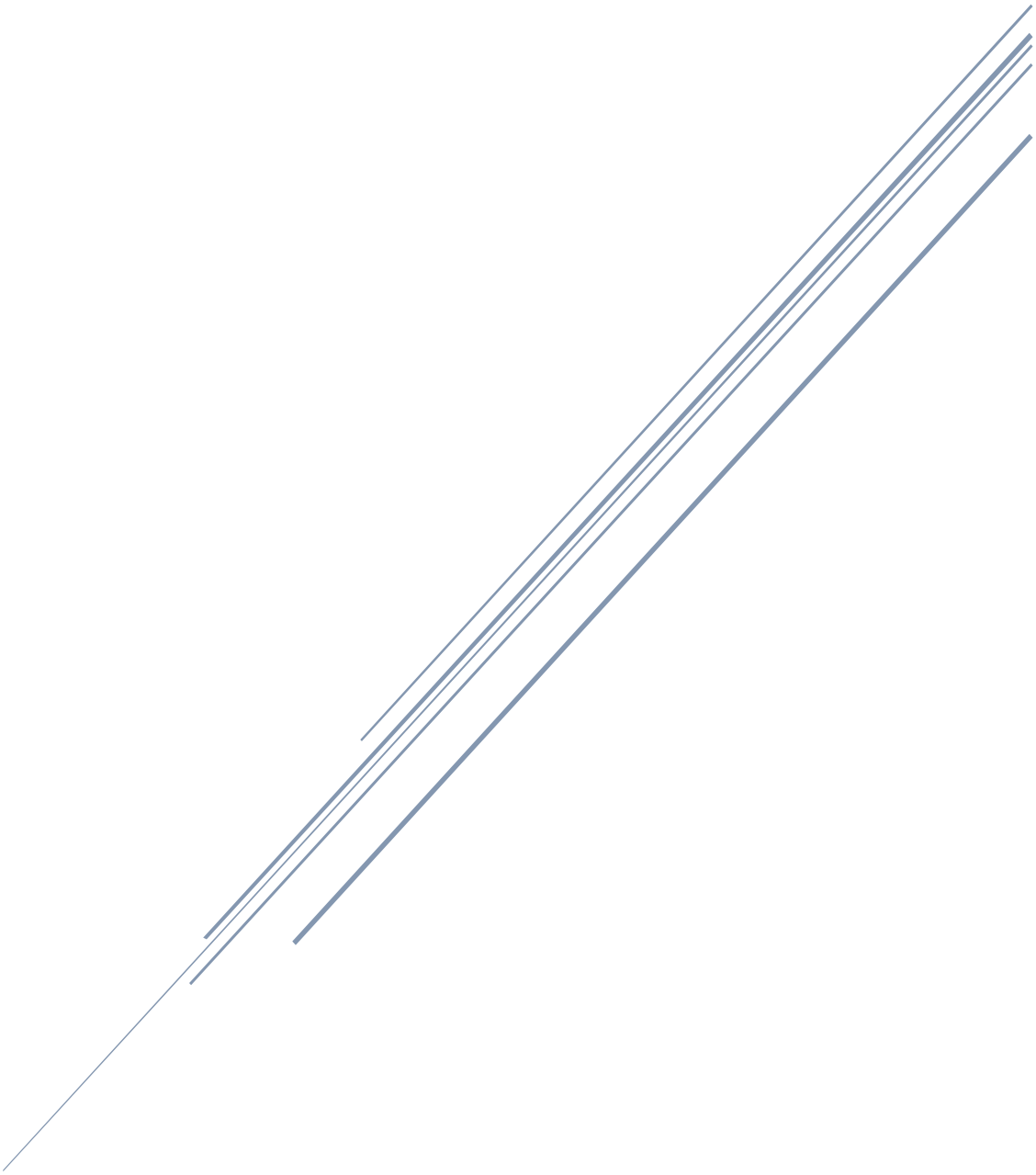
RAPPORT DE PROJET

Gestion de la consommation électrique



xxxxxxx

xxxxxxxxxxxxxxxxxxx

BTS SN-EC 2

TABLE DES MATIERES

1 Introduction 2
2 Le projet 2
2.1 Contexte 2
2.2 Problématique 3
2.3 Enedis et les compteurs 3
2.4 Cahier des charges 3
2.5 Répartition des tâches 3
3 Conception générale du projet 4
4 Conception électronique 5
4.1 Fonctionnement du télé info 5
4.1.1 Codage des bits 5
4.1.2 Format de transmission d'un octet 6
4.1.3 Format d'une trame télé info 6
4.2 Voltampère et Watt 7
4.3 Choix des composants 9
4.3.1 La Raspberry PI 9
4.3.2 L'optocoupleur 9
4.3.3 Les résistances 10
4.3.4 Le bornier 10
4.4 Conception du circuit 10
4.5 Modélisation KiCad 13
5 Réalisation électronique 14
6 Conception et réalisation software 16
6.1 Le service 16
6.1.1 Protocole 16
6.1.2 Programme 18
6.2 La base de données 18
6.3 La récupération des données télé info 19
6.4 Le site web 21
6.4.1 Backend (Server web) 21
6.4.2 Frontend (Page web) 21
6.5 Gestion des programmes 22
7 Conclusion 22
8 Références 23

1 INTRODUCTION

Dans le cadre de la validation des compétences de nos deux années de BTS, nous devons concevoir et réaliser un projet qui met en œuvre les connaissances acquises.

Nous sommes en groupe et devrons utiliser les différentes matières du BTS pour mener à bien ce projet, que ce soit la physique, l'expression, les mathématiques ou encore l'anglais, et bien évidemment notre spécialité.

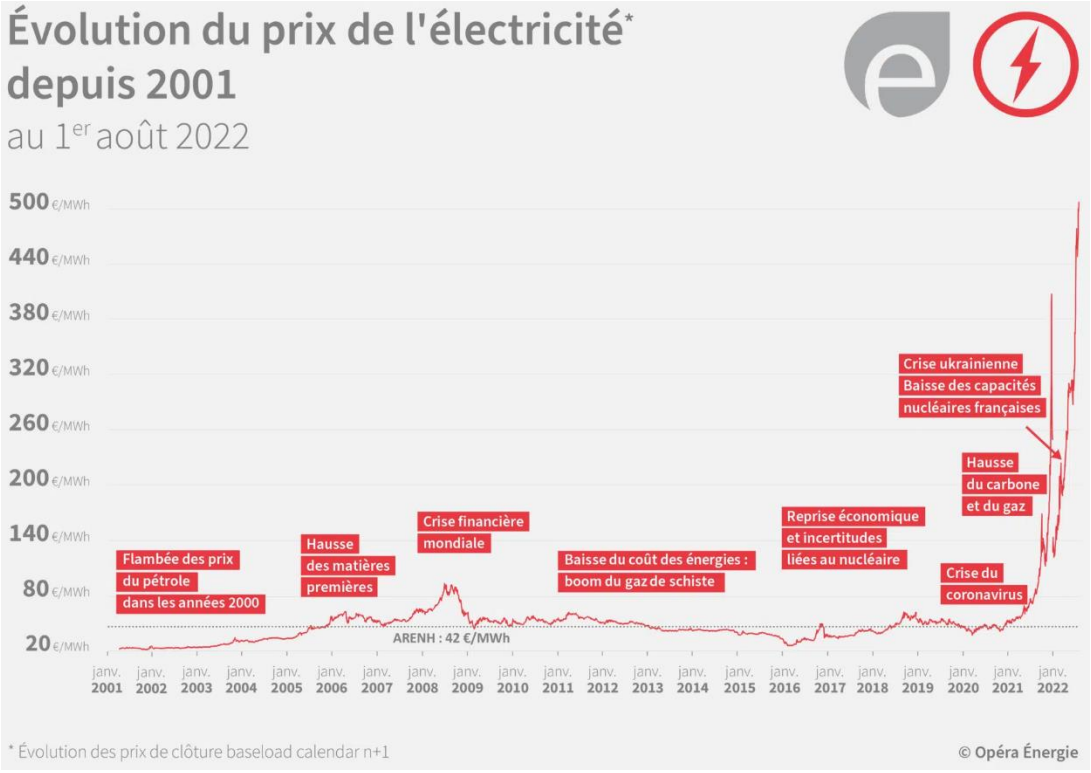
2 LE PROJET

2.1 CONTEXTE

L'énergie électrique est l'une des plus importantes et les plus consommées dans le monde. Les moyens de production sont très variés et reposent en grande partie sur des ressources non renouvelables.

La demande et la dépendance envers l'énergie électrique sont toujours plus élevées. De plus, les prix sont très sensibles à la situation économique, comme on a pu le voir lors de la crise de 2007 et celle de l'après-Covid.

Il est indispensable aujourd'hui de surveiller sa consommation électrique, même si cela n'est pas toujours évident, car l'électricité est tout simplement invisible et que les appareils de la maison peuvent avoir des cycles de consommation différents et inconnus du propriétaire.



2.2 PROBLEMATIQUE

On veut pouvoir facilement connaître la consommation énergétique de notre maison. On veut savoir quel appareil consomme le plus, savoir si le chargeur du téléphone fait une réelle différence, savoir à quel moment de la journée la consommation est la plus élevée.

Tout cela dans le but de permettre à n'importe qui de se sensibiliser et de faire attention à sa consommation électrique.

2.3 ENEDIS ET LES COMPTEURS

Enedis est l'entreprise qui fournit aux clients l'électricité. C'est le dernier maillon de la chaîne avant le consommateur. Les autres maillons sont les suivants :

- EDF, qui produit l'électricité (centrale nucléaire, barrage hydroélectrique...)
- RTE, qui la transporte, donc les pylônes électriques qui traversent la France.

