TP de Base + Projet de Bureau d'Étude : Commande en ligne de repas du RU

RESTO U-BER

CHIRINO Melet

NAVARRO Grégoire

Université Toulouse III - Paul Sabatier
Master I EEA-SME

2020/2021 15/01/2021

Rapport rédigé à l'attention de notre professeur encadrant :
M. Perisse Thierry

Sommaire

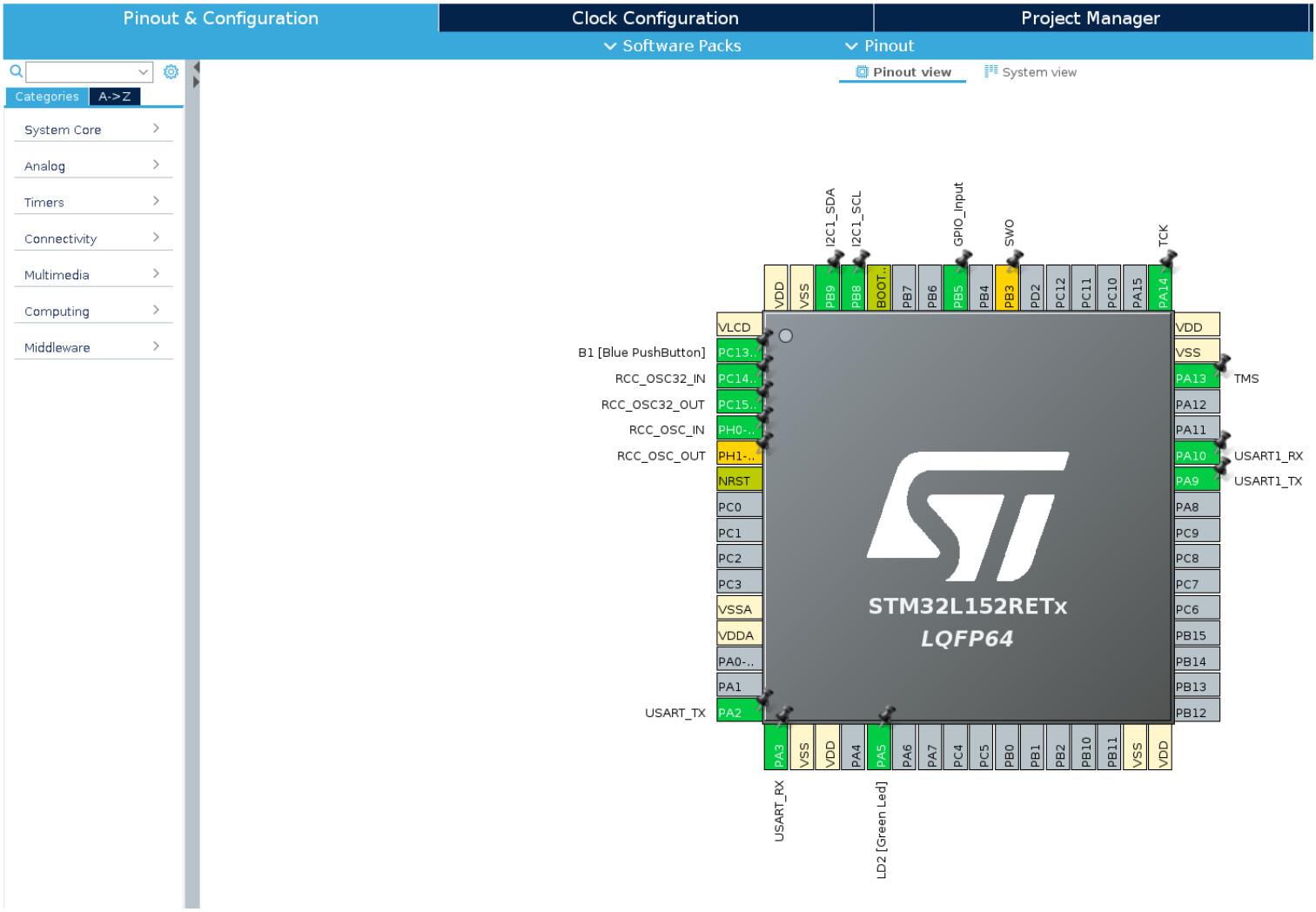
Introduction

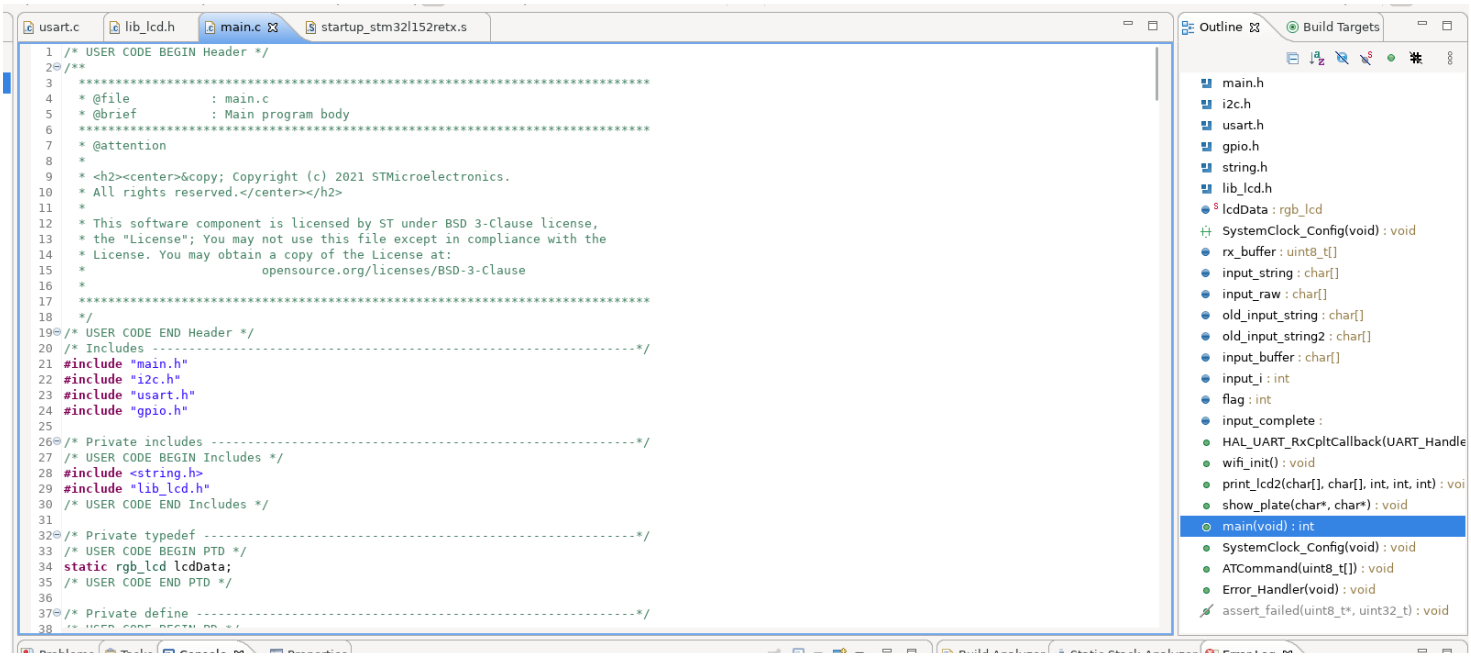
Dans le cadre de nos projets BE, Chirino Melet et moi-même, Navarro Grégoire avons initialement été amenés à réaliser un four connecté via une application. Nous avons d'abord réalisé plusieurs TP d'introduction afin de nous familiariser avec la carte STM32 ainsi que les logiciels Keil, CubeMX et TrueSTUDIO.

