'intégration. C'est un test final de l'application en sa totalité avant la validation.

   - Dans ce projet, nous avons à notre disposition un capteur de température et d'humidité. Ce qui nous permettra à réaliser des tests unitaires sur cette composante.

Chapitre 2

Conception et Modélisation du projet

Environnement Technique du projet

Le projet, avec son modèle client-serveur, est basé sur une carte Raspberry Pi 3 Model B qui offre 40 pins qui vont servir à interfacer toutes les composantes nécessaires pour ce projet, et qui fonctionne avec un système d'exploitation Raspberry Pi OS (anciennement appelé Raspbian) basé sur Linux, ce qui offre un niveau d'abstraction plus élevé et une gestion de ressources entre processus plus adéquate pour le projet.

La carte va opérer comme un serveur, donc un serveur web doit s'installer dedans. On choisit pour cette mission ***Lighttpd*** qui est un serveur web minimal, adapté plus aux applications embarqués et nécessite mois de ressources que les autres solutions, notamment Apache.

Lighttpd est un logiciel de serveur Web (ou HTTP : Hypertext Transfer Protocol) sécurisé, rapide et flexible. C'est un logiciel libre écrit en C et distribué selon les termes de la licence BSD.

Sa rapidité vient du fait qu'il a une plus petite empreinte mémoire que d'autres serveurs HTTP ainsi qu'une gestion intelligente de la charge CPU. Beaucoup de langages, comme PHP, Perl, Ruby, Python sont supportés via FastCGI.

L'utilisation des interfaces FastCGI, SCGI et CGI pour des programmes externes permet d'écrire des applications Web dans n'importe quel langage habituellement utilisé sur les serveurs. PHP étant populaire, ses performances ont été particulièrement optimisées.

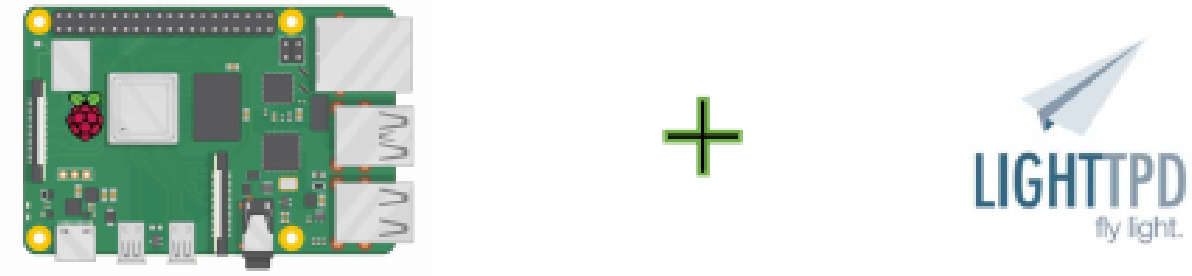


Figure 3 : Raspberry Pi 3 Model B et le serveur web Lighttpd

- Pour la mesure de la quantité de courant, on peut utiliser un capteur de courant. Ce capteur de courant à effet Hall est basé sur ACS712. Le capteur donne une mesure précise du courant pour les deux signaux AC et DC. Ceci permet de conserver la module isolée même si l'on mesure le courant d'un appareil alimenté par du 220V. Ce module ACS712 est calibré pour mesurer des intensités allant jusqu'à 30A.

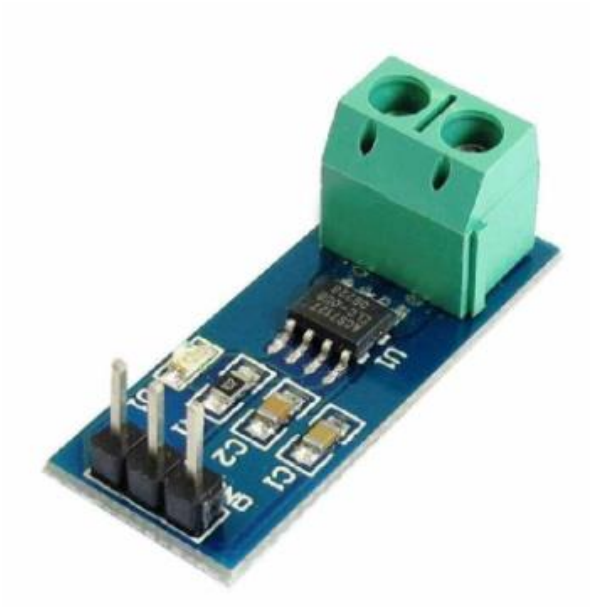


Figure 4 : capteur de courant ACS712 30A

- Pour la mesure de l'erreur ou la variation du débit de l'eau, pour la détection de toute fuite d'eau, on peut utiliser 2 débitmètres. Le premier sera proche au point de l'entrée et le deuxième sera proche au point de la sortie.

  Le débitmètre ou le compteur d'eau permet de faire le comptage de la quantité d'eau passant dans une canalisation. Grâce à une transmission magnétique du signal, seule la turbine est en contact avec le flux d'eau permettant ainsi aux différents axes et engrenages de travailler complètement à sec en évitant ainsi une usure et une corrosion prématurée du compteur.

  En considérant le décalage horaire, on calcule la différence entre les deux débits d'entrée et de sortie pour obtenir l'erreur en résultat.

Figure 5 : Débitmètre ou compteur d'eau

- Pour l'acquisition de la température et l'humidité, on utilise le capteur DHT11 qui est basé sur un protocole personnalisé sur un seul bus de données. Mais ce capteur utilise un protocole plus compliqué qu'un capteur 1-Wire, donc on va utiliser le driver fournit sous Python par la bibliothèque **Adafruit**.

  Le DHT11 est un capteur de température et d'humidité "deux en un".

Le capteur DHT11 est capable de mesurer des températures de 0 à +50°C avec une précision de +/- 2°C et des taux d'humidité relative de 20 à 80% avec une précision de +/- 5%. Une mesure peut être réalisée toutes les secondes.

le DHT11 est compatible à 3.3 volts et 5 volts (le fabricant recommande cependant de toujours alimenter le capteur en 5 volts pour avoir des mesures précises).