Pour que ce projet soit réalisable, de manière simple et sans installation coûteuse de suivi de consommation, Enedis, avec les compteurs électroniques, donne accès à un bus de communication appelé télé info permettant le suivi en temps réel de la consommation électrique, ainsi que d'autres informations plus ou moins utiles.

Il existe également une application créée par Enedis pour suivre sa consommation en ligne si l'on possède un compteur Linky. L'inconvénient est que les données sont mises à jour toutes les 30 minutes, ce qui est assez long.

2.4 CAHIER DES CHARGES

On veut donc répondre à la problématique en créant un système embarqué qui se branche au compteur Enedis et qui retransmet à l'utilisateur les informations de consommation.

Nous devons respecter quelques points pour réaliser le projet :

1. Extraire les données télé info d'un compteur Enedis, les stocker dans une base de données relationnelle, et enfin restituer les données via un site web.
2. Géré 3 entrées et 3 sorties TOR. Les trois sorties doivent être automatiquement contrôlées selon la consommation et servir d'alerte.
3. Acquérir et restituer la température et l'humidité ambiantes.
4. Afficher grâce à un écran LCD des données utiles concernant le projet au client.

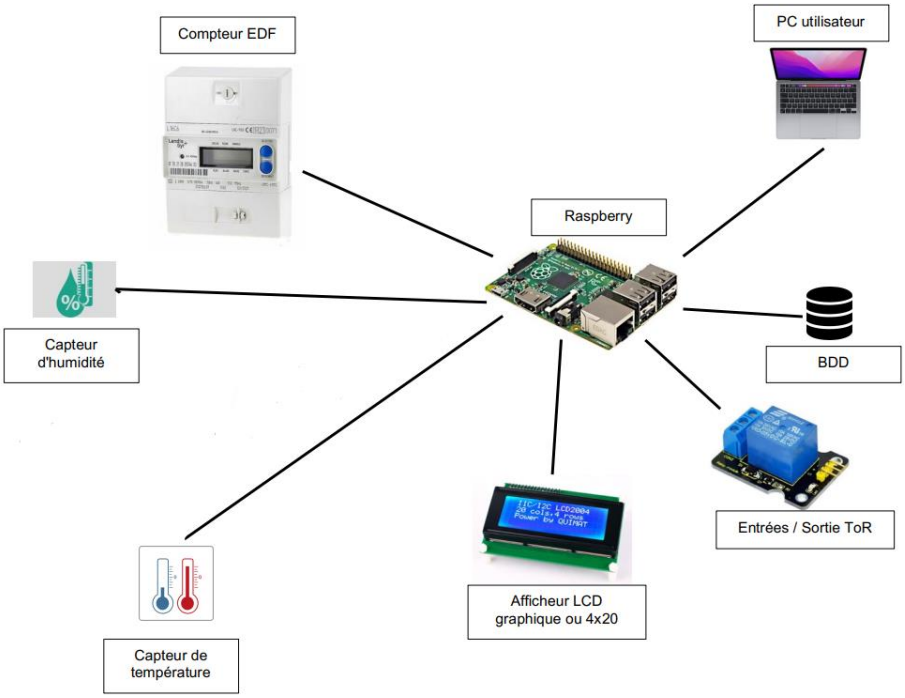
2.5 REPARTITION DES TACHES

Voici la répartition des tâches imposée par l'énoncé :

Evan	Je m'occupe du premier point, donc j'effectue l'extraction des données du télé info pour les enregistrer dans une base de données. Je dois aussi les restituer sur un site web.
Jérémy	Il s'occupe du troisième point, il doit acquérir et restituer de la manière qu'il souhaite des données de température et d'humidité.
Aléxis	Il a pour travail les points 2 et 4, il doit donc gérer 3 entrées et sorties TOR ainsi qu'un afficheur LCD. Les 3 sorties et l'afficheur doivent utiliser des données du projet.

3 CONCEPTION GENERALE DU PROJET

Voici la vison synoptique du projet que nous avons décidé :



C'est assez simple, toutes les parties du projet sont reliées à la Raspberry Pi.

Contrairement à celle proposée par le sujet, nous avons décidé de ne pas utiliser de microcontrôleurs. Étant donné que la Raspberry Pi dispose déjà d'entrées/sorties et que notre montage serait assez simple, il n'est pas nécessaire d'utiliser des microcontrôleurs pour faire le lien entre les montages et la Raspberry Pi.

Notre choix s'est porté sur la Raspberry Pi pour interpréter les données et utiliser les sorties. C'est un nano-ordinateur sur une petite carte qui fonctionne de manière indépendante. C'est un choix idéal car cela nous permet d'effectuer des opérations complexes sans trop de difficultés, comme l'utilisation d'une base de données locale, la création du serveur web ou encore la communication avec la base de données.

Nous avons également pris la décision importante de regrouper tous nos montages électroniques sur une même carte. Cela peut sembler absurde dans une situation réelle, car l'afficheur et la prise de température seraient situés là où se trouve le compteur électrique, c'est-à-dire à l'extérieur ou dans un placard. De plus, il serait nécessaire d'avoir une connexion au réseau local. Cependant, étant donné que le projet doit rester simple et que de toute façon nous n'aurions pas le temps de mettre en place un système de communication radio entre les différents éléments, nous avons choisi de tout regrouper sur une même carte.

4 CONCEPTION ELECTRONIQUE

4.1 FONCTIONNEMENT DU TELE INFO

Le télé info est un bus de communication mis en place sur les compteurs électriques d'Enedis. Il est à la disposition de tous les utilisateurs.

Le rôle du télé info est de fournir en temps réel (toutes les 2 secondes en fait) les données de consommation et d'abonnement du propriétaire.

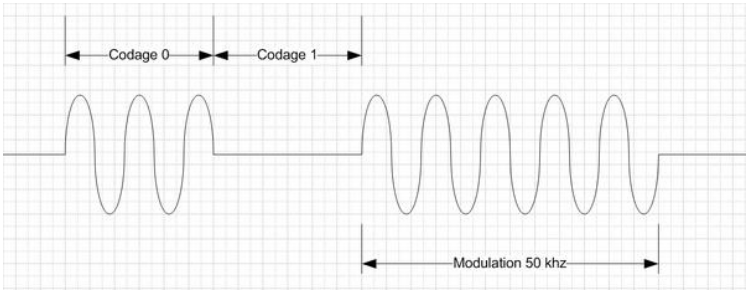
Le télé info est disponible en deux versions :

- Le mode historique, utilisé par les anciens compteurs électroniques.
- Le mode standard, qui est apparu avec les compteurs Linky.

Ils sont assez similaires à quelques détails près.

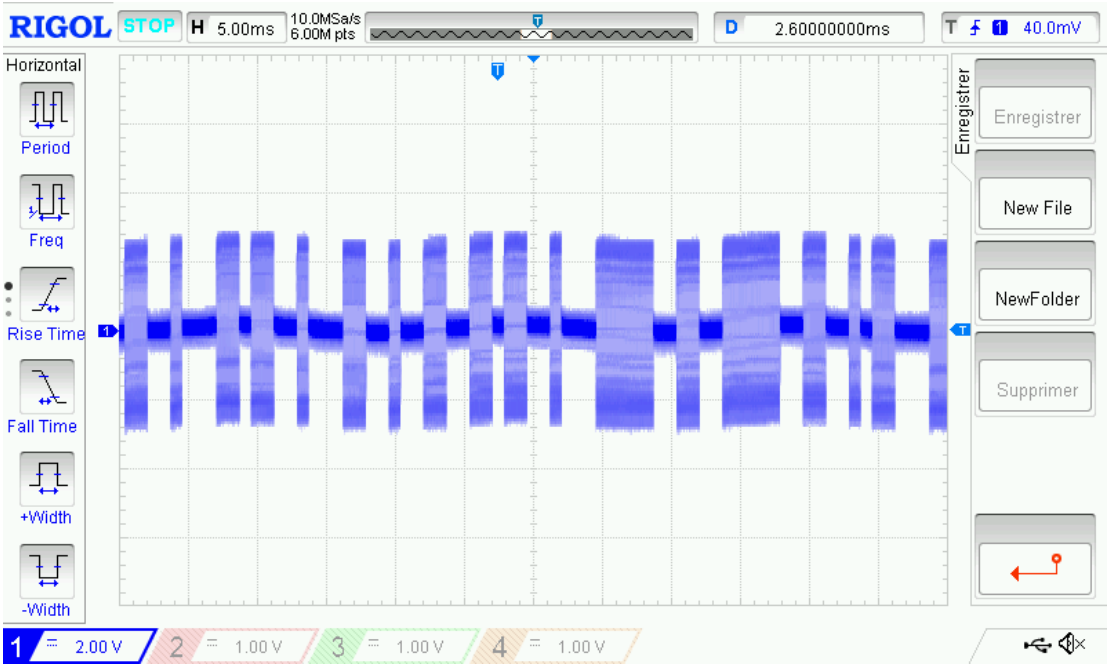
4.1.1 Codage des bits

Nous avons un 0 logique si une porteuse de 50 kHz est présente et un 1 logique si elle ne l'est pas.



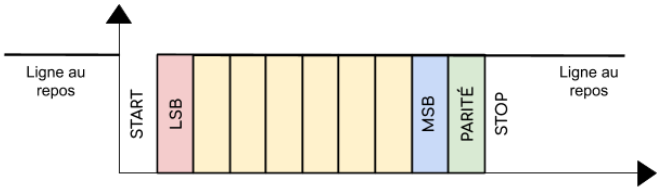
La vitesse de transmission est de 1200 bps pour le mode historique et de 9600 bps pour le mode standard.

