[Projet tutoré] Communication internet avec arduino uno

IUT Génie Electrique et Informatique Industrielle FI-ESE S3

Simon REMY
Asad MUHAMMAD
Thomas MARCHAND
Simon MARTIN

Sommaire :

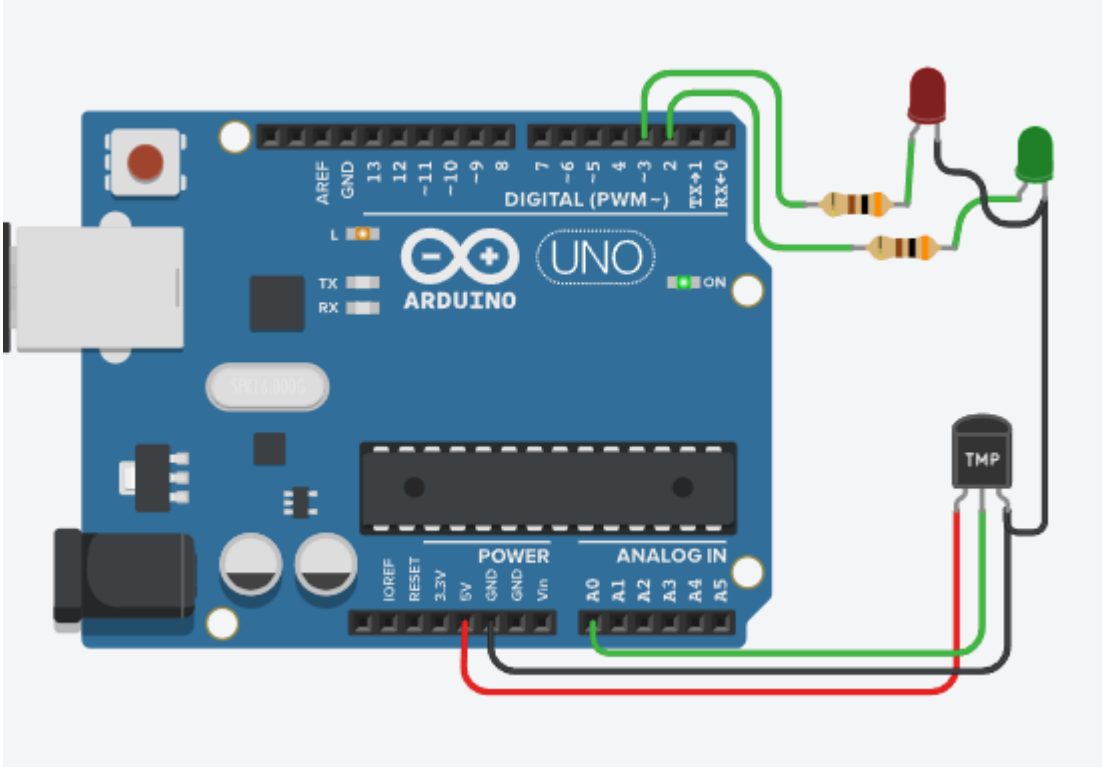
1. Introduction

Notre projet vise à créer une communication intelligente entre une carte Arduino (un boîtier placé chez soi et connecté à un fournisseur d'accès Internet) et un PC ou un smartphone n'importe où dans le monde où on a accès à Internet. Le matériel nécessaire comprend une carte Arduino Uno, une plaque lab, une carte Ethernet Shield 2 Rev3 et un capteur de température analogique TMP 36.

Les étapes clés de ce projet incluent le téléchargement et l'installation de l'environnement Arduino sur nos PC, la familiarisation avec cet environnement grâce à des tutoriels disponibles en ligne ou avec l'aide du Starter Kit, la mesure de la température avec le capteur TMP36, l'utilisation du Shield Ethernet en configurant notre box via les connaissances en réseaux, la lecture à distance de la température, et la simulation d'une commande de chauffage par l'allumage de LED. Enfin, une page web traditionnelle sera créée pour présenter la température en direct. Ce projet combine ainsi le monde d’Arduino, la connectivité Internet, et l'automatisation pour offrir une solution pratique et interactive.

2. Le capteur de température vers la carte Arduino

Pour réaliser cette partie, un capteur TMP36 nous a été fourni. Ce capteur possède 3 broches, l'alimentation, la masse et le fil de données(sous forme analogique).



Ci-dessous, le code permettant de lire la température du capteur :

```
/*------TMP36----------------------
**       _________
**       |       |
**       | TMP36 |
**       ---------
**         | | |
**       GND,A0,+5v
**
**     A0 = Pin Analog de Arduino
**
**
*/

#define brocheCapteur A0


void setup() {
```

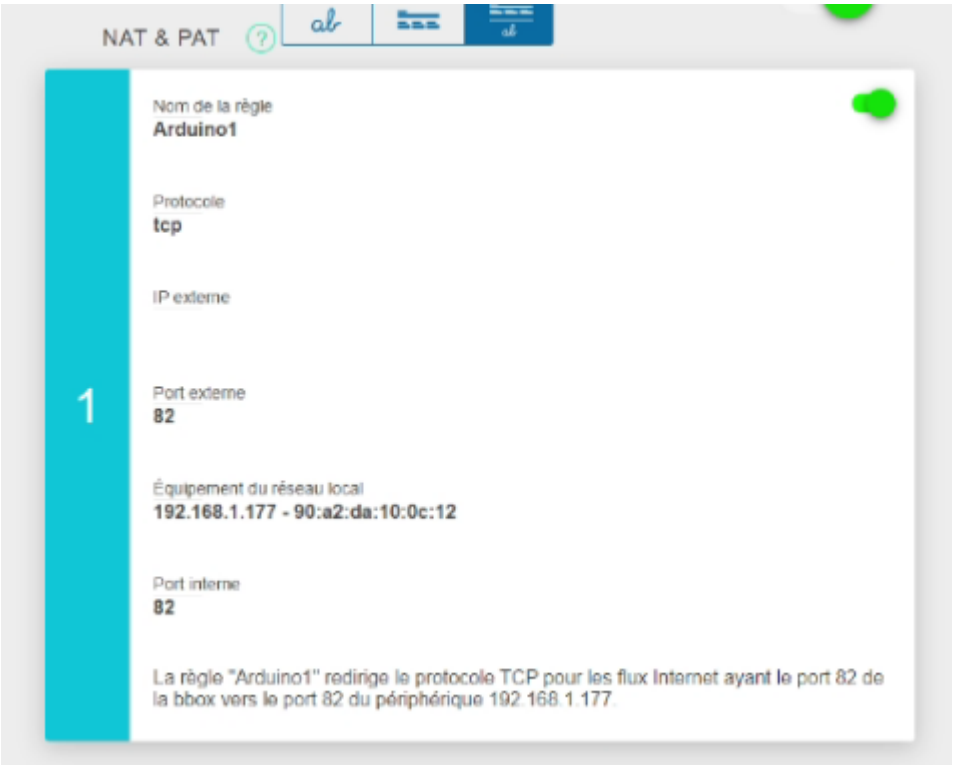
```
  Serial.begin(9600);
  // communication série entre l'ordinateur et la carte à 9600 Bauds
  pinMode(brocheCapteur, INPUT);
  //on a choisi le pin A0 pour recevoir la tension du capteur. On le
configure donc en entrée.
  Serial.println("Initialisation terminee");
}

float lire_temp() // Programme permettant de lire la température
transmise par le capteur
{
  float valeur = analogRead(brocheCapteur); // on récupère la valeur
transmise par le capteur
  Serial.println(valeur);
  // Cette valeur ne correspond pas à une température, il s'agit d'un
code binaire compris entre 0 et 1024 (2^8). Il faut donc convertir la
valeur en une tension puis en une température.

  // Conversion en une tension, il s'agit de la même procédure que pour
un CAN
  float tension = valeur * 5 / 1024.0;
  Serial.println(tension);

  //Conversion de la tension en une température. D'après la datasheet,
la courbe de la tension en fonction de la température possède un
coefficient directeur de 10 mV/°C et un offset de 0.5 V
  float temperature = (tension - 0.5) * 100.0;
  return temperature;
}

void aff_temp(float temperature) //Fonction permettant d'afficher la
température sur le terminale.
{
  Serial.print(temperature); Serial.println(" °C");
  Serial.println();
  delay(1000);
}

void loop()
{
  aff_temp(lire_temp());
}
```

3. Connexion de la carte arduino à internet

```
#include <Ethernet.h>
#include <SPI.h>       // appelle de 2 librairies

#define brocheCapteur A0

byte MAC[] = { 0x90, 0xA2, 0xDA, 0x10, 0x0B, 0xF6 }; //
IPAddress ip(192, 168, 1, 177);  // adresse IP local, il faut la
chercher sur google via
EthernetServer server(82); // correspond au port 82 d'Ethernet
```

La base avant de connecter notre shield ethernet sur un réseau, c'est de lui donner un identifiant. L'adresse MAC s'identifie de manière unique la carte réseau de l'Arduino, un peu comme un numéro d'identification personnelle. L'adresse IP, quant à elle, nous sert à localiser l'Arduino sur notre réseau. En utilisant ces deux types d'adresses, on permet à l'Arduino de communiquer et d'interagir avec n'importe quels appareils dans le même réseau.

Configuration de la box internet pour qu'on ait accès à l'arduino (avec Bouygues) :



Boucle Setup / exécutée une seule fois :

```
void setup() {
  Serial.begin(9600);
  analogReference(EXTERNAL);
  Ethernet.begin(MAC, ip);  // Fonction permettant de se connecter à
internet
  Serial.print("Adresse du server : ");
  Serial.println(Ethernet.localIP());
  Serial.println("Pret");
}
```

Cette partie de code est une configuration initiale pour notre projet Arduino.
Elle comprend les étapes suivantes :

- Initialisation de la communication série à une vitesse de 9600 bauds.
- Définition de la référence analogique sur EXTERNAL, ce qui signifie que le module Arduino utilisera une référence externe pour les mesures analogiques. Ici on utilise la référence externe pour L'ADC de l'arduino car a cause de notre shield ethernet il ya une chute de tension sur pin 5V de 5V à 4.7V cette tension est aussi utilisé comme référence analogique pour ADC, c'est à dire qu'au lieu d'avoir 5V pour 1023 on aura 4.7V pour 1023 donc la valeur de quantum change ce qui ajoute l'erreur dans notre mesures donc on utilise une référence externe qui est stable pour avoir des bonnes valeurs.
- Initialisation de la connexion Ethernet en utilisant une adresse MAC et une adresse IP spécifiées.
- Affichage de l'adresse IP locale du module Ethernet.
- Affichage du message "Prêt" pour indiquer que la configuration est terminée et que le module est prêt à fonctionner.

Boucle Loop / exécutée de manière infiniment :

```
void loop() {
  float temp = lire_temp(); // variable flottant qui prend en compte la
valeur de la température.
  Serial.println(temp, 2);

  EthernetClient client = server.available();
  if (client) {
    Serial.println("Nouveau client");
    boolean currentLineIsBlank = true;
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        if (c == '\n' && currentLineIsBlank) {
          client.println("HTTP/1.1 200 OK");
          client.println("Content-Type: text/html");
          client.println("Connection: close");
          client.println("Refresh: 3");
          client.println();
          client.println("<!DOCTYPE HTML>");
          client.println("<html>");
          client.println("<meta charset = 'UTF-8'>");
          client.println("<meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
          client.println("<title> Temperature </title> ");
          client.print("<center> <h1> Temperature : ");
          client.print(lire_tem p());
          client.println("°C ");
          client.println("</html></h1></center>");
          break;
        }
        if (c == '\n') {
          currentLineIsBlank = true;
        } else if (c != '\r') {
          currentLineIsBlank = false;
        }
      }
    }
    delay(100);
    client.stop();
    Serial.println("client disconnected");
  }
  delay(10);
}
```

Cette partie de code correspond à la boucle principale (loop).
Elle comprend les étapes suivantes :

Lit la température à l'aide de la fonction "lire_temp()" et l'affiche sur le port série avec deux décimales.
Vérifie s'il y a un client Ethernet en attente de connexion.
Si un client est disponible, il traite la demande HTTP en envoyant une réponse avec les en-têtes appropriés, le contenu HTML incluant la température actuelle, puis ferme la connexion.
Le code gère la communication avec le client en surveillant les données entrantes et en s'assurant que la ligne actuelle est vide pour déterminer la fin de la demande HTTP.
Après avoir traité la demande du client, il arrête la connexion, affiche un message indiquant que le client s'est déconnecté, puis attend 10 millisecondes avant de recommencer la boucle.
En résumé, cette partie de code permet à un Arduino de répondre à une requête HTTP en fournissant la température actuelle via une page HTML lorsque l'accès est fait via un navigateur web.

Voici la page web obtenue :

Il s'agit d'une capture d'écran du téléphone d'Asad lorsqu'il se connecte à l'adresse IP : 176.138.71.119 et en se connectant sur le port 82 du routeur.

De la même manière, cette capture est issu du navigateur de Simon REMY en se connectant à la même adresse.

4. Développement du site internet

4.1. Index.html et Index.css

Nous avons segmenté notre travail sur le site en deux parties distinctes : la première se concentre sur la conception de la page d'accueil, tandis que la seconde est dédiée à l'intégration directe du compte rendu sur le site.

```
<!DOCTYPE html>
<html lang='fr'>
    <head>
        <meta charset='UTF-8'>
        <meta name='viewport' content='width=device-width,
initial-scale=1.0'>
        <link rel='stylesheet' href='Index.css' type='text/css'/>
        <link rel="preconnect" href="https://fonts.googleapis.com">
        <link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin>;
        <link
href='https://fonts.googleapis.com/css2?family=Poppins&display=swap'
rel='stylesheet'>
        <title>Projet tutoré</title>
    </head>
    <body>
        <header class='header'>
            <h2 class='Logo'>Tutored Project</h2>
            <a href='https://www.cyu.fr'><img class='cyu’
src='images/CYU.png' alt='CY.TECH'></a>
            <nav class='navigation'>
                <a href='/About.html'>About</a>
            </nav>
        </header>

        <div class='welcome-container'>
            <h1 class='welcome-text'>WELCOME TO OUR WEBSITE</h1>
        </div>

        <div class='temperature'>
            <h2>Il fait     °C</h2>
        </div>
    </body>
</html>
```

Ce code HTML crée une page web avec une structure basique. Il démarre par des métadonnées et des liens externes pour le style. Ensuite, il présente un en-tête avec un logo et une barre de navigation. Une section de bienvenue dynamique accueille les visiteurs, suivie d'une zone réservée à l'affichage de la température (à implémenter).

Pour améliorer l'apparence de notre site web, il est crucial d'avoir un code CSS associé au code HTML. Le code CSS joue un rôle essentiel en définissant les styles, tels que les couleurs, les polices, les espacements et les mises en page. Sans ce code, la page apparaîtrait sans personnalisation, adoptant des aspects par défaut basiques.

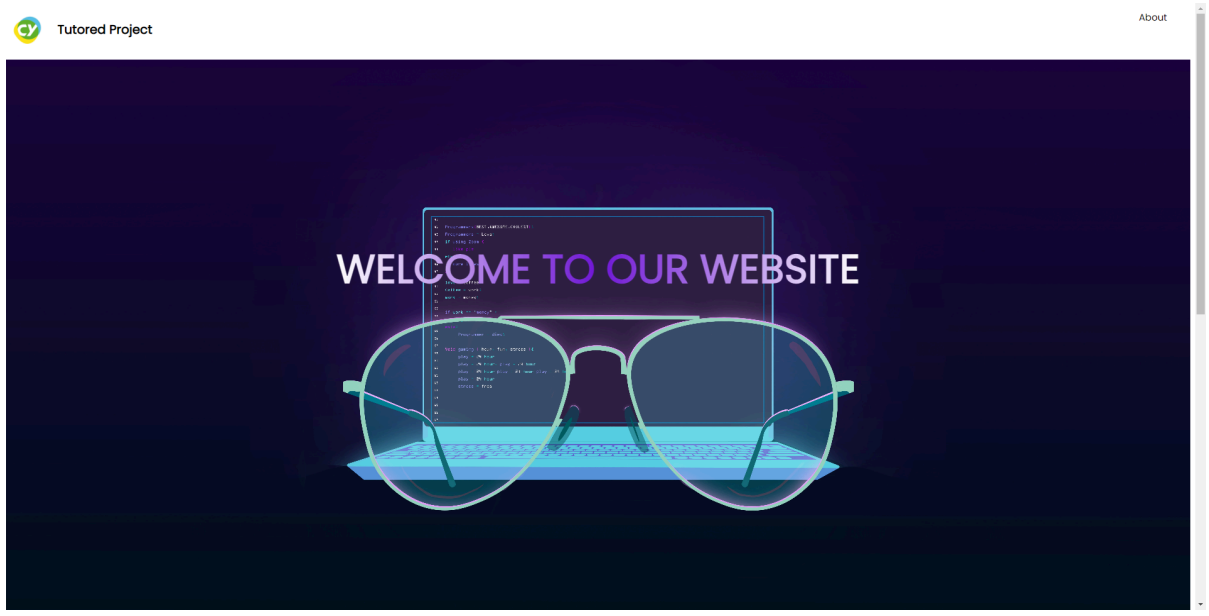
Voici le code en CSS :

```css
body{
    font-family: 'Poppins',sans-serif;
    margin: 0;
}
header{
    position: fixed;
    top: 0;
    left: 0;
    width: 95%;
    padding: 10px 90px;
    display: flex;
    justify-content: space-between;
    z-index: 99;
}
.header{
    background-color: white;
}
.Logo{
    font-size: 20px;
    color: black;
    user-select: none;
}
.cyu {
    display: block;
    margin: 0 auto;
    width: 45px;
    height: 45px;
    top: 20px;
    left: 20px;
    position: absolute
}
```

```css
.navigation a{
    position: relative;
    font-size: 15px;
    color: black;
    text-decoration: none;
    font-weight: 500;
    margin: 40px;
}
.navigation a::after {
    content: '';
    position: absolute;
    width: 100%;
    height: 3px;
    background-color: black;
    border-radius: 5px;
    left: 0;
    bottom: -6px;
    transition: .5s;
    transform: scaleX(0);
    transform-origin: right;
}
.navigation a:hover::after{
    transform: scaleX(1);
    transform-origin: left;
}
.navigation .btnlogin-popup{
    position: relative;
    font-size: 1.1em;
    width: 130px;
    height: 50px;
    background: black;
    outline: none;
    cursor: pointer;
    border-radius: 6px;
    color: white;
    font-weight: 500;
    margin-left: 40px;
    transition: .5s;
}
.navigation .btnlogin-popup:hover{
    background-color: white;
    color: black;
}
```

```css
.welcome-container{
    height: 900px;
    margin-top: 3%;

    display: flex;
    justify-content: center;
    align-items: top;

    background-image: url(/images/peakpx.jpg);
    background-size: cover;
    background-position: center;
}
.welcome-container .welcome-text{
    margin-top: 15%;
    font-size: 4em;
    color: transparent;
    background-image: linear-gradient(to right, #fff, rgb(112, 23,
214), #fff);
    -webkit-background-clip: text;
    background-clip: text;
    animation: animate 8s linear infinite alternate-reverse;
    position: absolute;
}
.temperature{
    height: 600px;
    font-size: 4em;
    margin-top: 15%;
    text-align: center;
}
```

Ce code CSS stylise notre page d'accueil HTML. Il définit les styles de base pour le corps de la page, créer un en-tête fixe avec un logo, des liens de navigation et un bouton de connexion. Il organise également une section de bienvenue, d'affichage de la température, et un pied de page. Les sélecteurs ciblent des éléments spécifiques pour leur appliquer des styles tels que la police, la couleur, la position et les dimensions.
Ci-dessous, la page web du site :



4.2. About.html et About.css

Dans cette partie du programme, nous allons y inscrire le compte rendu du projet afin de pouvoir y accéder lorsque la carte est en fonctionnement. Cependant, la page n'est pas terminée puisque le projet n'est pas complètement fini. C'est pourquoi, seul le ode css est présenté ci-dessous :

```
body{
  font-family: 'Poppins',sans-serif;
  background-color: white;
  background-image: url(/images/Fond_About.jpg);
  background-size: cover;
```

```
}
header{
  position: fixed;
  top: 0;
  left: 0;
  width: 95%;
  padding: 10px 90px;
  display: flex;
  justify-content: space-between;
  z-index: 99;
}
.Tutored_Project{
  font-size: 20px;
  color: white;
  user-select: none;
}
.logo_cyu{
  display: block;
  margin: 0 auto;
  width: 45px;
  height: 45px;
  top: 20px;
  left: 20px;
  position: absolute
}
.navigation a{
  position: relative;
  font-size: 15px;
  color: white;
  text-decoration: none;
  font-weight: 500;
  margin: 40px;
}
.navigation a::after {
  content: '';
  position: absolute;
  width: 100%;
  height: 3px;
  background-color: white;
  border-radius: 5px;
  left: 0;
  bottom: -6px;
```

```
  transition: .5s;
  transform: scaleX(0);
  transform-origin: right;
}
.navigation a:hover::after{
  transform: scaleX(1);
  transform-origin: left;
}
.navigation .btnlogin-popup{
  position: relative;
  font-size: 1.1em;
  width: 130px;
  height: 50px;
  background: transparent;
  outline: none;
  cursor: pointer;
  border-radius: 6px;
  color: white;
  font-weight: 500;
  margin-left: 40px;
  transition: .5s;
}
.navigation .btnlogin-popup:hover{
  background-color: white;
  color: #162938;
}
.Titre{
  color: white;
  width: 80%;
  margin: 20px auto;
  padding: 50px;
  text-align: center;
}
.content{
  text-align: left;
  width: 80%;
  margin: 20px auto;
  text-decoration: none;
}
footer {
  margin-top: 30px;
  border-top: 1px solid #ddd;
```

```
  padding-top: 10px;
  font-size: 0.8em;
  text-align: right;
}
```

Ce code reprend beaucoup les fonctions d’Index.css à l'exception du footer à le rendu suivant sur la page web :

5. Implémentation du site web sur la carte arduino

Pour implémenter le site web sur la carte arduino, nous avons utilisés une carte SD afin de stocker les fichiers html et css nécessaire au site web, cette méthode nous permet d'avoir un site web plus développé en comparaison d'implémenter le site web directement dans le code. ci-dessous, le programme correspondant (ici, le site web est affiché en local) :

```
#include <Ethernet.h>
#include <SPI.h>
#include <SD.h>
```

On utilise la bibliothèque SD.h afin de pouvoir communiquer avec elle.

```
String reception = "";
char c;
```

Déclaration de variable (utilisé pour lire les requêtes HTTP du site web).

```
void setup() {
  Ethernet.begin(MAC, ip);                                          //
Fonctions permettant de se connecter à internet
  Serial.begin(9600);
  server.begin();
  Serial.print("Adresse du server : ");
  Serial.println(Ethernet.localIP());

  if(!SD.begin(4)){                                                 //
Fonction permettant d'initialiser la carte SD (port 4 de la carte)
ainsi que la bibliothèque SD.h
    Serial.println("Erreur, la carte SD n'a pas été initialisee");
    return;                                                         //
Si la carte n'est pas initialisé, alors on arrête le programme
  }
  Serial.println("Carte SD initialisee");

  if(!SD.exists("Index.txt")){                                      //
On teste si la carte arduino arrive à accéder au différent fichier du
site web
    erreur("Index.txt");
    return;
  }
  if(!SD.exists("Index.css")){
    erreur("Index.css");
```

```
    return;
  }
  if(!SD.exists("About.txt")){
    erreur("About.txt");
    return;
  }
  if(!SD.exists("About.css")){
    erreur("About.css");
    return;
  }
  delay(1000);                                                    //
On attend 1s pour que le shield puisse s'initialiser
  Serial.println("Preparation termine");
}
```

Dans la fonction setup, on initialise la communication internet, puis on teste si les différents composants essentiels à l'affichage du site web sont présents et accessibles. Pour cela, on appelle ces deux fonctions :

```
int test_fichier(String nom){ // Fonction permettant de tester
l'existence des fichiers avant de les ouvrir
  return !SD.exists(nom) ? 0 : 1;
}

void erreur(String nom){ // Fonction permettant d'afficher un message
d'erreur si un fichier n'a pas été trouvé
  Serial.print("Erreur, le fichier ");
  Serial.print(nom);
  Serial.println(" est introuvable");
}
```

On teste l'existence du fichier avec 'test_fichier'. Si le fichier est absent ou inaccessible, on affiche un message d'erreur sur la communication série avec 'erreur' et on arrête le programme.

```
void loop() {
  EthernetClient client = server.available();
  if (client){
    reception = "";
    if (client.connected()){                                      // Si
une personne est connectée sur le site web
      Serial.println(F("Client connecte"));
      if (client.available()) c = client.read();                  // On
lit la requête HTTP du site web
```

```
      while (client.available() && c != '\n'){                    //
Tant qu'il y a une personne et
        reception = reception + c;                                 //
que la 1ère ligne de la requête n'est pas terminé
        c = client.read();                                         // On
lit la requête
      }
      Serial.println(reception);

      if (reception.startsWith("GET / HTTP/1.1")){                 //
Partie permettant de répondre au requête du site web
        afficher(client, "Index.txt", "html");
      }
      else if (reception.startsWith("GET /Index.css HTTP/1.1")){
        afficher(client, "Index.css", "css");
      }
      else if (reception.startsWith("GET /About.html HTTP/1.1")){
        afficher(client, "About.txt", "html");
      }
      else if (reception.startsWith("GET /About.css HTTP/1.1")){
        afficher(client, "About.css", "css");
      }
      else if (reception.startsWith("GET /images/CYU.png HTTP/1.1")){
        afficher(client, "/images/Cyu.png", "png");
      }
      else if (reception.startsWith("GET /images/peakpx.jpg
HTTP/1.1")){
        afficher(client, "/images/peakpx.jpg", "jpg");
      }
      else if (reception.startsWith("GET /images/Fond_About.jpg
HTTP/1.1")){
        afficher(client, "/images/Fond_About.jpg", "jpg");
      }

      delay(1);
      client.stop();
    }
  }
}
```

Dans la fonction loop, on regarde si une personne est connecté sur le site, si elle connecté, alors on récupère la première ligne de la requête HTTP envoyé (Seule la première ligne est

utile pour pouvoir répondre correctement à la requête). En fonction de la requête HTTP, on envoie le fichier correspondant, en utilisant la fonction ci-dessous :