Pour résumer, voici les caractéristiques du capteur sous forme de tableau:

	DHT11
Humidité (relative %)	20 ~ 80%
Précision (humidité)	+/- 5%
Température	0 ~ +50°C
Précision (température)	+/- 2°C
Fréquence mesure max	1Hz (1 mesure par seconde)
Tension d'alimentation	3 ~ 5 volts
Stabilité à long terme	+/- 1% par an

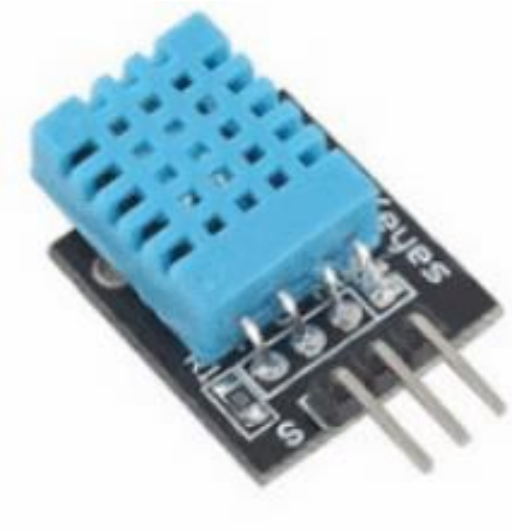


Figure 6 : Capteur DHT11 de température et d'humidité

Les mesures sont enregistrées dans la base de données « **temp_hum.db** ».

- Pour la partie de la base de données, on va installer et manipuler la bibliothèque **SQLite**, qui fournit une base de données embarqué légère, sans serveur qui écrit et lit directement dans et depuis le stockage interne.

  SQLite est une bibliothèque C qui fournit une base de données sur disque légère qui ne nécessite pas de processus serveur distinct et permet d'accéder à la base de données à l'aide d'une variante non standard du langage de requête SQL. Certaines applications peuvent utiliser SQLite pour le stockage de données internes.

  Pour faire l'interfaçage entre SQLite et Python, on utilise le module **sqlite3** qui fournit les fonctions de manipulation des bases de données depuis le code Python.

Figure 7 : SQLite3 et Python

- Pour la partie client, c'est une interface Web, créée avec les langages classiques HTML, CSS, JavaScript et PHP.

  Les deux premiers langages s'occupent du design de l'interface (Front-end), à l'aide du Framework ***Bootstrap***.

  Bootstrap est le framework HTML, CSS et JS le plus populaire pour développer des projets réactifs et mobiles sur le Web. Il rend le développement Web frontal plus rapide et plus facile. Il est fait pour les appareils de toutes formes et les projets de toutes tailles. Bootstrap est open source. Il est hébergé, développé et maintenu sur GitHub.

  Le code JavaScript contient les fonctions nécessaires pour la communication entre l'interface et le serveur. Les intermédiaires de cette communication sont des scripts PHP, qui permettent d'exécuter des commandes Linux directement sur le serveur et envoyer les résultats aux fonctions du JavaScript, pour mettre à jour l'interface (Back-end).

Figure 8 : Framework de BootStrap

Architecture générale du projet

Le projet adopte le modèle client-serveur, dont la carte Raspberry Pi est à la fois un moyen d'acquisition et le serveur lui-même. Lorsque le client navigue à la page Web dédiée pour l'application à partir d'un navigateur internet avec l'adresse IP du serveur, il se trouve devant cette barre de navigation au sommet de la fenêtre :

Figure 9 : Barre de menu ou MenuBar

L'interaction avec la barre de menu repose sur le clic. Chaque élément de la barre est un hyperlien qui nous emmène à une page web.

Dashboard : Cet hyperlien pointe vers l'URL « **fichier.html** », c'est la page web sur laquelle on visualise les 4 grandeurs captées par notre système (Température, humidité, erreur de débit et quantité de courant). Elle est divisée en 4 graphiques en courbes fournies par **Chart.JS** d'**AJAX** pour fournir une distinction visuelle claire entre les jeux de données.

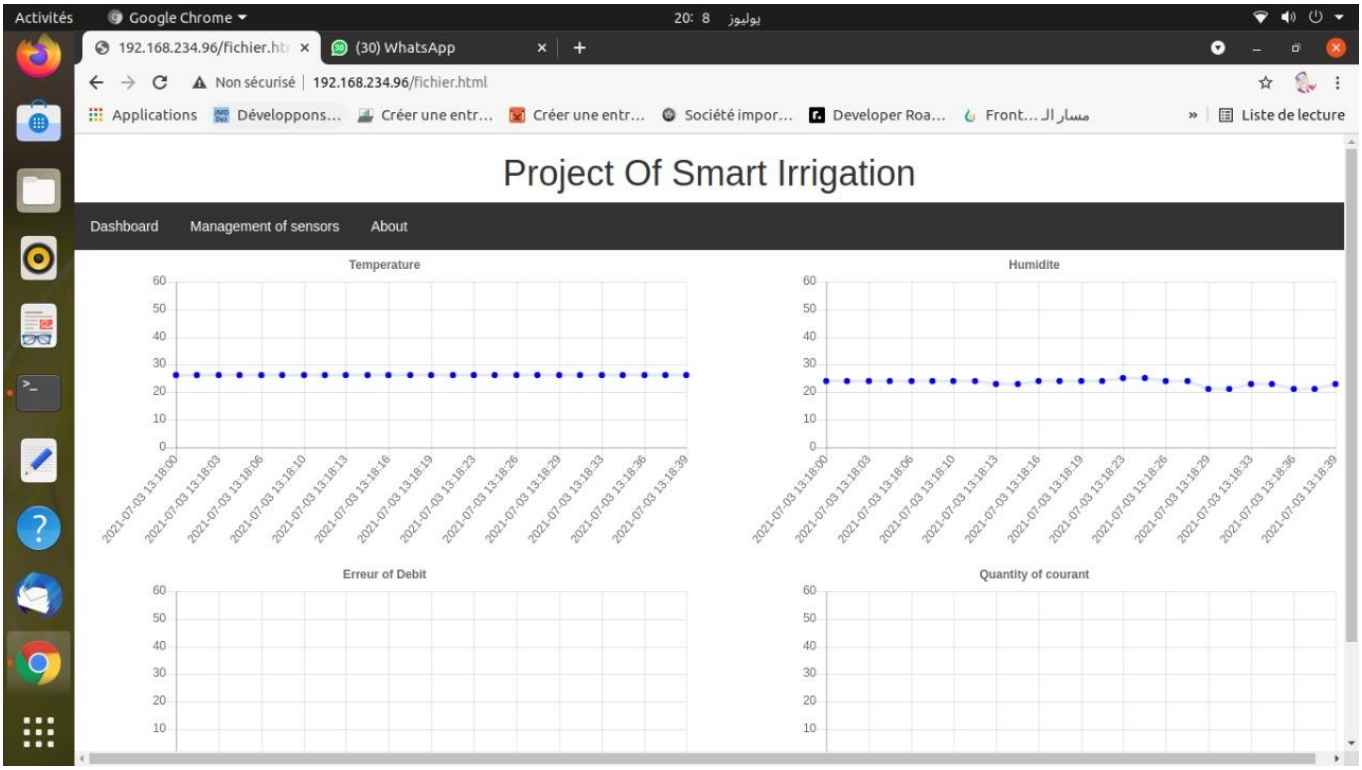


Figure 10 : Page Web "fichier.html"

- Ajax est une méthode utilisant différentes technologies ajoutées aux navigateurs web, et dont la particularité est de permettre d'effectuer des requêtes au serveur web et, en conséquence, de modifier partiellement la page web affichée sur le poste client sans avoir à afficher une nouvelle page complète. Cette architecture informatique permet de construire des applications Web et des sites web dynamiques interactifs. Ajax est l'acronyme d'asynchronous JavaScript and XML : JavaScript et XML asynchrones.