Voici une mesure faite à l'oscilloscope avec une sonde différentielle qui montre les bits d'une trame :



4.1.2 Format de transmission d'un octet

Le format d'un octet est identique à celui des protocoles UART et RS232, ce qui signifie qu'il comprend un bit de start, 8 ou 7 bits de données, 1 bit de parité et 1 ou 2 bits de stop.



Pour le mode historique et standard, le format d'un octet est le suivant :

- 1 bit de start
- 7 bits de données
- 1 bit de parité paire
- 1 bit de stop

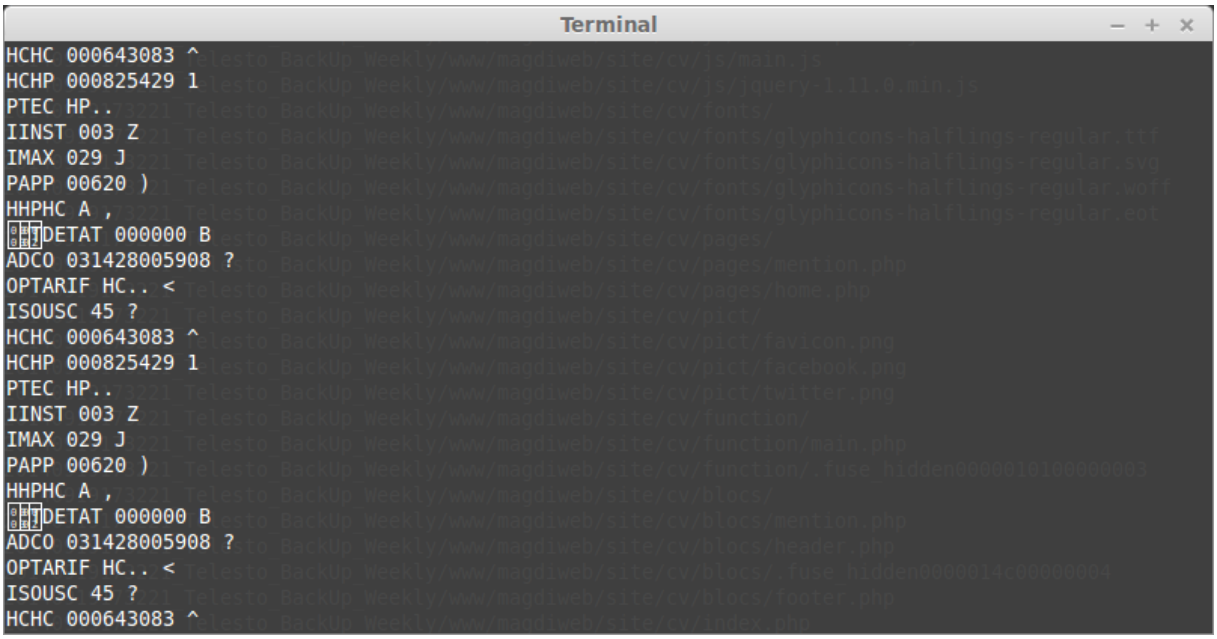
Les concepteurs du protocole ont choisi d'utiliser 7 bits de données car seuls des caractères ASCII sont transmis, et la valeur d'un caractère ASCII ne nécessite que 7 bits au maximum.

4.1.3 Format d'une trame télé info

Une trame télé info est constituée de plusieurs lignes. Chaque ligne est constituée de trois parties qui sont séparées par un espace :

- Étiquette : Nom de la valeur
- Valeur : Valeur correspondante à l'étiquette
- Checksum : Contrôle des erreurs

Chaque trame commence par l'identifiant ADCO, qui indique le numéro de série du compteur. Voici un exemple :



La ligne qui semble la plus intéressante est IINST, qui représente l'intensité instantanée. Cependant, étant donné que le télé info ne fournit que la puissance apparente (PAPP), nous sommes tentés de l'utiliser dans un calcul de puissance, comme le font certains tutoriels en ligne. Cependant, cela n'est

pas précis car nous n'avons pas les décimales (ce qui entraîne une erreur de 200 W) et nous n'avons pas non plus le déphasage. Le résultat obtenu sera toujours égal ou supérieur à la puissance apparente, ce qui n'a donc aucun intérêt.

Les index heures creuses et heures pleines sont très intéressants car ils fournissent la puissance consommée en Wattheures (Wh) depuis l'installation du compteur. Ils nous permettent donc d'avoir une mesure précise de la consommation sur une période donnée en faisant la différence entre deux relevés.

Voici une liste documentée des lignes du télé info, leur signification et unité :

Désignation	Étiquette	Nombre de caractères	Unité	Compteur Linky monophasé	
				Contrat historique	Contrat non historique
Adresse du compteur	*ADCO*	12		ADS	
Option tarifaire choisie	*OPTARIF*	4		Selon contrat	"BASE"
Intensité souscrite	*ISOUSC*	2	A	PREF (en VA)/200 V	
Index option Base	*BASE*	9	Wh	Index fournisseur 1	Index Totalisateur
Index option Heures Creuses Heures Creuses Heures Pleines	 *HCHC* *HCHP*	 9 9	 Wh Wh	 Index fournisseur 1 Index fournisseur 2	 NON TRANSMIS
Index option EJP Heures Normales Heures de Pointe Mobile	 *EJPHN* *EJPHPM*	 9 9	 Wh Wh	 Index fournisseur 1 Index fournisseur 2	 NON TRANSMIS
Index option Tempo Heures Creuses Jours Bleus Heures Pleines Jours Bleus Heures Creuses Jours Blancs Heures Pleines Jours Blancs Heures Creuses Jours Rouges Heures Pleines Jours Rouges	 *BBRHCJB* *BBRHPJB* *BBRHCJW* *BBRHPJW* *BBRHCJR* *BBRHPJR*	 9 9 9 9 9 9	 Wh Wh Wh Wh Wh Wh	 Index fournisseur 1 Index fournisseur 2 Index fournisseur 3 Index fournisseur 4 Index fournisseur 5 Index fournisseur 6	 NON TRANSMIS
Préavis Début EJP (30 min)	*PEJP*	2	min	"30", en préavis EJP	NON TRANSMIS
Période Tarifaire en cours	*PTEC*	4		Selon contrat et tarif	"TH.."
Couleur du lendemain	*DEMAIN*	4		Selon annonce, en Tempo	NON TRANSMIS
Intensité Instantanée	*IINST*	3	A	Courant efficace (en A)	
Avertissement de Dépassement De Puissance Souscrite	*ADPS*	3	A	Courant efficace, si IInst > IR	
Intensité maximale appelée	*IMAX*	3	A	90 (en A)	
Puissance apparente	*PAPP*	5	VA	S (en VA), arrondi à la dizaine la plus proche	
Horaire Heures Pleines Heures Creuses	*HHPHC*	1		"A"	
Mot d'état du compteur	*MOTDETAT*	6		"000000"	

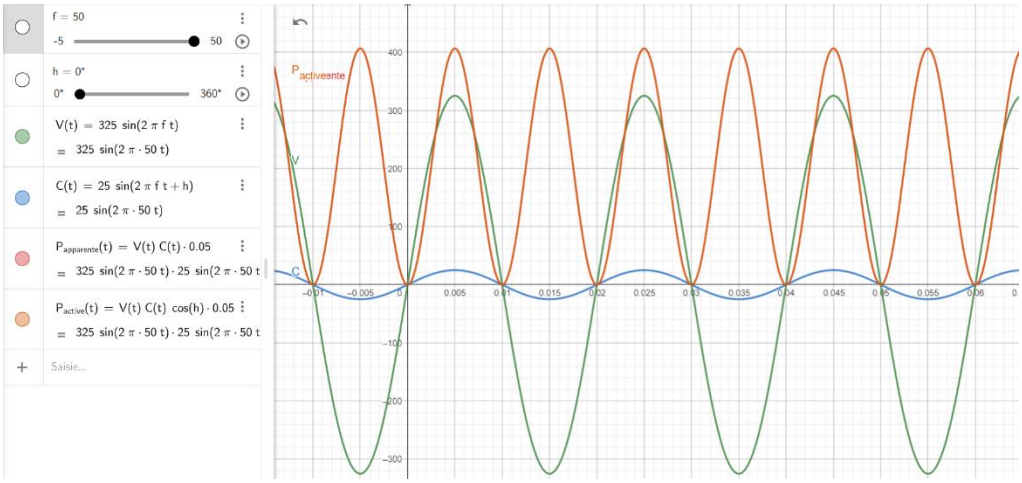
4.2 VOLTAMPERE ET WATT

La différence entre ces deux unités est assez importante et source de confusion. Sur les compteurs Linky, il est possible de voir sur l'écran que la consommation instantanée n'est pas en Watt mais en Voltampère. Il est important de distinguer les deux :

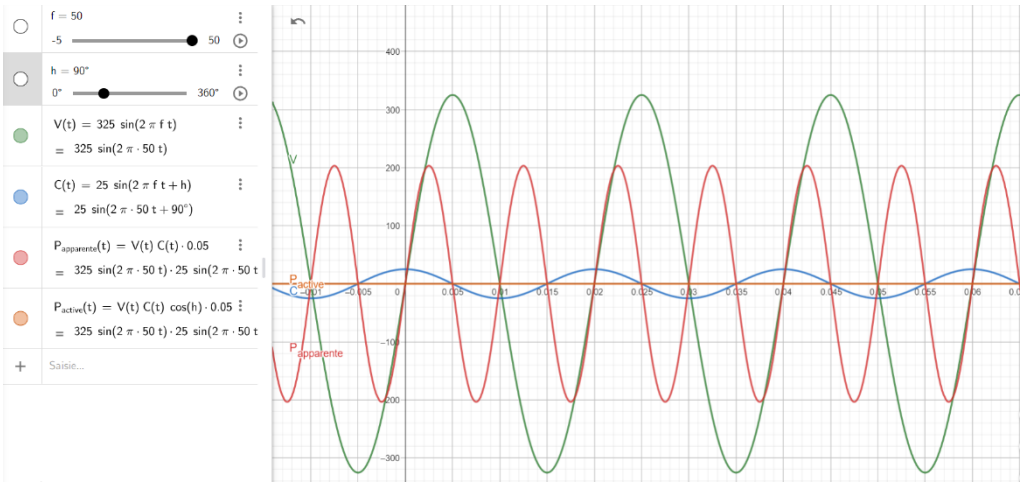
- La puissance active : exprimée en Watt, c'est la formule classique à laquelle nous sommes habitués pour le courant alternatif. Elle est calculée selon la formule $P = U * I * \cos(\phi)$.
- La puissance apparente : exprimée en Voltampère (VA), c'est ce que nous retrouvons sur nos compteurs. Elle est calculée selon la formule $P = U * I$. La puissance apparente ne prend pas en compte le déphasage entre la tension et l'intensité.