I. TP 1 : Clignotement d'une LED

Ce TP a été développé pour apprendre à utiliser les programmes nécessaires comme CubeMX et CubeIDE.

Les codes complets se trouvent sur github mais la partie la plus important est la suivante :





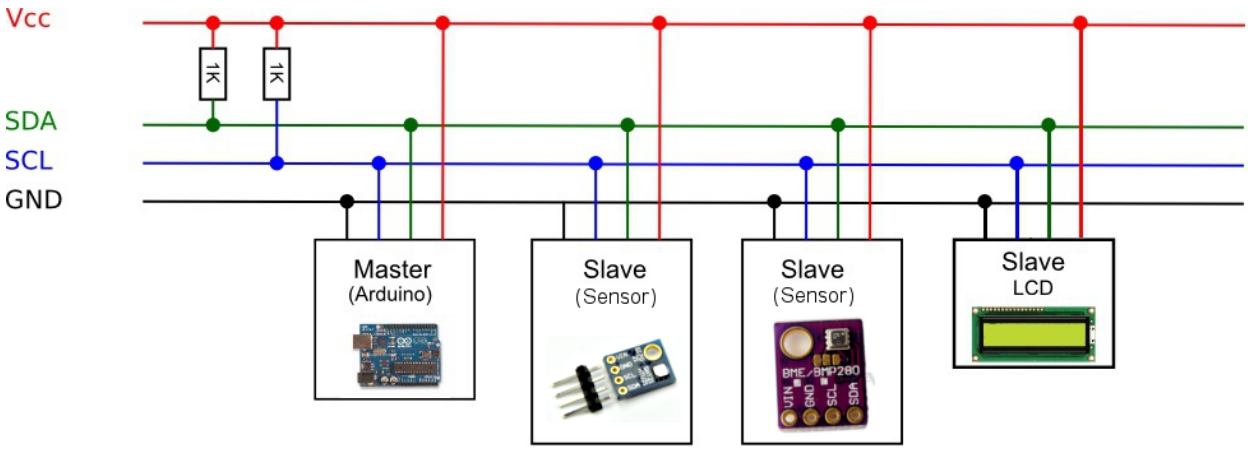
Code :

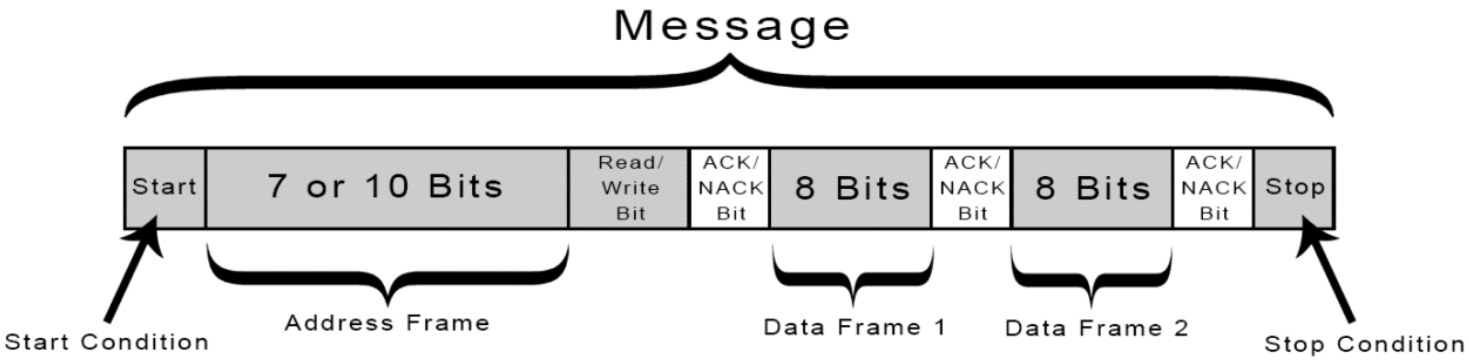
```
while (1)
{
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);
        HAL_Delay(500);
 /* USER CODE END WHILE */

 /* USER CODE BEGIN 3 */
}
```

Avec ces lignes la led clignote en utilisant le GPIO 5.

II.TP 2 et 3 : Capteur Température et Humidité (I2C)

Le capteur de température et d'humidité SHT31 marche avec l'interface I2C. L'interface I2C est utile pour un réseau d'appareils avec juste 2 câbles. La communication I2C marche avec un appareil master et des appareils "esclaves" qui répondent au master.