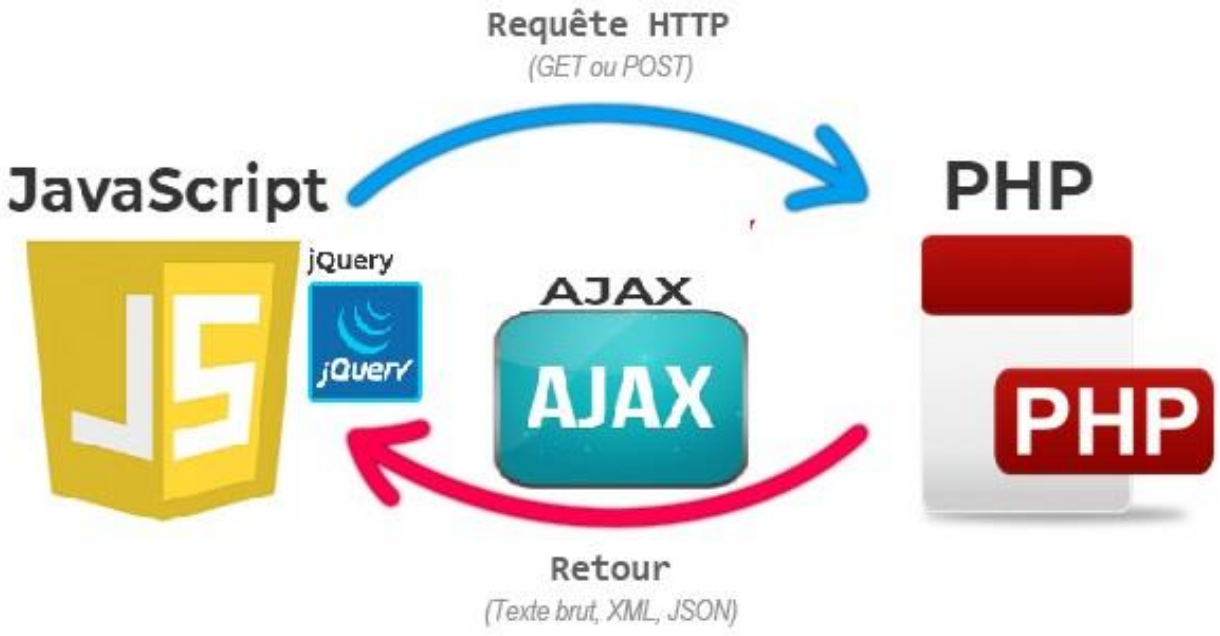


Figure 11 : AJAX Asynchronous JavaScript and XML

- Chart JS est simple mais flexible pour les concepteurs et les développeurs.

  Chart.js est une bibliothèque JavaScript open source sur Github qui vous permet de dessiner différents types de graphiques à l'aide de l'élément canvas HTML5.

  L'élément HTML5 offre un moyen simple et puissant de dessiner des graphiques à l'aide de JavaScript. Il peut être utilisé pour dessiner des graphiques, faire des compositions photo ou faire des animations simples (et pas si simples).

  Il est réactif dans la nature, ce qui signifie qu'il redessine le graphique lors du redimensionnement de la fenêtre pour une granularité d'échelle parfaite.

Le fichier **« fichier.html »** envoie les requêtes suivantes à partir des fonctions (JavaScript) définies dans le même fichier avec les scripts Back-end du PHP dans le fichier **« f.php »** :

- Une requête pour obtenir la température et l'humidité à partir du capteur DHT et l'afficher dans la page Web d'une manière périodique chaque 10 secondes.
- Une requête pour obtenir la quantité de courant consommée à partir du capteur de courant et l'afficher dans la page Web en temps réel.
- Une requête pour obtenir l'erreur du débit de l'eau à partir des deux débitmètres et l'afficher dans la page Web en temps réel.

Le fichier **« f.php »** extrait toutes les lignes de la base de données **SQLite3** sous la forme d'un tableau associatif, après il encode ce tableau, c'est-à-dire ; convertir le tableau PHP en représentation **JSON.**

- JSON signifie JavaScript Object Notation.

  JSON est un format léger pour le stockage et le transport de données.

  Il est souvent utilisé lorsque des données sont envoyées d'un serveur à une page Web. JSON est « auto descriptif » et facile à comprendre.

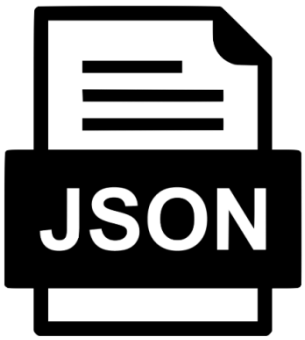


Figure 12 : JSON

Management of sensors :

Cet hyperlien pointe vers l'URL « **sensors.html** », c'est la page web sur laquelle le client peut démarrer ou éteindre/activer ou désactiver la vanne à partir de deux boutons, comme elle nous montre la figure suivante :