Les images suivantes ont été réalisées avec Geogebra pour illustrer mes propos.

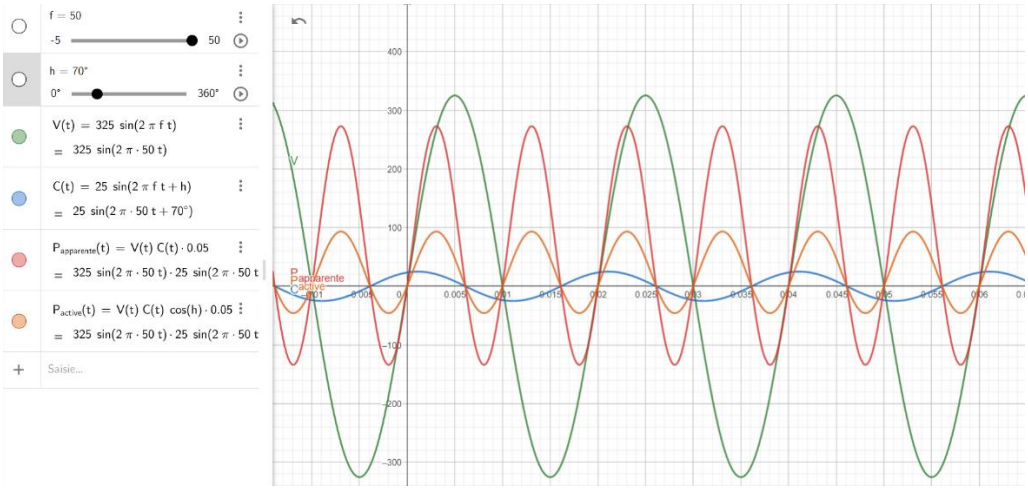
Il est important de noter qu'en pratique, les courbes de courant et de tension ne sont jamais parfaitement en phase. L'image ci-dessous présente un cas idéal sans déphasage, où seul un effet résistif est appliqué. Il est intéressant de constater que les puissances apparente et active sont égales.



En raison des effets capacitifs et inductifs, un déphasage se produit entre les deux courbes. C'est ce que l'image ci-dessous montre. Le déphasage est de 90 degrés. Étant donné que toute la puissance demandée est restituée, la puissance active est nulle car rien n'est consommé.



Ci-dessous, l'image présente un déphasage de 70 degrés. On peut remarquer qu'en cas de déphasage, la puissance apparente est plus élevée que la puissance active.



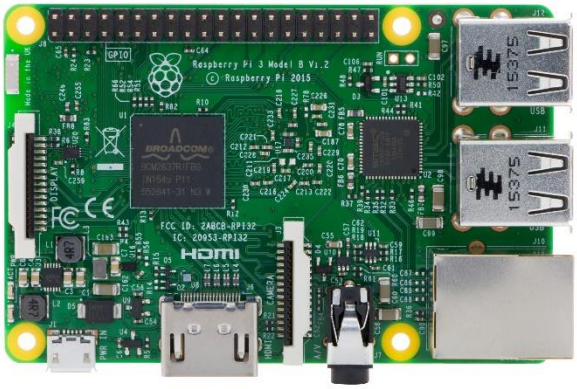
Nous sommes facturés uniquement sur l'électricité que nous consommons, c'est-à-dire sur la puissance active (en Watts). La puissance apparente (en Voltampères) est utilisée pour déterminer le déphasage du client. Le déphasage est un effet nuisible pour le réseau. Demander du courant sans l'utiliser réellement et le renvoyer dans le réseau entraîne des pertes d'énergie et des instabilités.

Cette mesure de puissance active permet au gestionnaire d'énergie d'anticiper et d'appliquer des pénalités à ceux qui causent des déphasages importants sur le réseau. Il peut également mettre en place des compensateurs de puissance réactive, tels que des condensateurs ou des moteurs électrique, pour « rephaser » le courant et réduire les pertes et les instabilités.

4.3 CHOIX DES COMPOSANTS

4.3.1 La Raspberry PI

Pour gérer la lecture et l'écriture des différents capteurs et modules, nous utilisons une Raspberry Pi 3. Nous avons choisi la Raspberry Pi car nous l'avons déjà utilisée dans notre formation. De plus, ce type de carte est assez coûteux, donc en utilisant le matériel à notre disposition, nous évitons des dépenses inutiles.



Comme expliqué précédemment, la Raspberry Pi est un nano-ordinateur sur une petite carte qui fonctionne de manière indépendante. Elle utilise un processeur SoC (System-on-a-Chip), ce qui signifie que le processeur intègre tous les éléments nécessaires à son fonctionnement (mémoire, interfaces de périphériques, unité logique, etc.). Cela permet de réduire considérablement la taille de la carte.

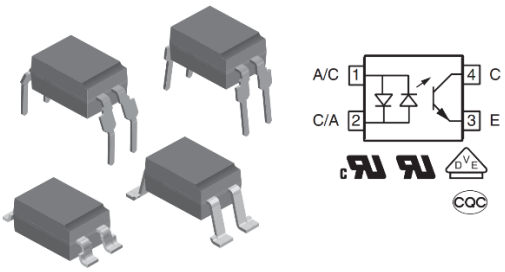
Son système d'exploitation est Raspbian, un OS spécialement conçu pour les cartes Raspberry P. Raspbian est basé sur Debian, qui est à son tour basé sur GNU/Linux.

La Raspberry Pi permet l'utilisation de GPIO (General Purpose Input/Output). De plus, la carte prend en charge nativement plusieurs protocoles de communication tels que SPI, UART, I2C, Bluetooth, etc.

La carte est très populaire et dispose d'une multitude de ressources sur internet, ce qui simplifie grandement l'utilisation.

4.3.2 L'optocoupleur

C'est le cœur du système de conversion télé info vers UART. J'ai choisi d'utiliser les optocoupleurs de référence SFH620. Le modèle SFH6206-2 a spécifiquement été choisi car il était le seul disponible en quantités raisonnables. En effet, la plupart du temps, ces optocoupleurs sont vendus en lots de 100 ou 1000 pièces.



L'avantage de cet optocoupleur est qu'il contient deux LED en dérivation de sens opposé. Cela permet au phototransistor d'être saturé lorsque l'entrée de l'optocoupleur est alimentée, peu importe la polarité. C'est utile car nous aurons en entrée un signal alternatif qui passe donc du négatif au positif.

4.3.3 Les résistances

Pour le montage, il faut deux résistances. Une résistance de 1,2kΩ pour l'entrée de l'optocoupleur, et une résistance de 10kΩ servant de pull-up. Les deux résistances sont des 1/4W.

Les valeurs indiquées seront justifiées dans la partie de conception du circuit.

4.3.4 Le bornier

Avec mon camarade Alexis, qui s'occupe des sorties TOR, nous avons choisi d'utiliser des borniers à vis. Étant donné que nous utilisons tous les deux des borniers, lui des x3 et moi des x2, nous avons opté pour les mêmes types afin de simplifier les commandes et d'avoir une carte cohérente.

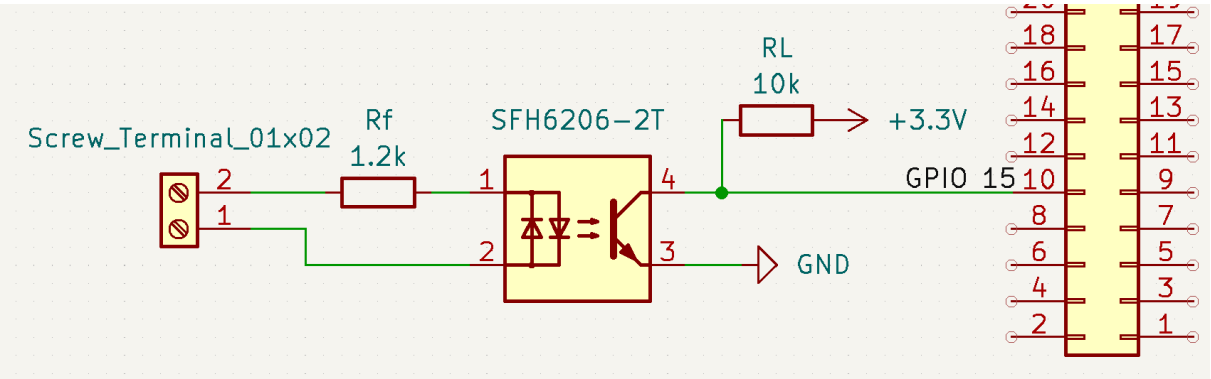
Les caractéristiques d'alimentation maximale sont de 125V alternatif pour la tension et de 10A pour l'intensité. Ces valeurs sont largement suffisantes pour ma partie, étant donné que le télé info délivre une tension de l'ordre de 3V alternatif.