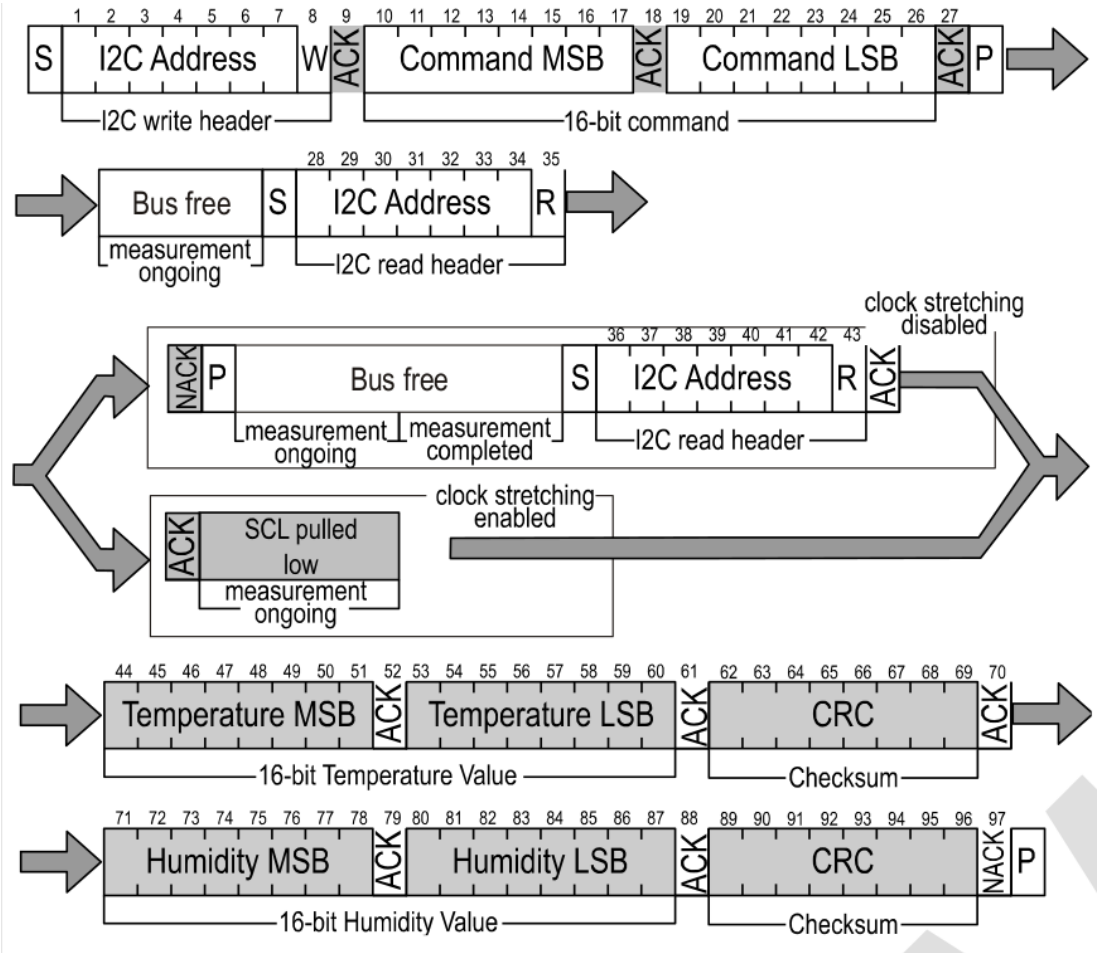




Les messages envoyés dans le bus I2C ont habituellement le bit de start, l'adresse de l'appareil et un instruction spécifique pour l'appareil envisagé. Pour nous, l'adresse principale du capteur est 0x44. Les instructions pour le capteur sont les suivantes :

Condition		Hex. code	
Repeatability	Clock stretching	MSB	LSB
High	enabled	0x2C	06
Medium			0D
Low			10
High	disabled	0x24	00
Medium			0B
Low			16

Nous avons utilisé l'instruction 0x2c06 pour recevoir les données d'humidité et de température. Le chronogramme de communication est le suivant :



Finalement, les données arrivent illisibles pour l'être humain, il nous faut donc les manipuler un peu. Le code principal que nous avons utilisé pour avoir les donnés et les manipuler est le suivant :

Code :

```
 while (1) {

ret = HAL_I2C_Master_Transmit(&hi2c1, SHT31_ADDRS,
&SHT31_MEAS_HIGHREP, 2, HAL_MAX_DELAY);

if(ret != HAL_OK)
{
		strcpy((char*)serial_text, "Error Tx\r\n");
}
else
{
		ret = HAL_I2C_Master_Receive(&hi2c1, SHT31_ADDRS, &buf, 2,
		HAL_MAX_DELAY);

		if( ret != HAL_OK)
		{

				strcpy((char*)buf, "Error Tx2\r\n");
		}
```

Suite code :

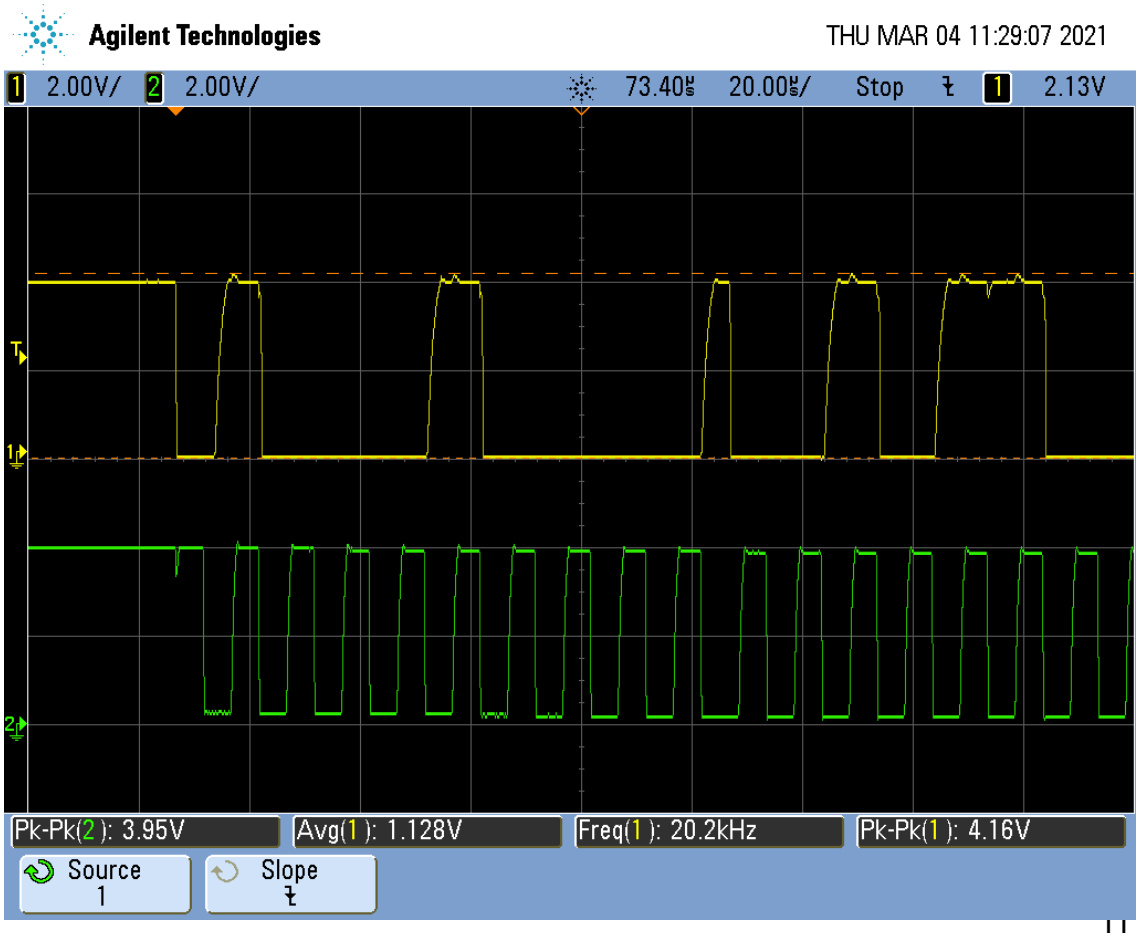
```
            else
            {
                        ST = buf[0];
                        ST <<= 8;
                        ST |= buf[1];
                        double stemp = ST;
                        stemp = ST;
                        stemp *= 175;
                        stemp /= 0xffff;
                        stemp = -45 + stemp;
                        temp = stemp;
                        sprintf((char*)serial_text,"%u.%0.2u C\n",
                        ((unsigned int) temp),
                        (((unsigned int) temp  % 100)*100));
            }
}
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
}
  /* USER CODE END 3 */
}
```