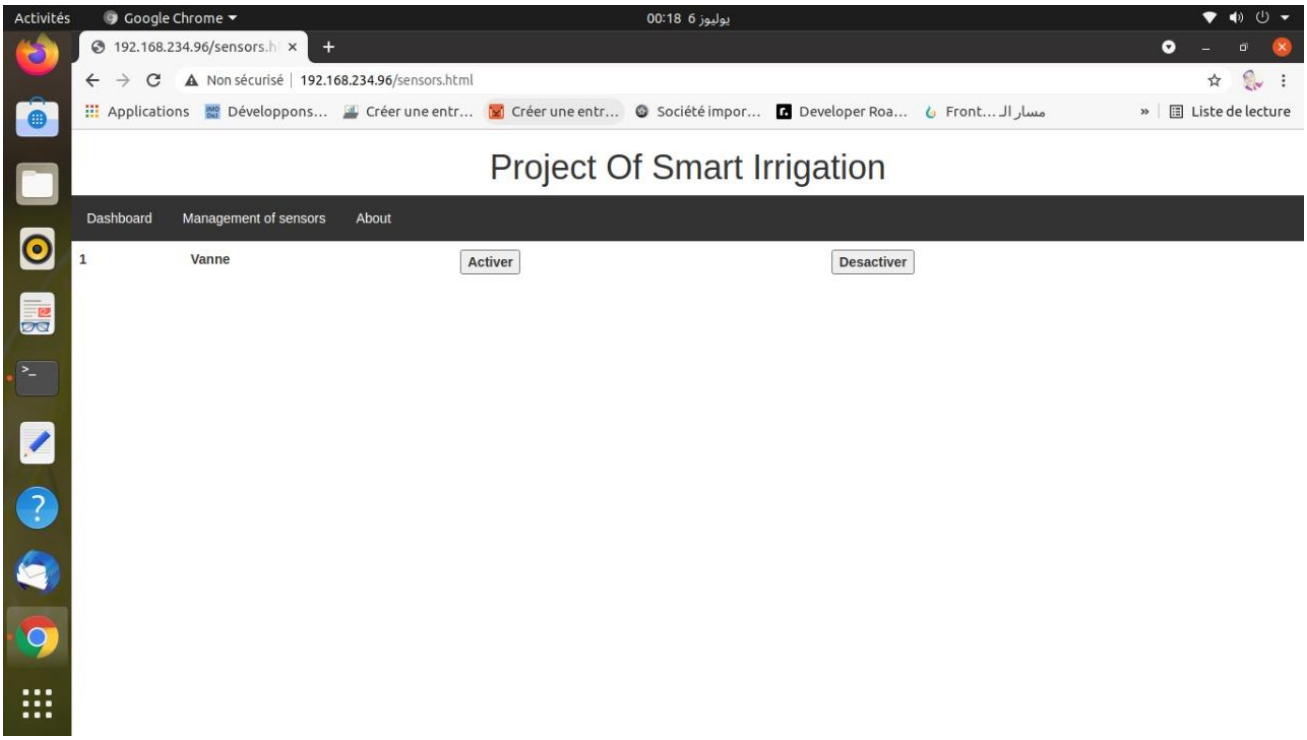


Figure 13 : Session Web "sensors.html"

A l'intervention de l'utilisateur, il peut exécuter ainsi les tâches suivantes:

- Activer/démarrer la vanne avec une alimentation 5V/3.3V.

- désactiver la vanne avec une coupure de courant.

A partir des scripts PHP du serveur, il envoie les réponses adéquates à chaque requête ou commande du client.

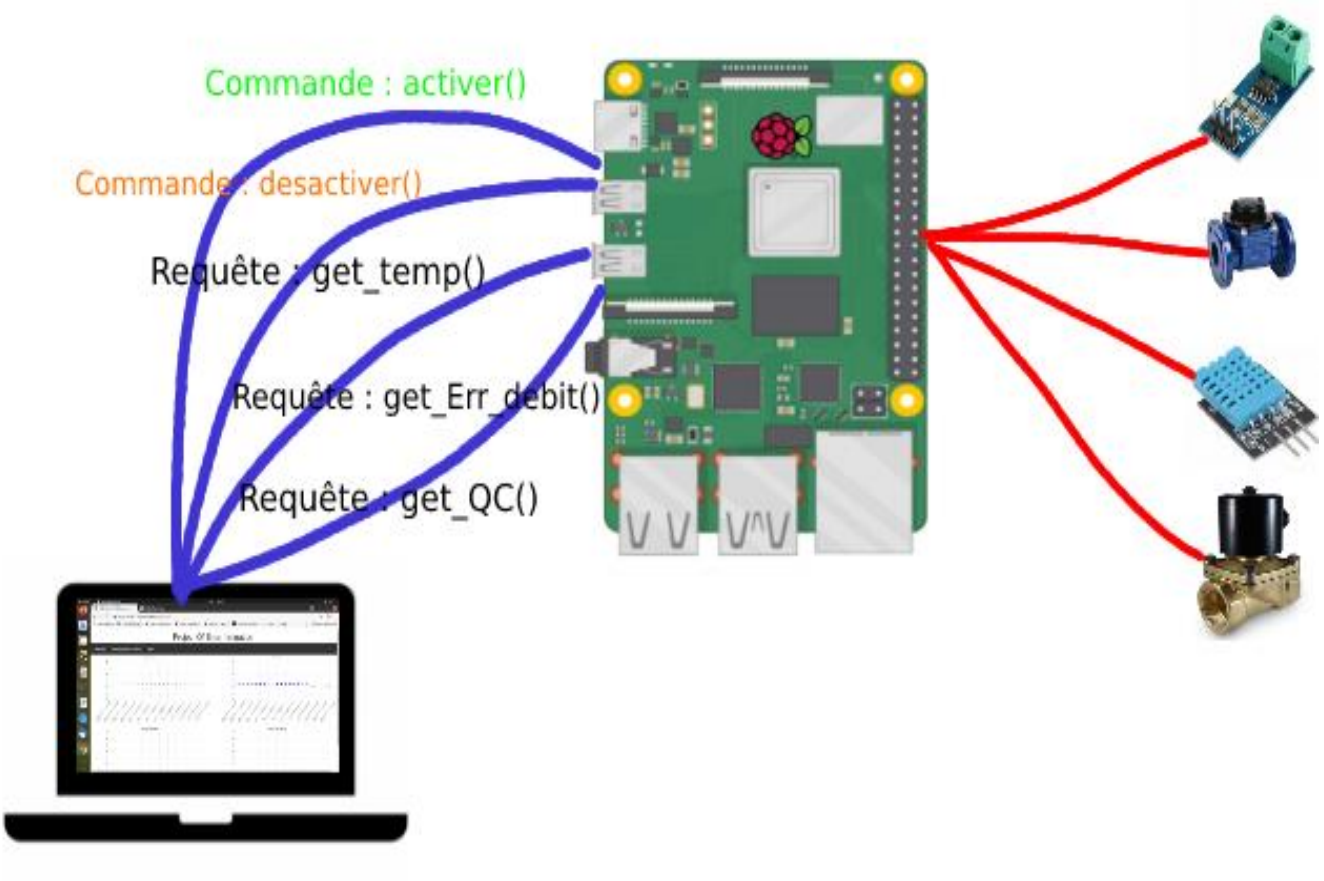


Figure 14 : Architecture générale du projet

Chapitre 3

Réalisation, implémentation et test du Projet

Réalisation du Projet

Pour la partie Serveur, l'interfaçage des composantes avec la carte Raspberry est de la manière suivante :

- Le capteur DHT11 est capable de mesurer des températures de 0 à +50°C avec une précision de +/- 2°C et des taux d'humidité relative de 20 à 80% avec une précision de +/- 5%. Une mesure peut être réalisée toutes les secondes.