4.4 CONCEPTION DU CIRCUIT

Le but est de convertir le signal télé info en signal UART. C'est un choix naturel car la principale différence entre ces deux signaux réside dans les niveaux de tension utilisés pour coder un bit. De plus, la Raspberry PI est prévue pour décoder nativement le protocole UART.

Il est nécessaire de convertir la porteuse de 50kHz en 0V et de convertir le 0V en un signal continu de 3,3 V (la tension maximale supportée par les GPIO de la Raspberry PI).

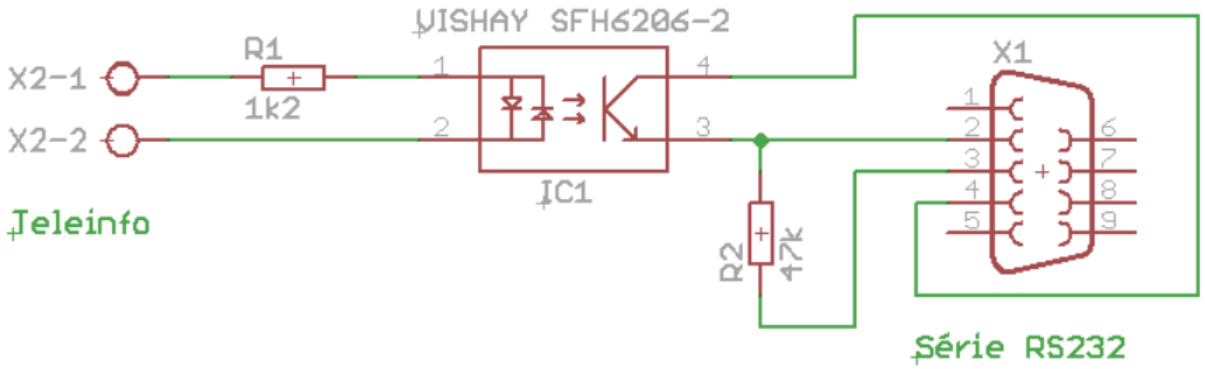
C'est ce que nous permet de faire le montage ci-dessous.



Lorsque la tension du signal télé info est à 0V, l'optocoupleur est bloqué, ce qui fait que le GPIO est connecté au 3,3V. Ainsi, la conversion du 0V au 3,3V est réalisée.

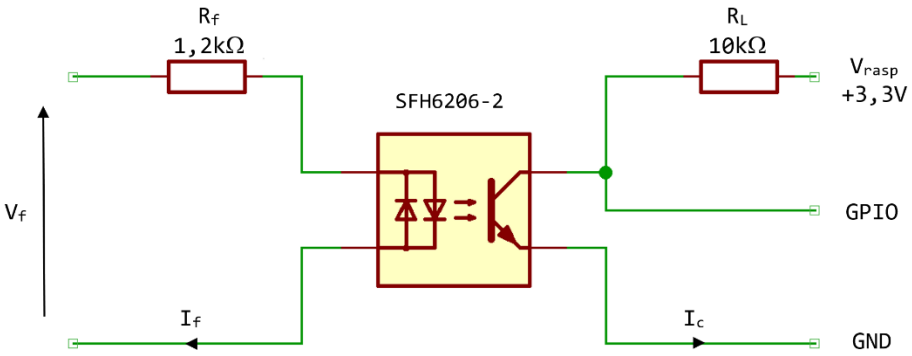
Lorsque la porteuse de 50kHz est présente, l'optocoupleur est saturé, ce qui fait que le GND est connecté au 3,3V et au GPIO. Comme le chemin ayant la résistance la plus faible est le GND, le 3,3V ne fournit plus suffisamment de courant au GPIO pour qu'il soit à l'état haut.

Le montage est basé sur le schéma ci-dessous. J'ai commis l'erreur de commander les résistances sans avoir effectué de calculs préalables. J'ai gardé la valeur de 1,2kΩ pour R1 et choisi pour R2 une valeur 3,3kΩ, car la pull-up est destinée à la Raspberry PI qui fonctionne en UART 3,3V et non en RS232.



Il s'avère que ce montage de base est conçu pour être utilisé avec un télé info ayant une porteuse de 5V alternatif. Cependant, sur le compteur Linky nous utilisons, la porteuse est de 3V alternatif. Cela a pour conséquence que le signal en entrée du GPIO 15 descend à 2V au lieu de 0V, ce qui fait que le niveau logique bas n'est jamais détecté. La valeur de la résistance Rf est trop élevée, ce qui fait que le phototransistor n'est pas suffisamment saturé pour permettre d'obtenir le niveau logique bas au GPIO.

Pour ne pas avoir à racheter de matériel, j'ai décidé de conserver la valeur de résistance R1. Voici le montage de référence pour les calculs :



Pour commencer, voici ce qu'il y a à l'entrée du SFH lorsque la porteuse est présente :

$f = 50\ kHz$

$V_{fmax} = 3\ V$

$V_{feff} = \frac{V_f}{\sqrt{2}} = \frac{3}{\sqrt{2}} = 2{,}12\ V$

$I_f = \frac{V_{feff}}{R_f} = \frac{2{,}12}{1200} = 1{,}76\ mA$

Voici la grille des ratios de transfert :

CURRENT TRANSFER RATIO (T_{amb} = 25 °C, unless otherwise specified)							
PARAMETER	**TEST CONDITION**	**PART**	**SYMBOL**	**MIN.**	**TYP.**	**MAX.**	**UNIT**
I_C/I_F	V_{CE} = 5 V, I_F = ± 10 mA	SFH620A-1	CTR	40		125	%
		SFH6206-1	CTR	40		125	%
		SFH620A-2	CTR	63		200	%
		SFH6206-2	CTR	63		200	%
		SFH620A-3	CTR	100		320	%
		SFH6206-3	CTR	100		320	%
	V_{CE} = 5 V, I_F = ± 1 mA	SFH620A-1	CTR	13	30		%
		SFH6206-1	CTR	13	30		%
		SFH620A-2	CTR	22	45		%
		SFH6206-2	CTR	22	45		%
		SFH620A-3	CTR	34	70		%
		SFH6206-3	CTR	34	70		%

Le ratio de transfert est de 45% donc :

$\frac{I_c}{I_f} = 0{,}45 \leftrightarrow I_c = I_f \cdot 0{,}45$

$I_c = I_f \cdot 0{,}45 = 1{,}76 \cdot 10^{-3} \cdot 0{,}45 = 0{,}795\ mA$

Le phototransistor ne laissera passer qu'un courant maximum de 0,795 mA, il est donc nécessaire de choisir une valeur de résistance R2 qui génère un courant inférieur à cette valeur. Si un courant excédentaire passe à travers le GPIO, il pourrait l'activer de manière incorrecte.

$\frac{V_{rasp}}{R_L} < I_c \leftrightarrow R_L > \frac{V_{rasp}}{I_c}$

$R_L > \frac{3{,}3}{0{,}795 \cdot 10^{-3}} \leftrightarrow R_L > 4150\Omega$

Donc il nous faut une résistance supérieure à 4,2 kΩ. Étant donné que nous avions des résistances de 10 kΩ à notre disposition, nous les avons utilisées.

Pour ce qui est du dimensionnement des résistances 1/4 W, voici les calculs de la consommation maximale des deux résistances :

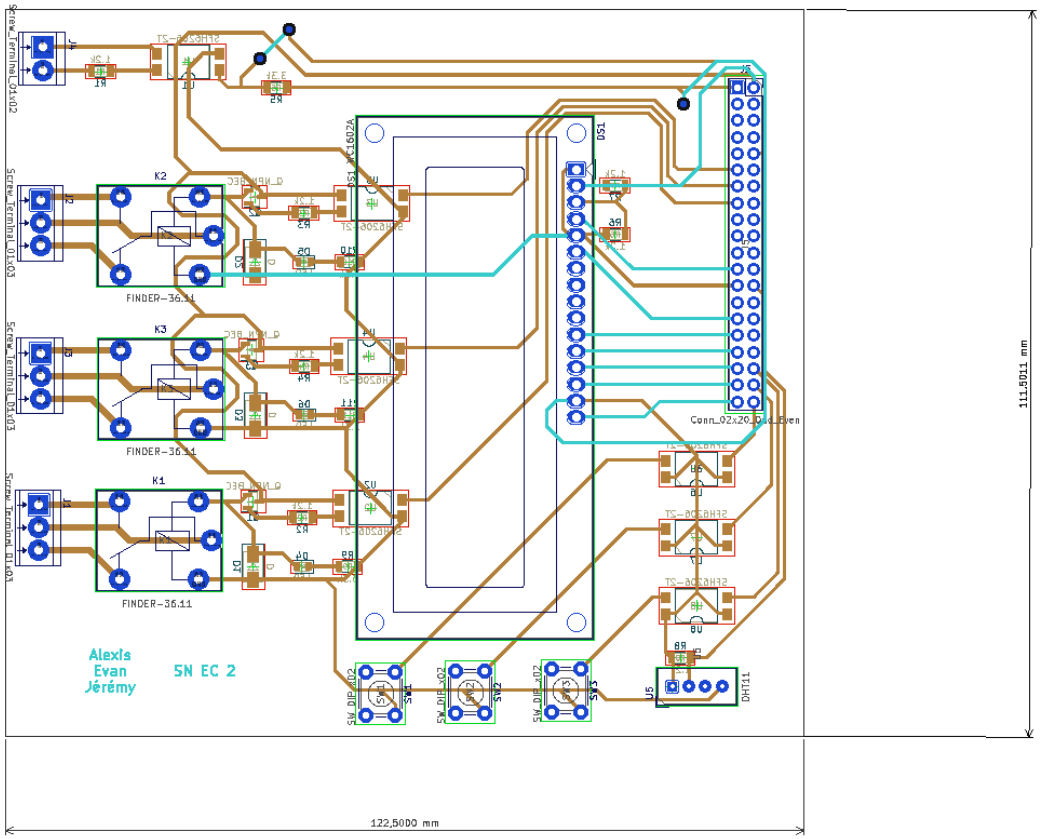
$$P_{Rf} = U \cdot I = 2{,}12 \cdot 1{,}76 \cdot 10^{-3} = 3{,}7\ mW$$

$$P_{RL} = U \cdot I = 3{,}3 \cdot \frac{V_{rasp}}{R_L} = 3{,}3 \cdot \frac{3{,}3}{10 \cdot 10^3} = 1\ mW$$

Donc $P_{RF} < \frac{1}{4}$ et $P_{RL} < \frac{1}{4}$ les résistances 1/4W sont largement suffisantes.