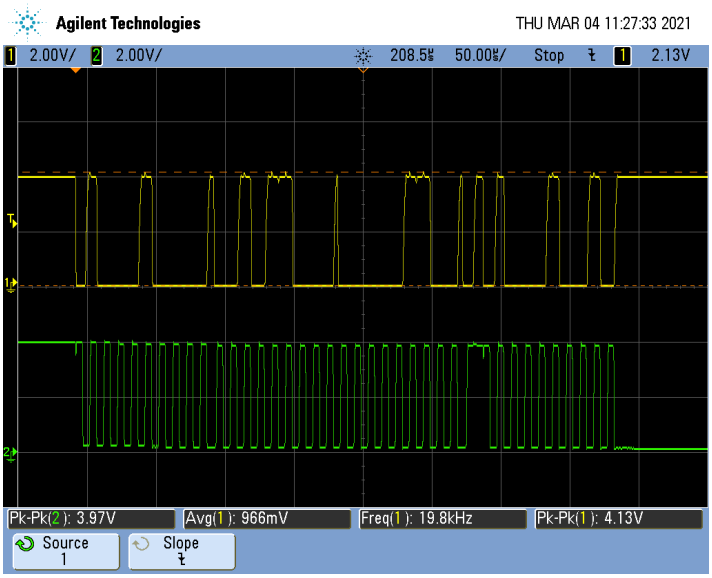
Pour le TP3, nous avons utilisé une librairie pour le LCD afin de ne pas refaire chaque commande, car cela aurait juste été plus long et compliqué. Vu que nous avions déjà les données de températures, c'était simple de les montrer avec l'écran LCD. Le code utilisé est le suivant :

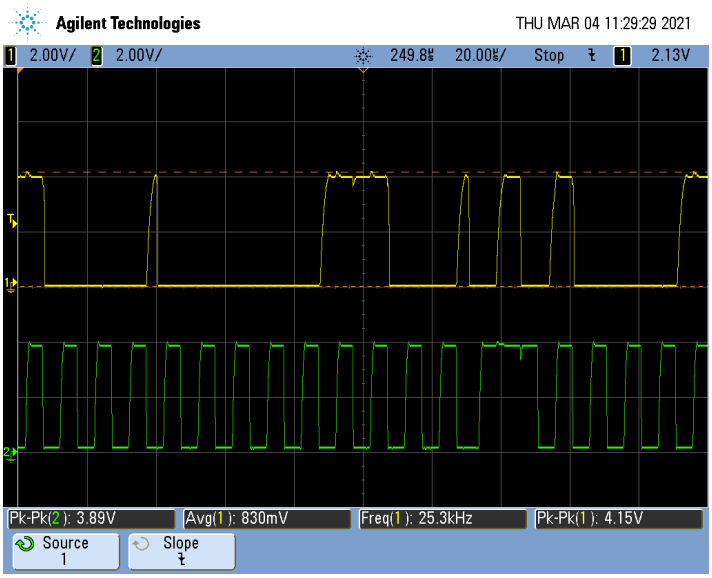
Code :

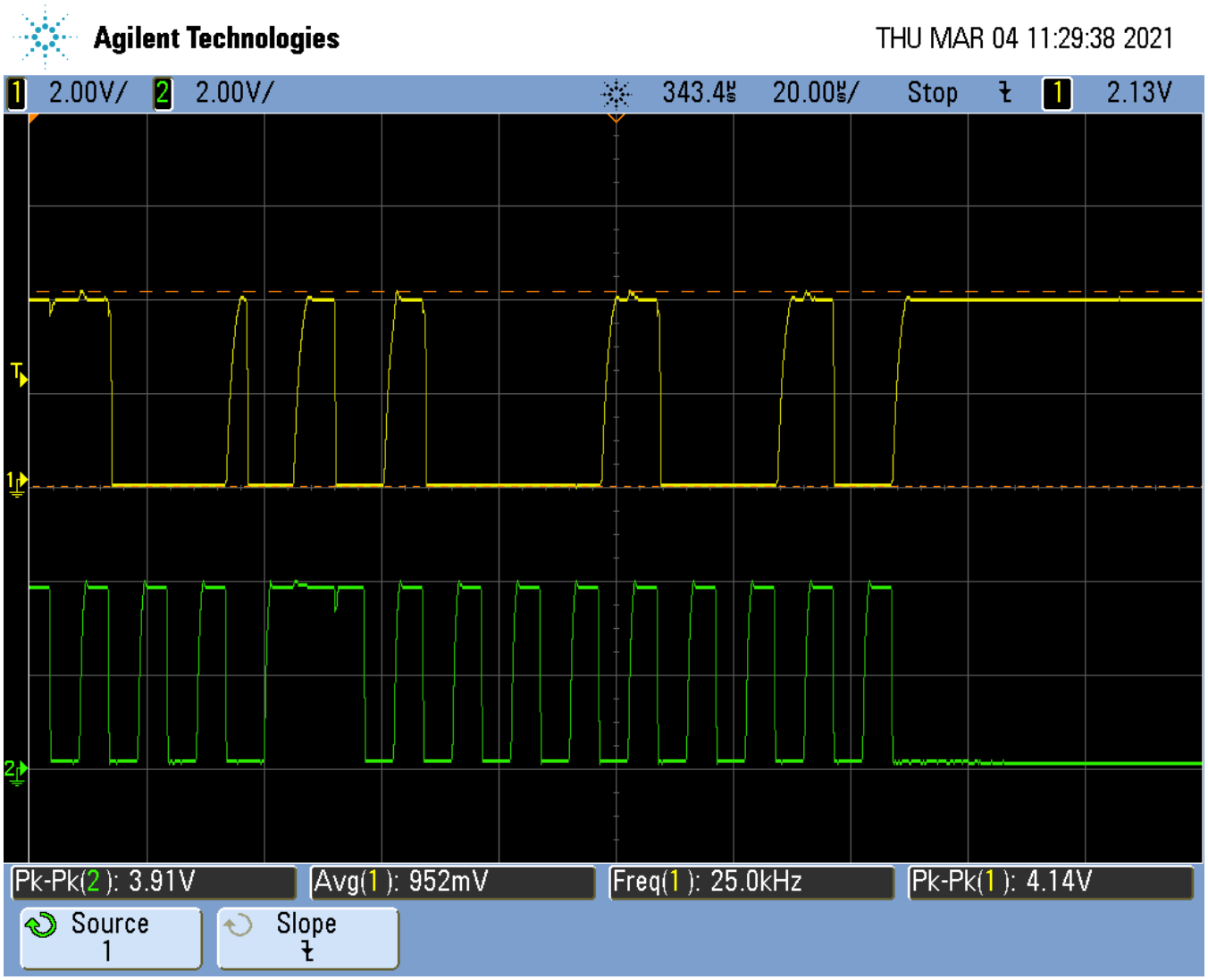
```
clearlcd();

lcd_position(&hi2c1,0,0);

lcd_print(&hi2c1,"Temp:");

lcd_position(&hi2c1,0,1);

lcd_print(&hi2c1, serial_text);

reglagecouleur(255,0,255);
```