  Le brochage du capteur est le suivant :

  - La broche n°1 est la broche d'alimentation (5 volts ou 3.3 volts).
  - La broche n°2 est la broche de communication.
  - La broche n°3 est la masse du capteur (GND).

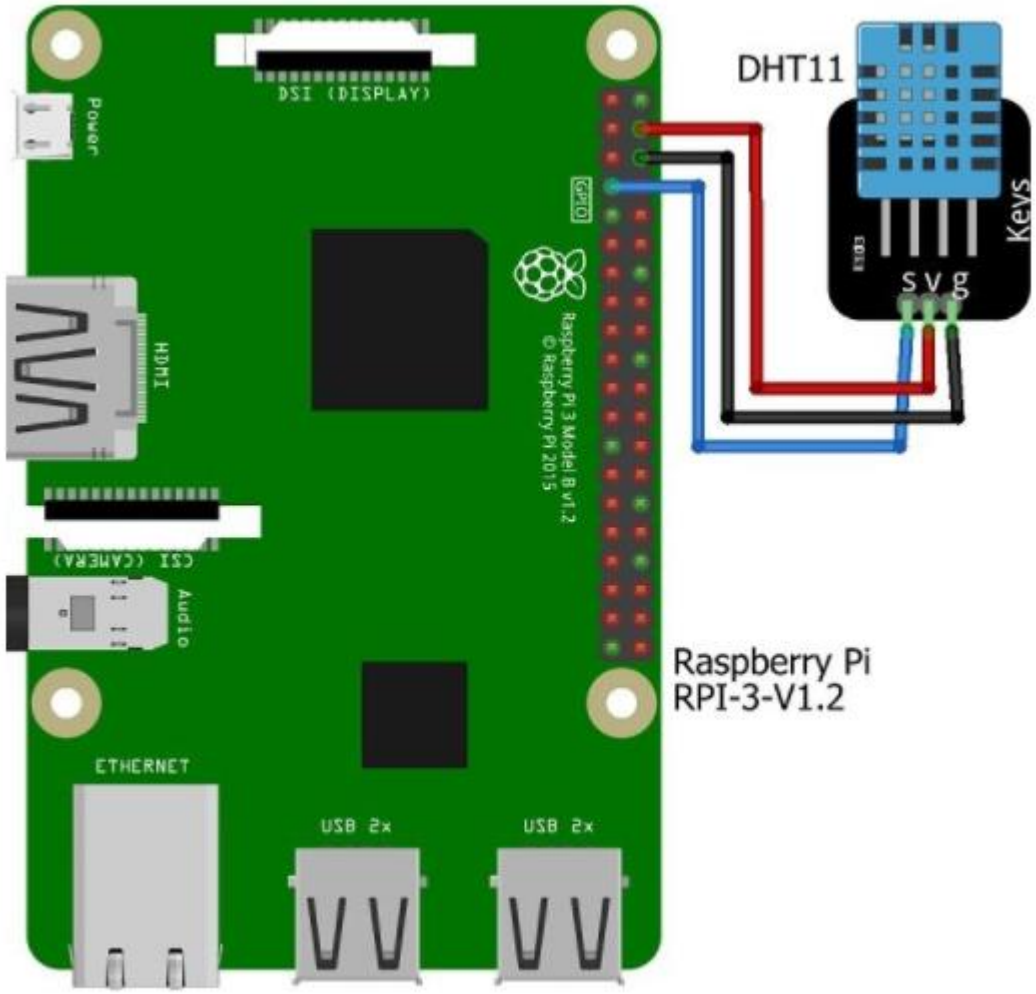


Figure 15 : Branchement du DHT11 avec la carte Raspberry Pi 3

- Le capteur de courant à effet Hall est basé sur ACS712. Le capteur donne une mesure précise du courant pour les deux signaux AC et DC. Ceci permet de conserver la module isolée même si l'on mesure le courant d'un appareil alimenté par du 220V. Ce module ACS712 est calibré pour mesurer des intensités allant jusqu'à 30A.

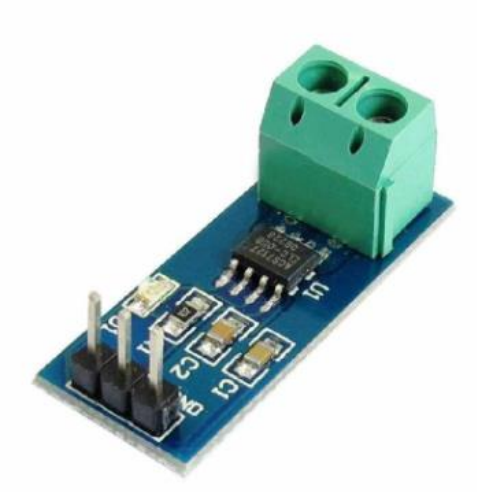


Figure 16 : Capteur de courant ACS712 30A

- Pour la mesure de l'erreur ou la variation du débit de l'eau, pour la détection de toute fuite d'eau, on peut utiliser 2 débitmètres. Le premier sera proche au point de l'entrée et le deuxième sera proche au point de la sortie. Le débitmètre ou le compteur d'eau permet de faire le comptage de la quantité d'eau passant dans une canalisation. Grâce à une transmission magnétique du signal, seule la turbine est en contact avec le flux d'eau permettant ainsi aux différents axes et engrenages de travailler complètement à sec en évitant ainsi une usure et une corrosion prématurée du compteur. En considérant le décalage horaire, on calcule la différence entre les deux débits d'entrée et de sortie pour obtenir l'erreur en résultat.

.

Figure 17 : débitmètr/compteur d'eau

- La vanne pour contrôler l'écoulement de l'eau.

L'interface Web se constitue de 2 pages principales :

- La première (**Dashboard**) contient principalement de 4 graphiques :
  - Le premier et le deuxième pour afficher les mesures de la température et l'humidité venant à partir du serveur après l'acquisition par le capteur DHT11. Cette opération se fait automatiquement chaque 10 secondes.
  - Le troisième pour afficher les mesures de la quantité de courant venant à partir du serveur après l'acquisition par le capteur de courant ACS712.
  - Le quatrième pour afficher les mesures de la variation ou l'erreur du débit venant à partir du serveur après l'acquisition par les deux débitmètres.