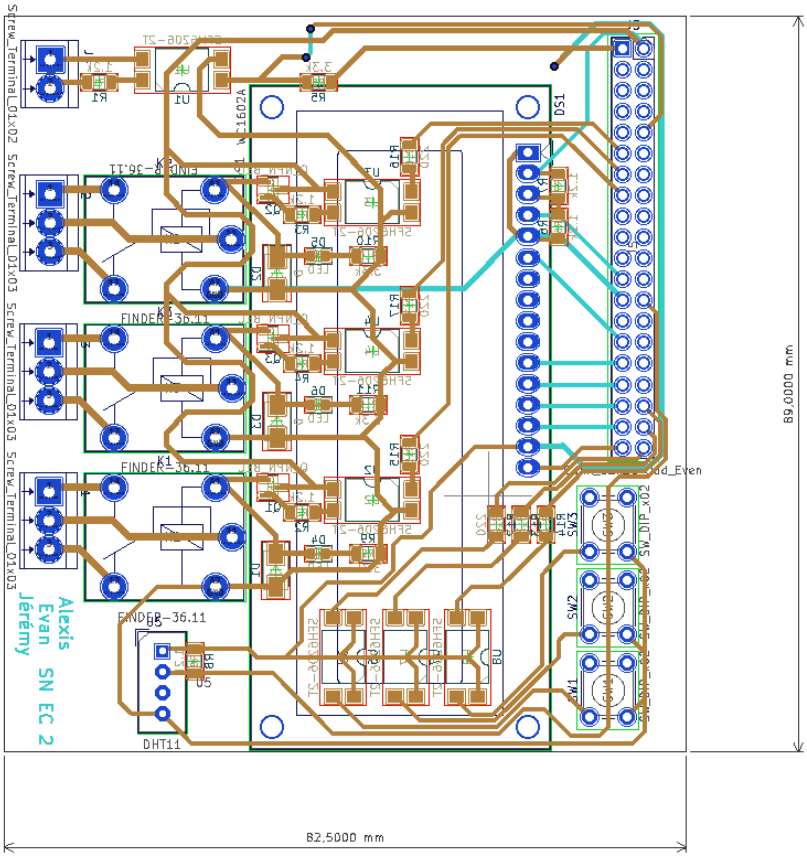
4.5 MODELISATION KICAD

Ci-dessous la première modélisation du PCB. Le seul défaut de cette version (sans prendre en compte les erreurs de schéma) est sa taille. Les dimensions sont assez grandes, ce qui rend le PCB beaucoup plus coûteux lorsqu'on souhaite le faire fabriquer par un fabricant spécialisé. De plus, étant donné que c'est une carte qui se branche sur la Raspberry PI, les deux cartes ensemble ne sont pas stables.



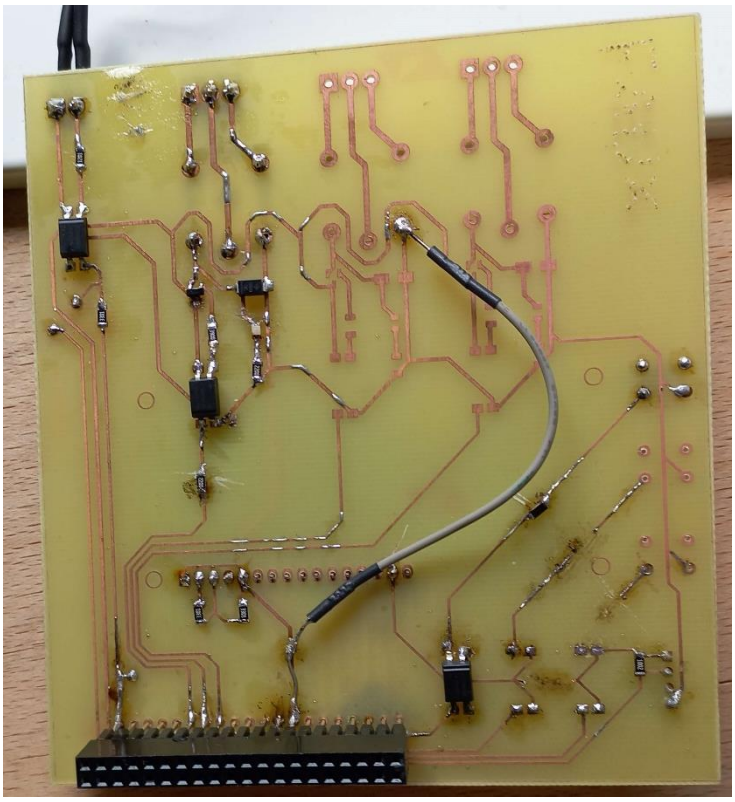
La dernière version ci-dessous corrige les problèmes de dimensions. Le connecteur 2x20 est positionné en haut à droite pour correspondre à la forme de la carte de la Raspberry PI. Les différentes parties sont regroupées : borniers ensemble, montages sorties TOR ensemble, etc.

.



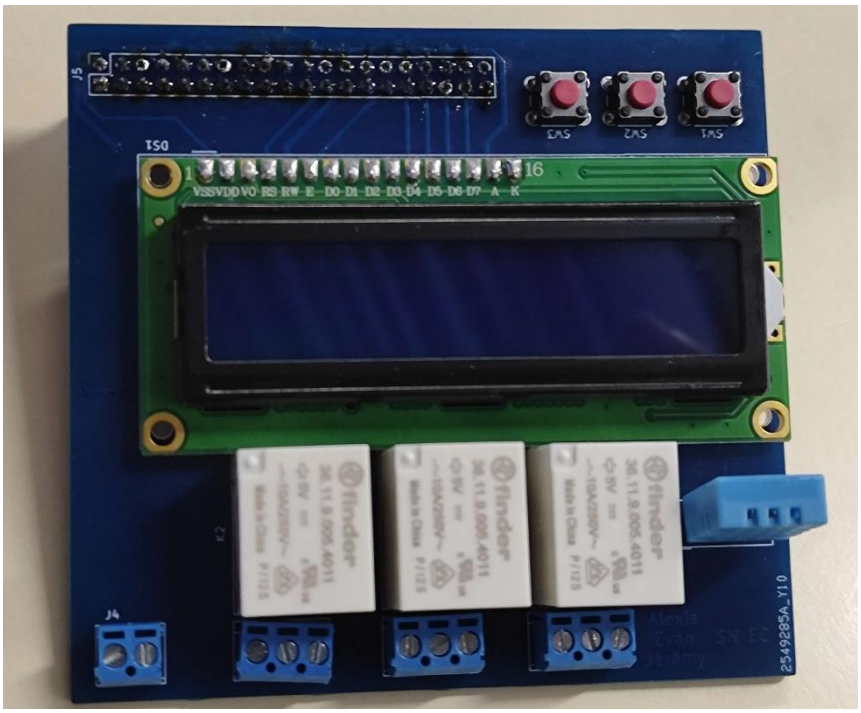
5 REALISATION ELECTRONIQUE

Ci-dessous notre première carte, c'était en quelque sorte la carte d'essai. Nous avons pu corriger nos erreurs de conception grâce à ce premier montage.



Nous avons soudé le strict minimum sur cette carte, car ayant vu les erreurs commises, nous avons décidé de l'utiliser uniquement pour tester nos parties.

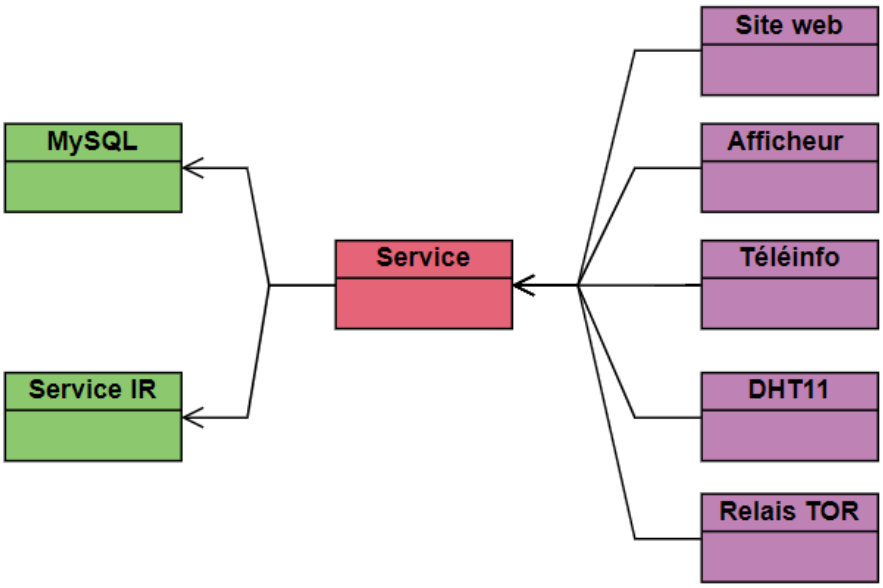
Nous avons ensuite décidé de commander les cartes auprès d'un spécialiste. Les cartes du lycée ne permettaient pas de relier les deux faces à travers les vias, ce qui signifiait que nous devions souder les vias des deux côtés, ce qui posait problème car la plupart des composants traversants nécessitent des pistes sur les deux faces. Il était donc impossible d'obtenir une carte propre si nous devions souder le connecteur 2x20 ou les relais des deux côtés.