Voilà quelques captures de l'oscilloscope :



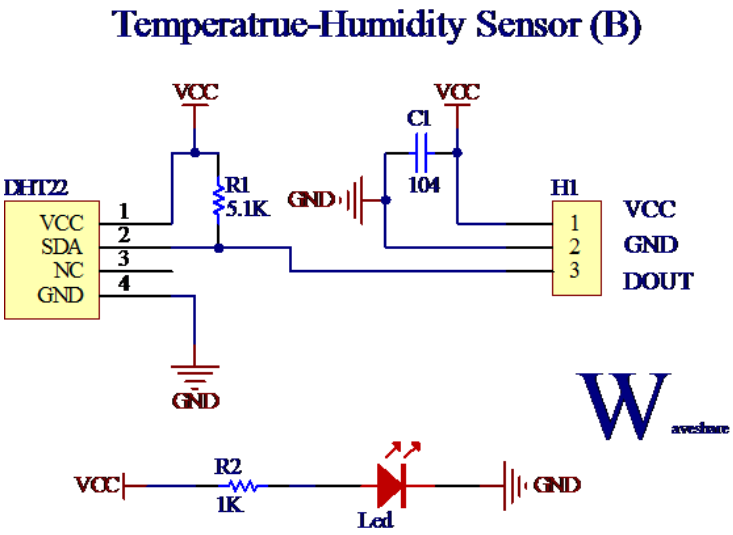

Agilent Technologies
THU MAR 04 11:27:33 2021
1 2.00V/ 2 2.00V/ 208.5ns 50.00ns/ Stop 1 2.13V
Pk-Pk(2): 3.97V
Avg(1): 966mV
Freq(1): 19.8kHz
Pk-Pk(1): 4.13V
Source 1
Slope


Agilent Technologies
THU MAR 04 11:29:29 2021
1 2.00V/ 2 2.00V/ 249.8ns 20.00ns/ Stop 1 2.13V
Pk-Pk(2): 3.89V
Avg(1): 830mV
Freq(1): 25.3kHz
Pk-Pk(1): 4.15V
Source 1
Slope

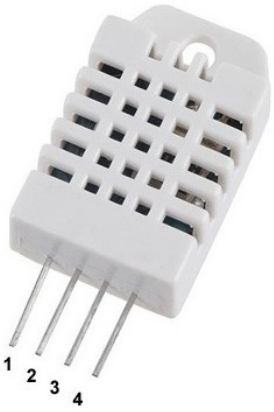

Agilent Technologies
THU MAR 04 11:29:38 2021
1 2.00V/ 2 2.00V/ 343.4ns 20.00ns/ Stop 1 2.13V
Pk-Pk(2): 3.91V
Avg(1): 952mV
Freq(1): 25.0kHz
Pk-Pk(1): 4.14V
Source 1
Slope

III. TP 2 et 3 : Capteur Température et Humidité (DHT22)

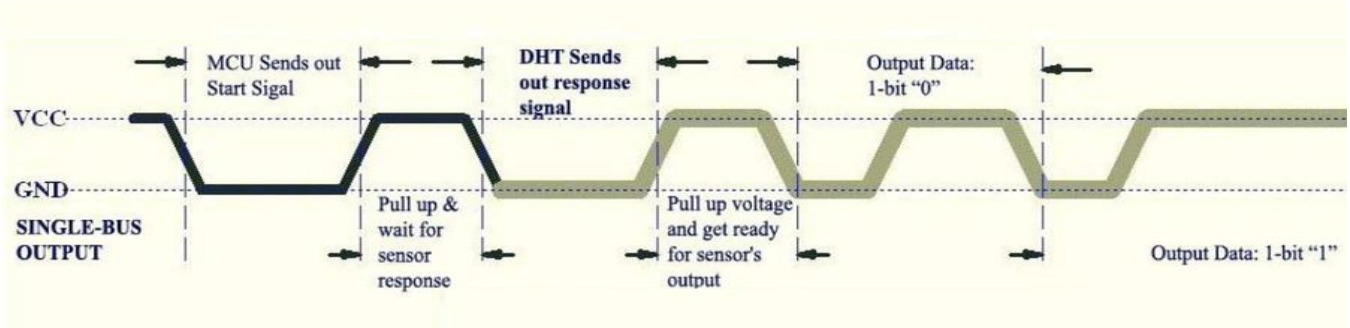
Pour cette partie, même principe que le III mais nous utilisons cette fois-ci le capteur de température et d'humidité DHT22.



DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND



Celui-ci communique grâce au bus AM2302 dans lequel une seule ligne de données est appliquée pour l'échange de données et le contrôle des données dans le système. Dans les applications, une résistance de pull-up externe, d'environ 5,1 kΩ, est généralement requise. Lorsque le bus est inactif, son état passe à l'état haut. Le SDA est utilisé pour la communication et la synchronisation des données entre le microprocesseur et l'AM 2302. Il adopte un format de données à bus unique, 40 bits de données dans une transmission, bit haut premier sorti. Le chronogramme correspondant est illustré ci-dessous :



L'ensemble du code est disponible sur Github mais la partie la plus intéressante est la suivante :

Code : Le capteur récupère les valeurs

```
static void DHT22_StartAcquisition(void)
{
	ONE_WIRE_PinMode(ONE_OUTPUT);
	ONE_WIRE_Pin_Write(0);
	DelayMicroSeconds(500);
	ONE_WIRE_Pin_Write(1);
	DelayMicroSeconds(30);
	ONE_WIRE_PinMode(ONE_INPUT);
}
```

Code : Récupération des datas en "Raw" dans un tableau

```
uint32_t rawBits = 0UL;
uint8_t checksumBits=0;
```

Dans une boucle if : rawBits |= (1UL << i);

Dans une boucle if : checksumBits |= (1UL << i);

On met ensuite ces données en raw dans un tableau :

```
data[0] = (rawBits>>24)&0xFF;
data[1] = (rawBits>>16)&0xFF;
data[2] = (rawBits>>8)&0xFF;
data[3] = (rawBits>>0)&0xFF;
data[4] = (checksumBits)&0xFF;
```

IV. Projet de Bureau d'Étude

Principe et objectifs :