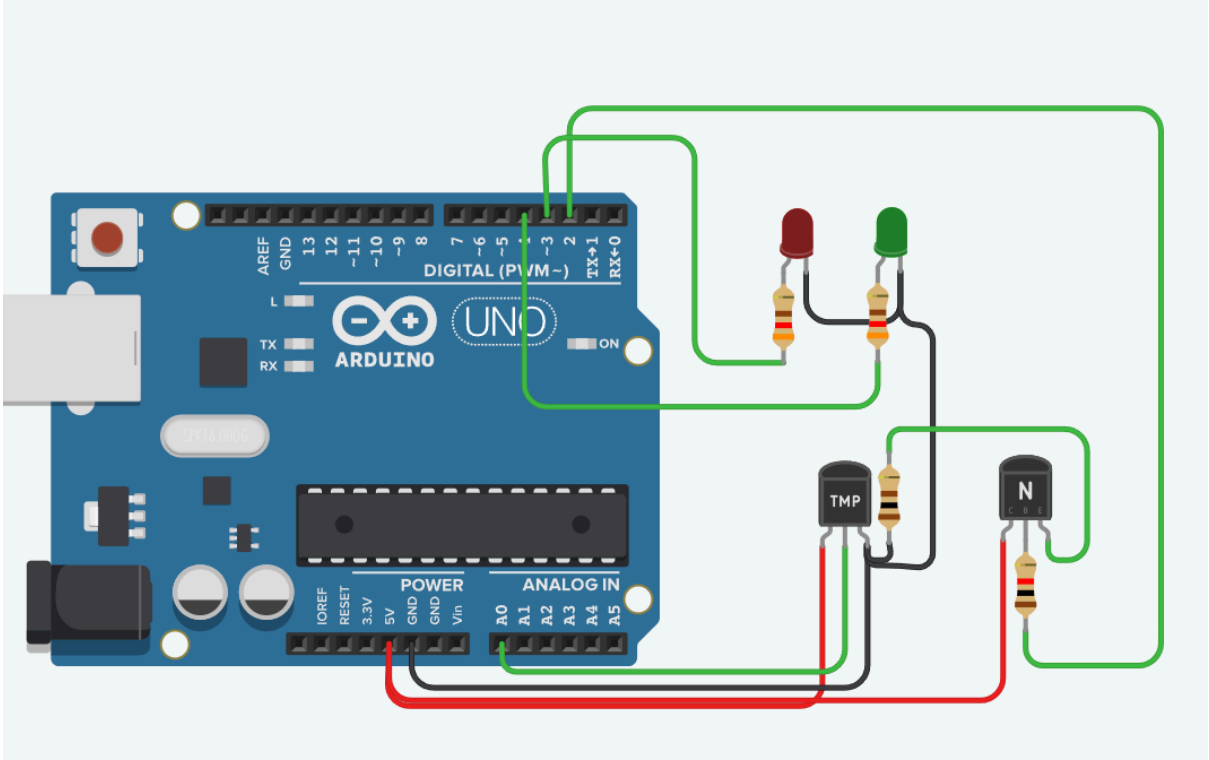
```
void afficher(EthernetClient client, String nom_fichier, String type){
// Fonction permettant de répondre au requête HTTP du navigateur

// En fonction des arguments, on envoie une réponse personnalisé
  client.println(F("HTTP/1.1 200 OK"));
// On envoi l'accusé de bonne réception au client
  client.print(F("Content-Type: text/"));
// On indique le type du contenu de la réponse
  client.println(type);
  client.println(F("Refresh: 10"));
// La page web se rafraîchit toutes les 10 s
  client.println(F("Connection: close"));
// La connexion sera fermée en fin de réception
  client.println();
// Ligne blanche imposée par le protocole HTTP
  delay(10);
// La communication n'est pas instantanée, il faut attendre

  File fichier = SD.open(nom_fichier, FILE_READ);
// On ouvre le fichier sur la carte SD voulue, en mode lecture
  while (fichier.available()){
// Tant que ce fichier est disponible
    client.write(fichier.read());
// On demande au site web d'afficher ce qui est lu par le fichier de la
carte SD
  }
  fichier.close();
// Une fois que le fichier a été lu dans son entièreté, on le ferme
}
```

Ces différentes fonctions permettent de répondre correctement à la requête en fonction des arguments donnés.

Cependant, avec ce programme, les images chargent très longtemps. Cela est possiblement causé par les capacités limitées de la carte Arduino. Une solution envisageable mais compliquée à mettre en place serait d'héberger les images sur un serveur et d'y accéder avec la carte Arduino.

6. Partie chauffage

On souhaite allumer une LED lorsqu'on atteint un certain seuil de température, dans notre cas on l'a fixée à 20°C. Pour cela, nous avons réalisés le montage suivant :



Cependant, nous n'avons pas encore testé le bon fonctionnement du montage.

7. Conclusion / AVANCÉE

En résumé, nous avons réussi à récupérer la température d'une pièce à l'aide du capteur TMP 36 et à l'intégrer sur une page web simple. Ensuite, nous avons développé un site web complet où nous l’avons stocké sur une carte SD, connectée à la carte Arduino, ce qui nous permet de stocker des fichiers beaucoup plus volumineux.

Pour la suite du projet, notre attention se portera sur l'implémentation et l'optimisation du site web. Actuellement, la carte n'est pas suffisamment puissante pour afficher le site web dans un délai raisonnable, donc nous chercherons à résoudre ce problème. De plus, nous envisageons de travailler sur la possibilité d'allumer à distance une LED lorsque la température dépasse un seuil prédéfini.