Ci-dessus, vous pouvez voir la carte finale. Nous avons procédé au soudage des composants par ordre de taille, en commençant par les plus petits. Bien sûr, nous avons commis quelques erreurs, comme le placement de composants à l'envers ou des soudures défectueuses. Dans de tels cas, nous avons utilisé de la tresse à dessouder pour enlever les composants et les remettre dans le bon sens. Pour les soudures défectueuses qui ne font pas contact, il est souvent plus difficile de les détecter visuellement. Dans ces cas, nous utilisons un multimètre pour effectuer des mesures et vérifier si elles correspondent aux attentes. Si ce n'est pas le cas, cela nous indique qu'il y a un problème et nous pouvons localiser la source de l'erreur.

6 CONCEPTION ET REALISATION SOFTWARE

Ci-dessous, la vue d'ensemble des différents programmes utilisés pour faire fonctionner le système, ainsi que leurs dépendances.



En vert, nous avons les bases de données. En rose, le service de données. En violet, les programmes "bas niveau" qui utilisent les GPIO de la Raspberry PI pour les capteurs et modules.

6.1 LE SERVICE

Le service est un programme qui permet de faire une forte abstraction de la gestion des données. Ici, les données sont celles du télé info et du capteur DHT de la partie de Jérémy.

Le client, qui utilise le service, demande ou envoie des données au service. Ainsi, le client n'a pas à se soucier de la manière dont les données sont stockées ou lues.

De plus, le service permet de manière transparente au client de changer le mode de stockage des données. Concrètement, cela est utile pour utiliser soit une base de données classique comme MySQL, soit le service proposé par nos camarades de l'option informatique ou réseaux.

6.1.1 Protocole

Pour que le service et le client puissent communiquer et échanger des informations, il est nécessaire de définir un protocole, c'est-à-dire un ensemble de règles qui spécifient la manière dont ils doivent communiquer pour se comprendre mutuellement.

La première étape consiste à établir un lien entre les deux programmes. Le lien est un support qui permet la transmission d'informations, de la manière qu'un câble est un support pour une transmission UART.

Pour cela, j'ai choisi d'utiliser le protocole réseau WebSocket, qui est populaire et simple à utiliser. Le protocole WebSocket permet une communication full-duplex. J'ai opté pour ce choix car la plupart des langages de programmation disposent de bibliothèques qui facilitent grandement l'utilisation du protocole WebSocket.

Avant de continuer, il est important de comprendre que le programme appelé « service » contient en fait plusieurs sous-services. Ces sous-services sont utilisés pour différencier les données provenant du télé info et celles provenant du DHT. Cela permet de traiter et de gérer ces données de manière séparée au sein du programme.

Maintenant que le support est défini, nous avons besoin d'un protocole. Notre protocole est basé sur un système de paquets (unités de données), qui sont en quelque sorte des trames. Ces paquets sont formatés en JSON, un format de données couramment utilisé pour l'échange de données entre systèmes.

Le JSON est un format de données qui fonctionne sur la base d'une structure clé/valeur, comme de nombreux autres formats. Les clés sont connues à la fois par le client et le service, elles représentent le nom des valeurs, qui elles peuvent varier.

Tous les paquets, à l'exception du ping, ont en commun deux clés de base :

- Paquet : elle permet d'identifier la fonction du paquet.
- Service : elle permet d'identifier le service ciblé par le paquet.

Voici respectivement les noms, côté (client ou service), et descriptions de tous les paquets du protocole:

ask_data	IN	Demande l'ensemble des données enregistrées par le service désigné.
data_registration		Ajout d'une nouvelle donnée au service désigné. La valeur de temps est définie par le service afin d'éviter toute incohérence.
enable_data_stream		Permet d'activer un flux de transmission en temps réel des nouvelles données enregistrées sur le service désigné.
ping		Paquet pour tester la connectivité avec le service
data_response	OUT	Réponse au paquet ask_data
new_data_registration		Transmet au client le flux de données demandé par le paquet enable_data_stream
Pong		Réponse au paquet ping

(Les paquets IN, représentent les paquets reçus par le serveur, les OUT ceux envoyés par le serveur)

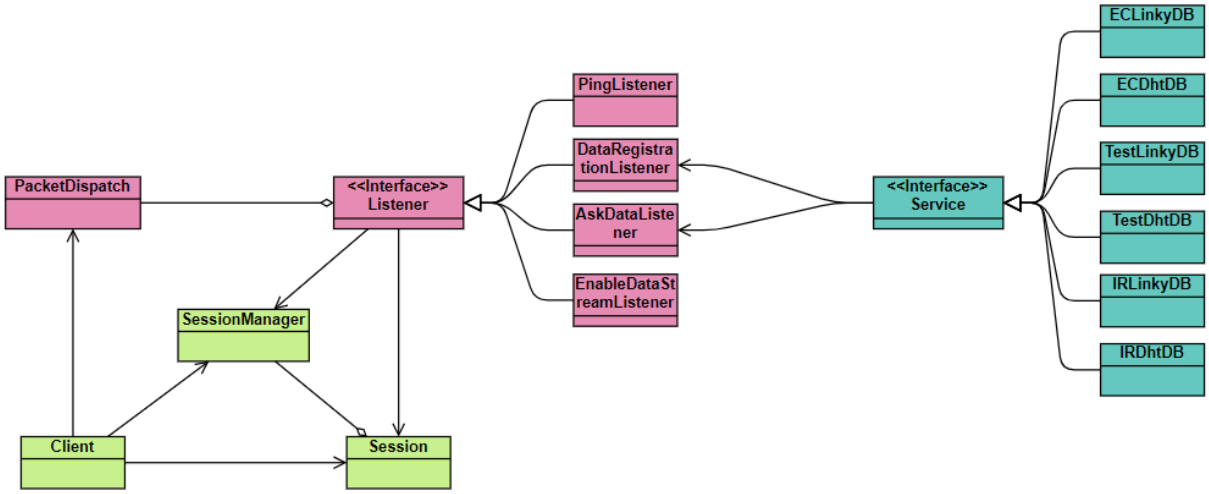
Voici l'exemple d'un paquet enable_data_stream pour le sous-service Linky (le télé info) :

```
{
    "packet": "enable_data_stream",
    "service": "linkyService"
}
```

6.1.2 Programme

Le service est programmé en Java, un langage de programmation orienté objet compilé. J'ai choisi Java car il est explicite et permet de développer des applications complexes tout en restant compréhensible. Je suis déjà familier avec Java, ce qui facilite le développement du service. Bien que Python soit couramment utilisé sur Raspberry Pi, j'ai opté pour Java en raison de ses fonctionnalités avancées et de sa syntaxe claire, qui sont adaptées à ce projet.

Ci-dessous le diagramme de classes simplifié du programme :



Nous avons la gestion des clients WebSocket en vert, le système d'événements qui gère les paquets reçus en rose, et la gestion des données en bleu.

Le code complet est disponible en annexe.

6.2 LA BASE DE DONNEES

La base de données nous permet de sauvegarder et de lire les données. Nous utilisons une base de données relationnelle, et avons choisi MySQL.

Pour que le service puisse utiliser MySQL, nous devons définir trois fichiers pour chaque sous-service :

- Un fichier pour la création des tables
- Un fichier pour la lecture des données
- Un fichier pour l'écriture d'une donnée

Voici par exemple le fichier SQL qui permet l'écriture d'une donnée pour le service du télé info :

```
USE ${database};

INSERT INTO Linky VALUES (
    ${time},
    ${id},
    ${subscribe_instensity},
    ${base_hour_index},
    ${full_hour_index},
    ${offpeak_hour_index},
    ${max_instensity},
    ${instant_intensity},
    ${current_tariff_option}
);
```

On peut voir des balises du style ${xxx}, qui servent à indiquer l'emplacement d'une donnée. Le service remplace simplement ces balises par une valeur correspondante.

Les deux autres fichiers SQL sont disponibles en annexe.

6.3 LA RECUPERATION DES DONNEES TELE INFO

Les données envoyées par le télé info sont converties en signaux UART et reçues par le port GPIO 15. Le système Linux permet la communication avec des périphériques externes grâce à des fichiers de périphériques. Ces fichiers servent d'interface pour les pilotes.

Dans notre cas, le fichier servant à lire les données du GPIO 15 en mode UART est le fichier /dev/ttyS0.