Génie des Systèmes Embarqués et Informatique Industrielle

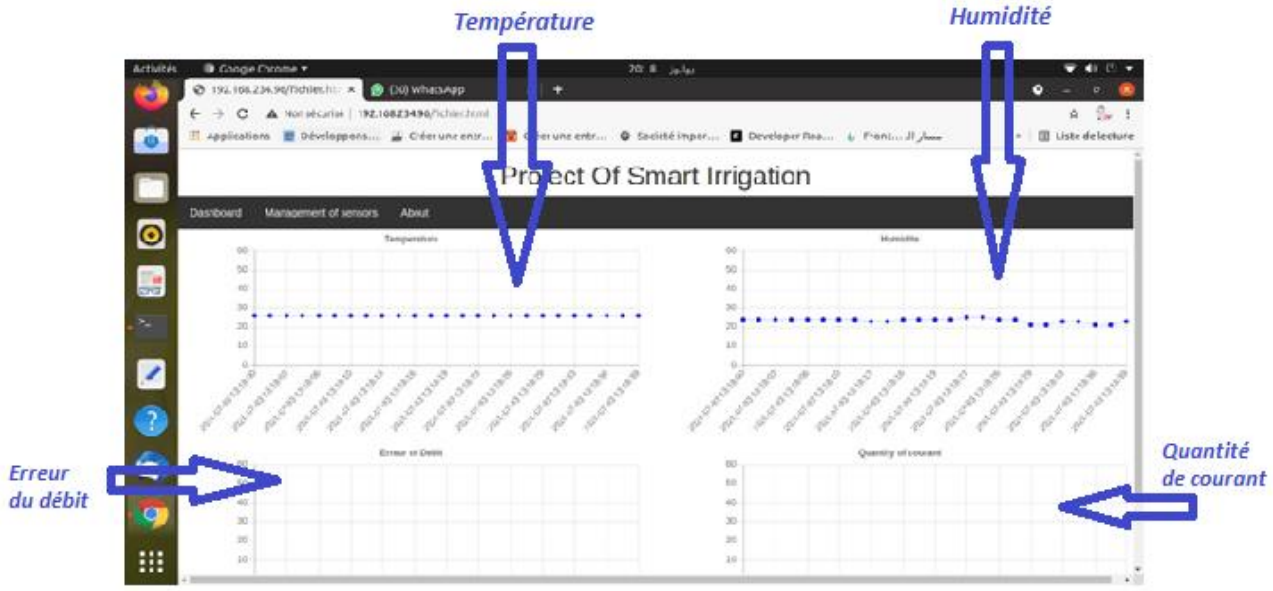


Figure 18 : les 4 graphes de la page Dashboard

- La deuxième page (**Management of sensors)** se constitue de deux buttons qui permettent au client d'activer ou désactiver la vanne.

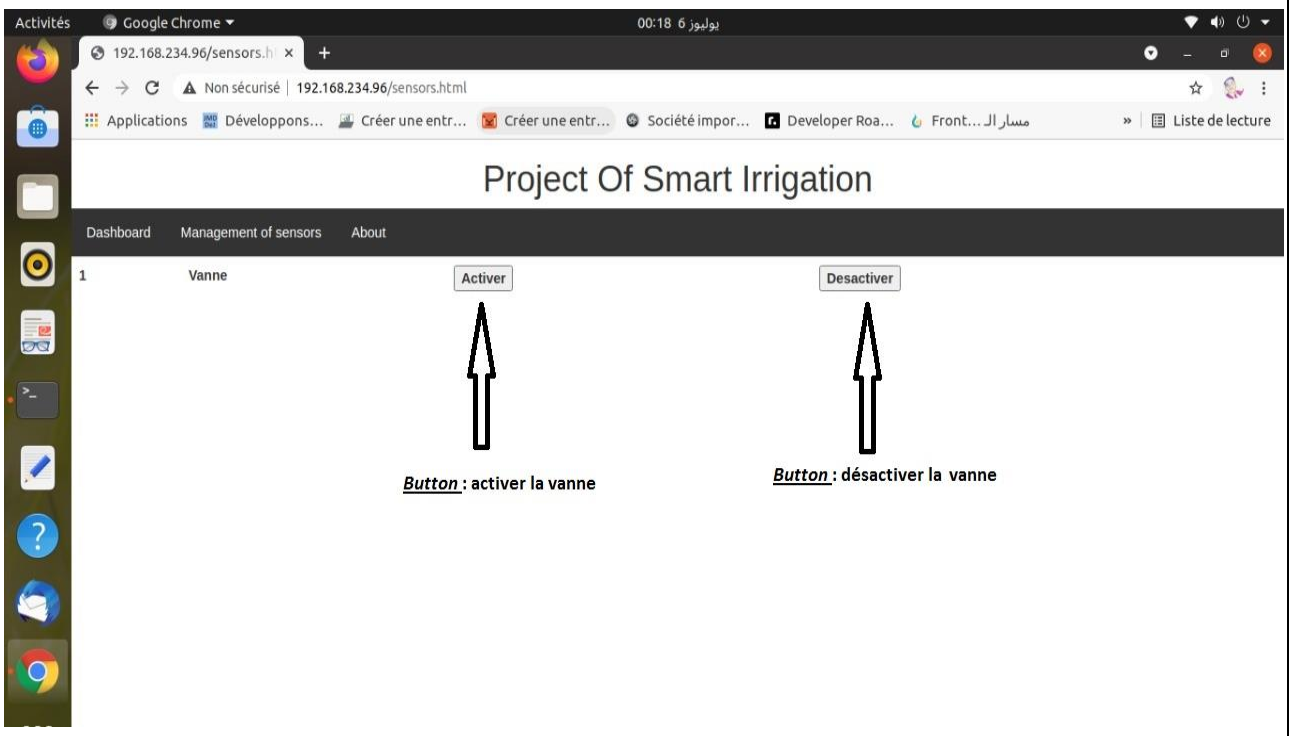


Figure 19 : les 2 boutons de la page Management of sensors

Codage et Implémentation

Nous allons attaquer dans cette partie, l'implémentation du capteur de température et d'humidité, le DHT11.

On commence par le script python qui capte les valeurs de température et d'humidité, les insère dans une base de données et les affiche sur le terminal.

Premièrement, on fait l'importation des bibliothèques dont on a besoin :

```
import sqlite3
import RPi.GPIO as GPIO
import os
import time
import glob
import time
import board
import adafruit_dht
```

Après, on initialise notre composante avec le pin de données numéro 17.

```
#Initial the dht device, with data pin connected to:
dhtDevice = adafruit_dht.DHT11(board.D17)
```

On fixe la période à 10 secondes, c'est-à-dire, après chaque 10 secondes on fait l'acquisition des données. Puis on donne le chemin de notre base de données à la variable « **dbname** ».

```
#Global variables
speriod=(10)
dbname='/home/pi/projet_PFA/temp_hum.db'
```

On définit la fonction « **get_temp()** » qui permet de récupérer les valeurs du capteur DHT11 et de les retourner.

```
#get temperature
def get_temp():
    try:
        # Print the values to the serial port
        global temp_c
        global humidity
        temp_c = dhtDevice.temperature
        humidity = dhtDevice.humidity
    except RuntimeError as error:      # Errors happen fairly often, DHT's are hard to read, just keep going
        print(error.args[0])
    return temp_c,humidity
```

Cette fonction « **log_temperature(temp_c,humidity)** » enregistre les valeurs retournées dans la base de données.

```
def log_temperature(temp_c,humidity):
    conn=sqlite3.connect(dbname)
    curs=conn.cursor()
    curs.execute("INSERT INTO info values(datetime('now'),(?),(?),(?),(?))",(temp_c,humidity,0.0,0.0))
    #commit the changes
    conn.commit()
    conn.close()
```

Cette fonction « **display_data()** » affiche notre table de base de données sur le terminal.

```
def display_data():
    conn=sqlite3.connect(dbname)
    curs=conn.cursor()
    for row in curs.execute("SELECT * FROM info"):
        print (str(row[0])+" "+str(row[1])+" "+str(row[2])+" "+str(row[3])+" "+str(row[4]))

    conn.close()
```

Enfin, la fonction principale « **Main** » qui permet d'appeler les fonctions définies en dessus dans une boucle infinie, chaque **speriod** (=10 secondes):

```
def main():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(4,GPIO.IN)
    while True:
        #get temp
        temperature,humidity=get_temp()
        if temperature !=None and humidity !=None:
            print ("tempera=" + (str(temperature)) + "      Humidity=" + (str(humidity)))
        else:
            temperature,humidity=get_temp()
            print ("temperature=" + (str(temperature)) + "         Humidity=" + (str(humidity)))
        log_temperature(temperature,humidity)
        display_data()
        time.sleep(speriod)
if __name__=="__main__":
    main()
```

Maintenant, on passe vers le code PHP (**f.php**) exécuté sur le serveur.

Ce code permet d'encoder en format JSON, le tableau retourné par l'exécution de la requête SQLite3 et l'envoyer vers le code JavaScript.

```php
<?php
header("Content-Type: application/json; charset=UTF-8");
$obj = json_decode($_GET["x"], false);
global $str;
$dbname='/home/pi/projet_PFA/temp_hum.db';
$mytable ="info";
$data=array();
if(!class_exists('SQLite3'))
  die("SQLite 3 NOT supported.");

$base=new SQLite3($dbname);
$res=$base->query('SELECT date_time,temp_c,humidity,q_courant,Debit from info');
while ($row = $res->fetchArray()) {
    //$jsonArray[] = $row;
  array_push($data,$row);
}
echo json_encode($data)
//echo json_encode($out);
?>
```

Puisque les codes sont fonctionnels dans le serveur, la partie Client est codée en deuxième lieu, pour l'interfaçage graphique sur une page Web (**fichier.html**) accessible depuis un navigateur internet, qui donne au client une visualisation de la variation ou l'erreur du débit et de l'énergie électrique consommée, plus la température et l'humidité, sur des graphiques en courbes pour fournir une distinction visuelle claire entre les jeux de données.

En premier lieu, on importe les bibliothèques bootstarp pour le design et Chart JS pour créer les graphes.

```html
<!DOCTYPE html>
<html>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.5.0/Chart.min.js"></script>
```

Cette partie représente le code CSS pour le style de la page.

```
<style>
 #collectionsTop {
width: 100%;
height:100%;
}
.topRight {
width:50%;
height:100%;
float:right;
}
.topLeft {
width:50%;
height:100%;
}
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}
li a:hover {
  background-color: #111;
}
</style>
```

Dans cette partie, on définit une barre de menu pour naviguer entre les différentes pages.

```
<body>

<center><h1>Project Of Smart Irrigation</h1></center>
<p id="demo">
</p>

<ul>
  <li><a class="active" href="fichier.html">Dashboard</a></li>
  <li><a href="sensors.html">Management of sensors</a></li>
  <li><a href="#about">About</a></li>
</ul>
```

Ici, on crée un tableau de quatre cases dans lequel on affiche les graphes.

```
<div class="Container">
  <div class="row">
    <div class="col-md-6">
      <canvas id="Temperature" ></canvas>
    </div>
    <div class="col-md-6">
      <canvas class="topLeft" id="Humidite"></canvas>
    </div>
  </div>
  <div class="row">
    <div class="col-md-6">
      <canvas id="Erreur"></canvas>
    </div>
    <div class="col-md-6">
      <canvas id="QC"></canvas>
    </div>
  </div>
</div>
```

On définit une fonction JavaScript qui récupère le fichier JSON, le transforme en objet, et le parcourt pour remplir les tableaux.

```
function real_time(){
var time=new Array();
var temperature=new Array();
var humidity=new Array();
var Q_C=new Array();
var Err_debit=new Array();
const xmlhttp = new XMLHttpRequest();
xmlhttp.onload =function ma() {
myObj = JSON.parse(this.responseText);
var l=myObj.length -25;
for (let x in myObj) {
  time[x]=myObj[x][0];
  temperature[x]=myObj[x].temp_c;
  humidity[x]=myObj[x].humidity;
  Q_C[x]=myObj[x].q_courant;
  Err_debit[x]=myObj[x].Debit;
 }
var xValues = time.slice(1);
```

Création des 4 graphes par le constructeur Chart :

Pour la température :

```
new Chart("Temperature", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      fill: false,
      lineTension: 0,
      backgroundColor: "rgba(0,0,255,1.0)",
      borderColor: "rgba(0,0,255,0.1)",
      data: temperature.slice(1)
    }]
  },
  options: {
  title: {
      display: true,
      text: "Temperature"
    },
    legend: {display: false},
    scales: {
      yAxes: [{ticks: {min: 0, max:60}}],
      }
    }
  }

);
```

Pour l'humidité :

```
new Chart("Humidite", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      fill: false,
      lineTension: 0,
      backgroundColor: "rgba(0,0,255,1.0)",
      borderColor: "rgba(0,0,255,0.1)",
      data: humidity.slice(1)
    }]
  },
  options: {
  title: {
      display: true,
      text: "Humidite"
    },
    legend: {display: false},
    scales: {
      yAxes: [{ticks: {min: 0, max:60}}],
    }

  }
});
```

Pour La quantité de courant (QC) :

```
new Chart("QC", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      fill: false,
      lineTension: 0,
      backgroundColor: "rgba(0,0,255,1.0)",
      borderColor: "rgba(0,0,255,0.1)",
      data: Q_C.slice(1)
    }]
  },
  options: {
  title: {
      display: true,
      text: "Quantity of courant"
    },
    legend: {display: false},
    scales: {
      yAxes: [{ticks: {min: 0, max:60}}],
    }

  }
});
```

Pour la variation du débit de l'eau (l'Erreur) :

```
new Chart("Erreur", {
  type: "line",
  data: {
    labels: xValues,
    datasets: [{
      fill: false,
      lineTension: 0,
      backgroundColor: "rgba(0,0,255,1.0)",
      borderColor: "rgba(0,0,255,0.1)",
      data: Err_debit.slice(1)
    }]
  },
  options: {
  title: {
      display: true,
      text: "Erreur of Debit"
    },
    legend: {display: false},
    scales: {
      yAxes: [{ticks: {min: 0, max:60}}],
    }
}
  }
);
```

On exécute le fichier PHP et on récupère le fichier JSON.

```
xmlhttp.open("GET", "f.php?x=");
xmlhttp.send();
```

On Appelle la fonction chaque 10 s.

```
setInterval(real_time,10000);
</script>
</body>
</html>
```

La partie Client possède une deuxième page web (**sensors.html**) pour contrôler la vanne, accessible depuis un navigateur internet, qui permet le client d'activer ou désactiver la pompe.

Comme la première page, on importe les bibliothèques bootstarp pour le design et Chart .js. Plus une partie qui représente le code CSS pour le style de la page. Et une autre, où on définit une barre de menu pour naviguer entre les différentes pages.

Ici, on crée les deux boutons pour activer ou désactiver la vanne.

```
<div class="table-responsive">
 <table class="table">
   <tbody>
     <tr>
       <th>1</th>
       <th>Vanne</th>
       <th><button>Activer</buttton></th>
       <th><button>Desactiver</button></th>
 </tr>
 </tbody>
 </table>
 </div>
```

Test du projet

Pour le test du projet, et vu qu'on a juste le DHT11, on a inséré alors des valeurs nulles pour la quantité de courant et l'erreur du débit.

Or le DHT11 a bien fonctionné comme on a vu sur la figure 10 et la figure 18.

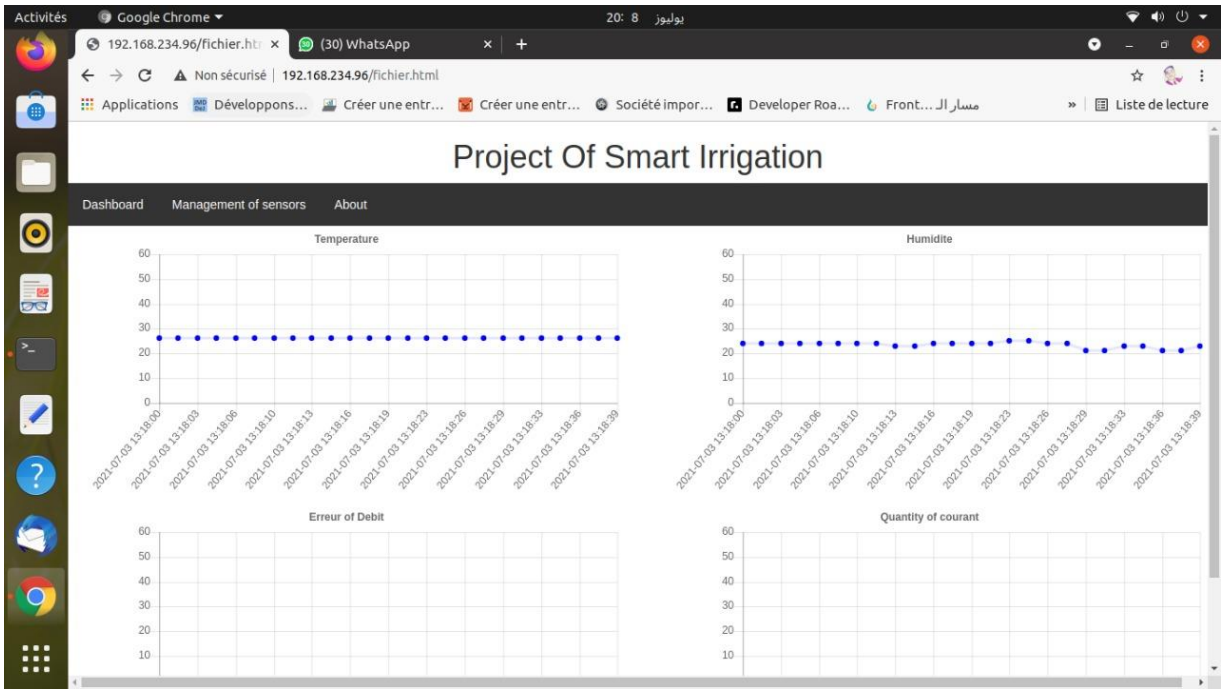


Figure 20 : Page Web "fichier.html"

Conclusion

Ce projet qui entre dans le cadre du PFA, nous a beaucoup apporté à notre savoir technique, car sa réalisation comportent l'interaction entre plusieurs éléments informatiques et embarqués, notamment l'étude approfondie de la carte Raspberry Pi, sa circuiterie, ses éléments d'interfaçage GPIO, en plus du système d'exploitation qui est basé sur Linux et qui offre plusieurs fichier de virtualisation de ressources matérielles qui rendent la réalisation des drivers beaucoup plus fiable.

Ainsi, l'architecture client-serveur apporte plusieurs complexité à résoudre avant d'attaquer tout projet qui est basé sur ce modèle, principalement la mise en marche du serveur web, ses fonctionnalités, son espace du travail vis-à-vis du Linux, l'ensemble des interactions entre l'utilisateur web et les permissions attribuées par le système pour contrôler l'accès à toute page et ressource. De plus, faire un chemin de travail souple et fiable entre la partie logicielle, notamment les scripts JS, les script du back-end PHP et l'exécution ou la terminaison des commandes ou processus sous Linux, nécessite une connaissance approfondie des 2 niveaux, hardware et software, une chose que nous avons réussie a établir à partir de ce projet.

Merci.

Références

Configuration du Lighttpd :

https://elinux.org/RPi_webserver

Adafruit Repo:

https://github.com/adafruit/Adafruit_Python_DHT

Chartjs biblio :

https://www.chartjs.org/

Bootstrap Framework :

https://getbootstrap.com/

AJAX :

https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started

Tutoriels pour la Raspberry Pi :

https://www.raspberrypi.org/

https://raspberrypi-tutorials.fr/