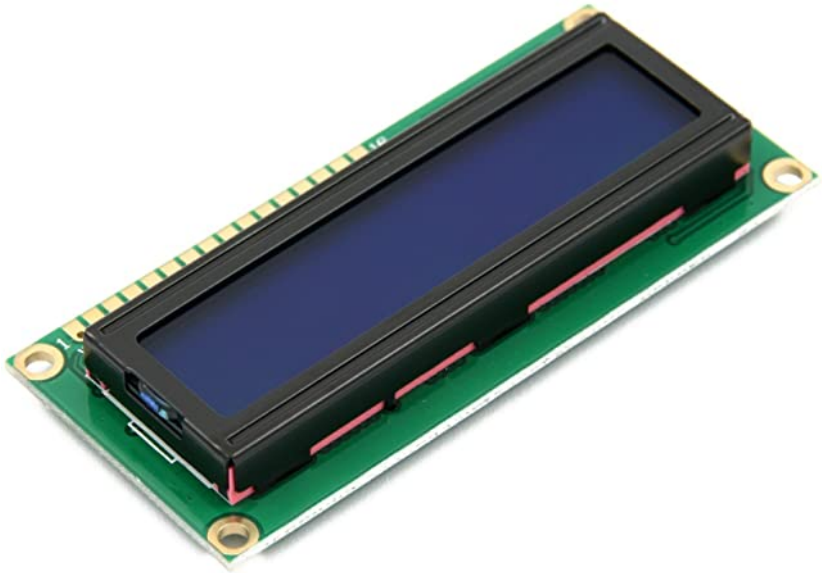
Le Restaurant Universitaire principal de l'Université Paul Sabatier est toujours plein et les étudiants passent beaucoup de temps dans la file d'attente et ne peuvent pas profiter de leur pause du midi. Le restaurant fonctionne comme une chaîne de montage où l'étudiant attend son tour dans la file puis choisit ce qu'il veut, paye et sort de l'établissement. Le problème vient du temps perdu à choisir et à payer alors que le nombre d'étudiant qui arrive ne fait qu'augmenter (il y a toujours quelqu'un qui ne peut pas prendre une décision rapide). Notre idée est que les étudiants puissent voir le menu, choisir ce qu'ils veulent et payer à partir de leur téléphone portable. Ils recevront ensuite un e-mail indiquant que leur nourriture est prête à être récupérée. Pour cela, nous utiliserons une carte de développement nucléo 152l-RE, un module Wifi (esp8266), un LCD standard qui fonctionne via I2C et un serveur web alimenté par Django (python).

Attente et fonctionnement :

Le processus devra diminuer le temps d'attente des étudiants et du personnel de cuisine. Les étudiants pourraient profiter de ce temps d'attente pour étudier ou faire autre chose qu'ils n'auraient pas pu faire s'ils étaient dans une file d'attente classique. Ils pourront également consulter le menu très tôt le matin afin de prendre leur temps pour choisir sans ralentir la file d'attente. Le temps du personnel de cuisine sera également optimisé car il n'a affaire qu'à un petit écran qui leur annoncera quoi cuisiner. Le personnel de cuisine sera également en mesure de modifier le menu en fonction des disponibilités et des plats du jour.

Module écran LCD

Le module LCD est un écran standard de 2 lignes et 16 colonnes qui fonctionne via I2C. Pour communiquer avec cet écran, nous avons utilisé une bibliothèque fournie par notre professeur qui fonctionne très bien, car son processus d'initialisation est assez compliqué à implémenter ligne par ligne. Cet écran affichera les commandes du personnel de cuisine dans un ordre chronologique.



Module Serveur Web

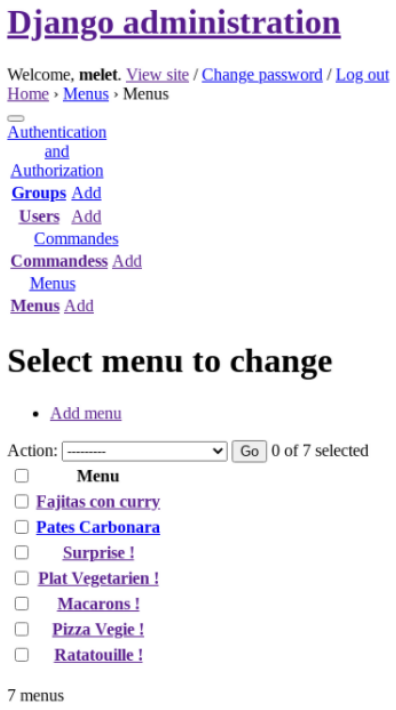
Le serveur web a été flashé en utilisant le framework Django, écrit en python. Ce cadre nous permet d'utiliser des scripts python pour gérer la base de données et la communication réseau tout en utilisant des requêtes HTTP régulières pour communiquer avec un appareil IoT. Le flux régulier de ce framework accepte une requête HTTP, exécute un script python, vérifie une base de données et finalement renvoie une réponse (généralement une réponse http) pour afficher le site Web avec toutes les informations demandées.

Puisqu'il s'agit d'un projet académique, ce serveur Web a été déployé dans un serveur de développement local, ce qui signifie que le serveur ne peut pas traiter plus de 1000 demandes en même temps et qu'il a été lié à un domaine via la redirection de port. Fondamentalement, ce serveur fonctionne sur nos ordinateurs et il doit être activé et désactivé pour les tests. IL N'EST PAS HÉBERGÉ SUR INTERNET. Dans ce projet, nous avons déclaré 3 pages principales: page de destination, page d'administration et page api.

Page d'accueil :

C'est la page à laquelle accèdent les étudiants pour commander leur nourriture, elle est assez simple et conviviale. On vérifie la base de données du menu à chaque fois qu'une demande est faite, puis on affiche ces informations dans le frontend (Les textes sur la page Web). Cette page a un lien direct vers la page d'administration.

Page d'administration :

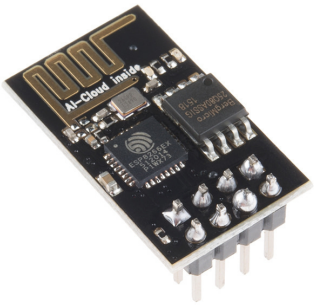


C'est la page qui affiche la base de données et les informations du menu. Ce n'est pas compliqué mais Django permet une évolutivité si nécessaire. Ce framework (Structure) offre également un middleware (Logiciel qui protège les données des utilisateurs) d'authentification assez sûr, nous ne nous sommes donc pas inquiétés de la sécurité des données à ce niveau. Nous utilisons une simple base de données sqlite3, mais elle peut être changée en une base de données PostgreSQL pour offrir une application Web plus robuste.

Page API :

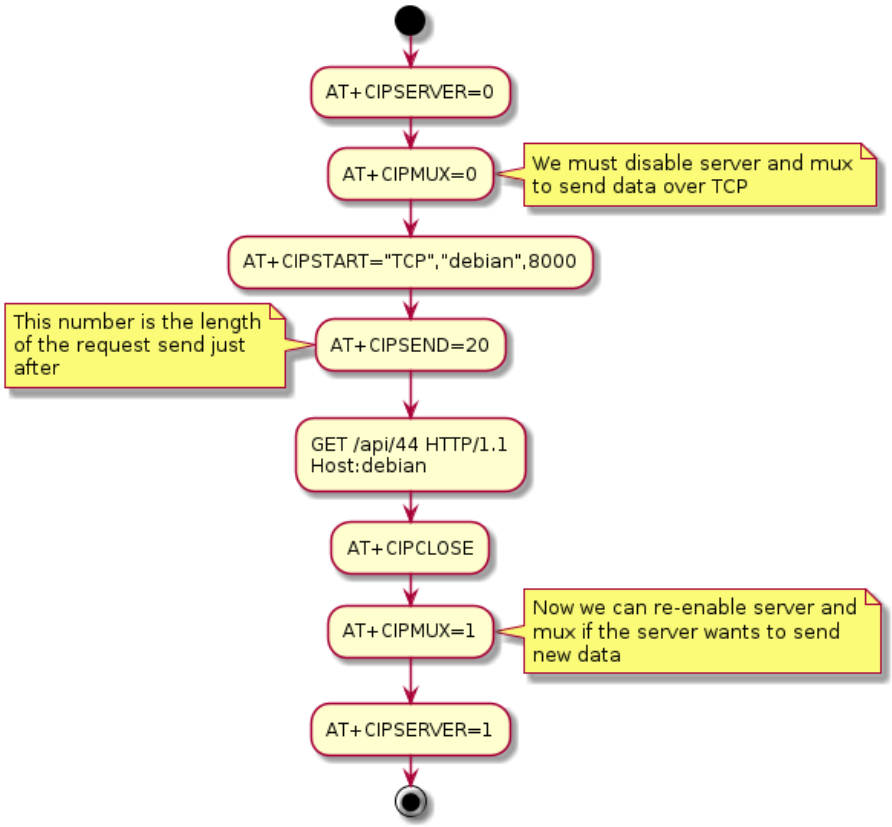
Cette page API ne fonctionnera pas à partir d'un navigateur Web, elle est destinée à recevoir une communication du module wifi et exécute un script python pour informer l'étudiant que sa nourriture est prête, puis renvoie les données de la file d'attente (s'il y en a). À l'heure actuelle, il envoie des informations directement à l'adresse IP du module wifi via le protocole TCP, donc l'adresse IP statique du module wifi doit être configurée avant. Cette API utilise un service SMTP de messagerie pour envoyer des notifications aux étudiants.

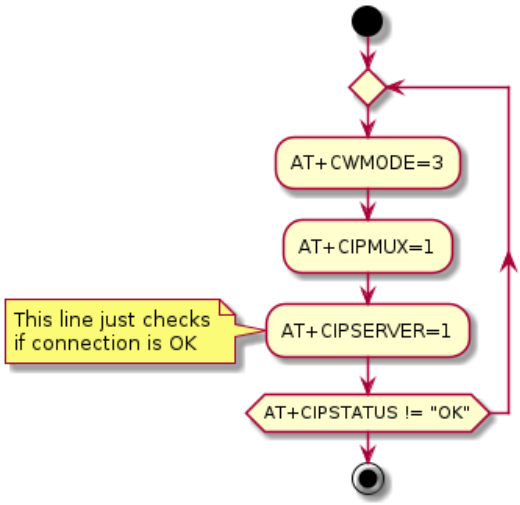
Module Wifi



Ce module wifi est l'ESP8266. C'est un microcontrôleur qui communique via une communication série. Afin d'interagir avec lui, certaines commandes peuvent être utilisées pour entrer et sortir des informations sur le réseau, plus d'informations sont disponibles dans sa fiche technique (très complète).

Ses commandes peuvent être programmées directement sur le module avec un convertisseur TTL par communication série ou avec une carte arduino pour se connecter au wifi et obtenir des informations brutes, mais notre objectif est de le faire fonctionner avec la carte stm32. Pour des raisons de sécurité, dans ce projet, nous n'allons pas définir d'authentification par mot de passe. Ce module Wifi prend en charge les protocoles de communication UDP et TCP, le dernier choisi pour ce projet. Les algorithmes de configuration, de communication et d'authentification sont présentés ci-dessous :





STM32 Serial Protocol

Puisque nous utilisons STM32 avec Cube IDE, nous devons utiliser des gestionnaires HAL pour les protocoles de communication. Ces gestionnaires simplifient beaucoup le code, cependant, les gestionnaires pour UART ont besoin d'une longueur spécifiée de la chaîne qu'ils vont lire et cela crée un problème car le module wifi que nous utilisons n'envoie pas toujours la même quantité de données et cela peut créer des erreurs. Afin de résoudre ce problème, nous avons utilisé le gestionnaire avec des interruptions à chaque octet reçu dans le tampon série. Il lit chaque octet et le colle dans une chaîne jusqu'à ce qu'il lit un '\ n'. Ensuite, il lève un drapeau indiquant que nous pouvons utiliser la chaîne lue dans le tampon série.

Diagrammes :

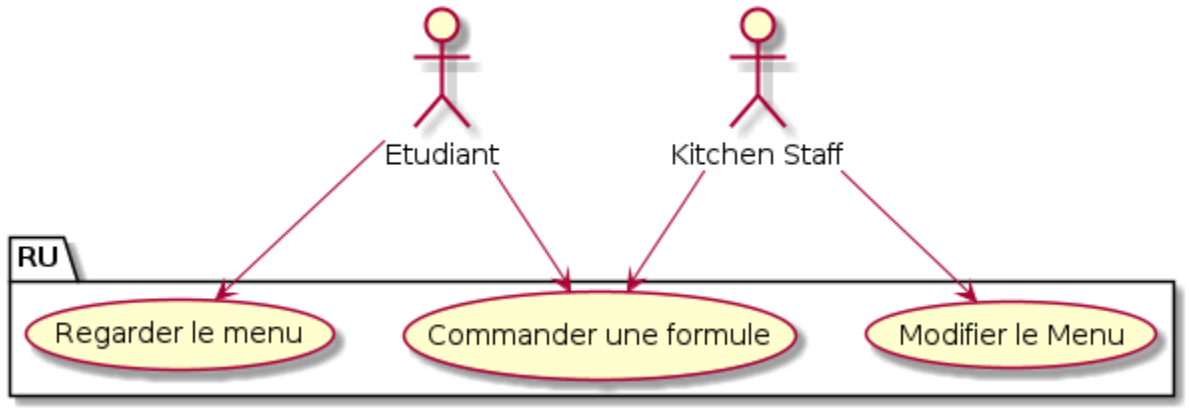


Fig. 1 – Diagramme de cas d'utilisation Utilisateurs - RU

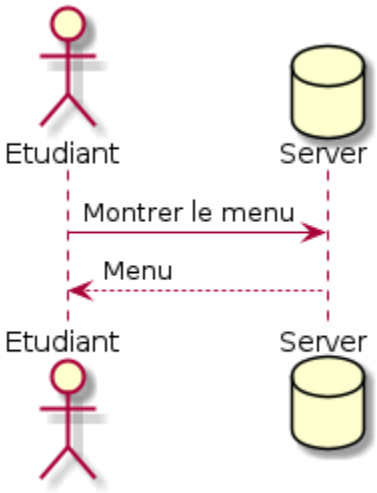


Fig. 2 – Diagramme de cas d'utilisation Utilisateurs - Application Server

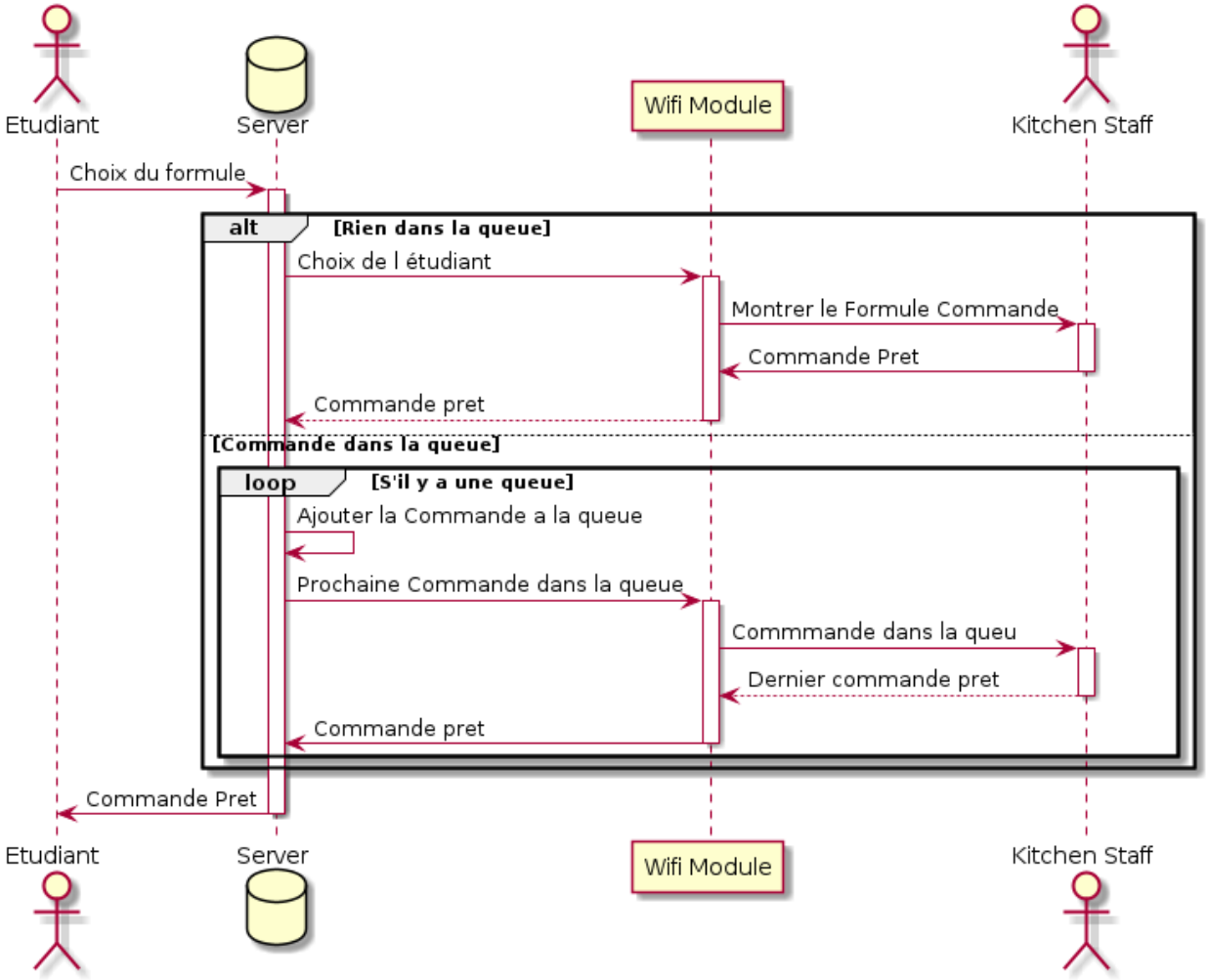


Fig. 3 – Diagramme de cas d'utilisation Utilisateurs - Wifi

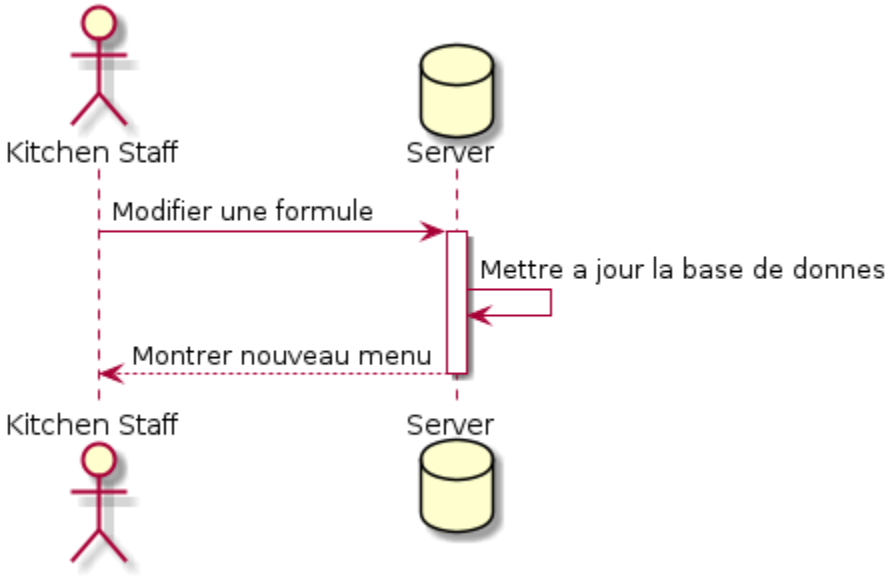


Fig. 4 – Diagramme de cas d'utilisation MàJ Menus

V. Code

Tous les codes sont classés et disponibles sur la page github sur projet BE :

https://github.com/MeletChirino/STM32_25

VI. Vidéo explicative

Dans cette vidéo, nous démontrons le résultat final de notre projet.
La vidéo récapitulative est disponible sur youtube avec le lien suivant :

https://www.youtube.com/watch?v=uAYdncR-61Q

Conclusion

Nous avons pu réaliser correctement les différents travaux pratiques avec les deux capteurs différents et aussi terminer ce projet BE en changeant quelque peu nos objectifs et le résultat final que nous attendions. Nous avons utilisé l'idée de base du four connecté pour au final partir sur une application de livraison de nourriture rendant ce projet encore meilleur et plus intéressant. Chirino Melet et moi-même avons pu ainsi découvrir le fonctionnement d'une carte réseau mais aussi comment réaliser une interface propre et qui attire l'œil via Django.

Ce fût évidemment une expérience enrichissante et nous remercions M. Perisse pour avoir encadré ce projet et nous avoir accompagné durant les travaux pratiques.