Je fais la lecture avec un programme Python. J'ai choisi Python car le code est assez court et simple, ce qui rendra le développement plus rapide et plus facile.

```
import websocket
import serial
import json
import time

[…]

datas = dict()

isSync = False
mapping = dict()

#Boucle infini
while True:
    #Récuperation étiquette et valeur
    line = ser.readline()
    key, value = parse(str(line))

    #Synchronisation avec le ADCO
    if not isSync and key == "ADCO":
        isSync = True
    #Si déjà synchro on envoi les données de la dernière trame
    #pour commencer la nouvelle
    elif key == "ADCO":
        packet = {"packet": "data_registration", "service": "linkyService",
"data": mapping}
        toSend = json.dumps(packet)
        print(toSend)
        ws.send(toSend)

    if not isSync:
        continue

    #Si l'étiquette nous intéresse on la fait dans un dictionnaire (map)
    if key in keys:
        if types[key] == "str":
            mapping[names[key]] = value
        elif types[key] == "int":
            try:
                mapping[names[key]] = int(value)
            except ValueError as ve:
                continue
```

J'utilise trois librairies :

- Serial, pour l'utilisation du port série
- Websocket, qui permet la gestion du client
- Json, pour formater des données en Json

Le programme est assez simple et ne comporte aucune vérification de l'intégrité des données. Par manque de temps, je n'ai pas implémenté la vérification du checksum. Dans notre situation, cela n'est pas très important car la quantité de données collectées en une journée est assez énorme et les erreurs

commises sont négligeables. Cependant, cela peut poser problème si la ligne est brouillée et que les informations transmises sont totalement incohérentes.

6.4 LE SITE WEB

Le site web a été réalisé en collaboration avec mon camarade Jérémy. Il nous permet d'afficher les données de consommation du télé info, ainsi que la température et l'humidité.

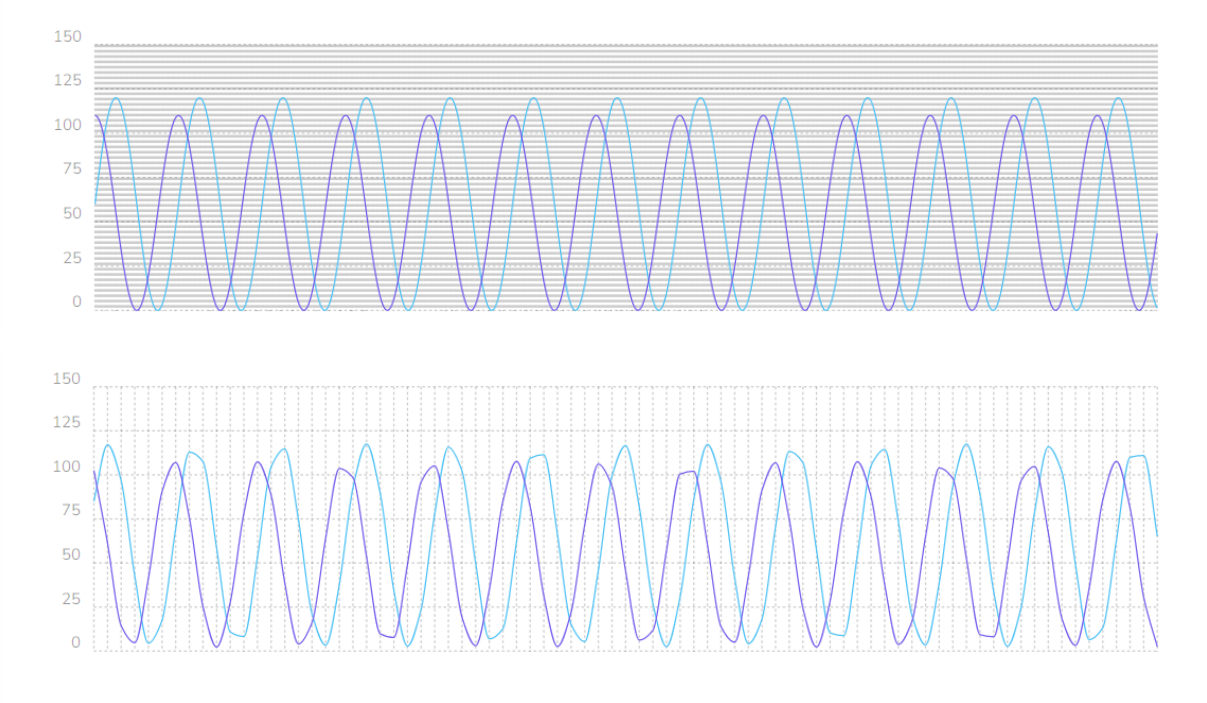
6.4.1 Backend (Server web)

Le serveur web a été développé en Java en utilisant la librairie standard HttpServer de Sun. Étant donné que nous n'avions pas besoin d'un framework complexe ni d'un système de gestion de serveur étendu, cette approche a été suffisante pour nos besoins.

Le programme établit également une connexion avec le service afin de recevoir les données provenant de la partie de Jérémy et de moi-même.

En raison du grand nombre de données possibles, il n'est pas possible de les transmettre directement sur une page web. Il pourrait y avoir plusieurs centaines de millions de données, et la génération de graphiques serait trop longue, voire impossible. C'est pourquoi le serveur stocke en cache les données compressées du télé info et du DHT (partie de Jérémy).

La compression réalisée est assez simple, elle consiste à effectuer une moyenne de plusieurs valeurs. Voici un exemple d'une sinusoïde classique et compressée représentées sur un graphique :



Cela ressemble un peu à de l'échantillonnage. On peut voir une baisse de qualité du graphique quand il est compressé mais cela reste acceptable compte tenu du volume élevé de données à gérer.

6.4.2 Frontend (Page web)

Nous avons utilisé un template pour la conception de base de la page web. Cela nous a permis de gagner du temps.

Il y a trois pages principales :

- Dashboard du télé info
- Dashboard de la temperature et humidité
- Dashboard des deux premiers confondu

Ici, un Dashboard désigne une page permettant la visualisation des données à travers des graphiques et d'autres éléments.

Pour que les données instantanées de nos deux parties se mettent à jour, une connexion avec un serveur WebSocket est établie. Le serveur WebSocket est intégré dans le serveur web.

Il y a donc un programme JavaScript intégré dans la page web qui permet de recevoir les nouvelles données par WebSocket et de mettre à jour le graphique correspondant.

6.5 GESTION DES PROGRAMMES

Nous avons tous les trois fait des programmes différents pour faire fonctionner nos parties. Il y a 7 programmes différents, et ils doivent fonctionner en même temps.

Quand on se connecte à la Raspberry c'est par SSH. De cette façon on ne peut exécuter qu'un seul programme à la fois et il faut maintenir la connexion SSH ouverte en permanence. Cela signifie qu'il serait nécessaire d'utiliser sept ordinateurs différents pour exécuter simultanément les sept programmes nécessaires.

Pour résoudre ce problème, nous utilisons Screen. Screen est un programme distribué par le projet GNU et disponible sur certaines distributions de type UNIX. Il permet de créer des terminaux virtuels qui restent ouverts même lorsque la session SSH est inactive.

Donc, pour chacun de nos programmes, nous créons un nouveau terminal virtuel dans lequel nous l'exécutons. Voici un exemple :

```
screen -dmS teleinfo sudo python3 programme_teleinfo.py
```

Pour nous simplifier la tâche, nous utilisons un script en Bash qui nous permet d'exécuter tous les programmes dans une session Screen :

```
screen -dmS service java -jar ./Service/service.jar
screen -dmS web java -jar ./site/webserver.jar
screen -dmS teleinfo sudo python3 programme_teleinfo.py
screen -dmS afficheur sudo python3 programme_afficheur.py
screen -dmS dht sudo python3 programme_dht.py
screen -dmS bouttons sudo python3 programme_bouttons.py
```

7 CONCLUSION

Ce projet a été une opportunité d'apprentissage pour chacun d'entre nous. Nous avons mis en œuvre nos compétences et en avons acquis de nouvelles pour la conception et la réalisation de ce projet.

Malgré son apparence simple, ce projet cache en réalité beaucoup de complexité dans différents domaines. Son titre, "Gestion de la consommation électrique", nous a conduit à constater l'importance de l'énergie électrique dans le monde, son caractère indispensable, ainsi que sa dépendance vis-à-vis des multiples autres sources d'énergie, ce qui rend sa gestion et son coût complexes. Nous avons ensuite remarqué que les nouveaux compteurs électriques en France mettent à disposition des clients

un moyen de récupérer ces informations de consommation. Cependant, cette récupération n'est pas si simple. Nous avons donc dû concevoir un montage électronique permettant la conversion et la réception de ces informations. Une fois ces informations reçues, nous devons maintenant les interpréter et les présenter à l'utilisateur.

Ce projet est passionnant, car il offre de nombreuses possibilités d'amélioration, et nous n'avons exploré qu'une infime partie des fonctionnalités d'un tel système. C'est le genre de projet qui ne peut jamais vraiment être considéré comme terminé.

Je vous remercie d'avoir lu mon rapport. J'espère vous avoir intéressé pour ce projet et les nombreux domaines qu'il englobe.

8 REFERENCES

1. **Olivier.** Suivi conso de votre compteur électrique. *magdiblog.* [En ligne] https://www.magdiblog.fr/gpio/teleinfo-edf-suivi-conso-de-votre-compteur-electrique/.

2. **hd31.** Adaptateur téléinfo. *Chaleur Terre.* [En ligne] http://www.chaleurterre.com/forum/viewtopic.php?t=13232.

3. **Vishay Semiconductors.** sfh620a. *Vishay.* [En ligne] https://www.vishay.com/docs/83675/sfh620a.pdf.

4. **NEERDEN, TIMO VAN.** C'est quoi le « cos(phi) » et la puissance réactive ? *Couleur Science.* [En ligne] https://couleur-science.eu/?d=0f397f--pourquoi-edf-compte-la-puissance-en-kva-et-non-en-kw-cest-quoi-le-cosphi-et-la-puissance-reactive.

5. **Mosaic Industries.** GPIO Electrical Specifications. *Mosaic Industries.* [En ligne] http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications.