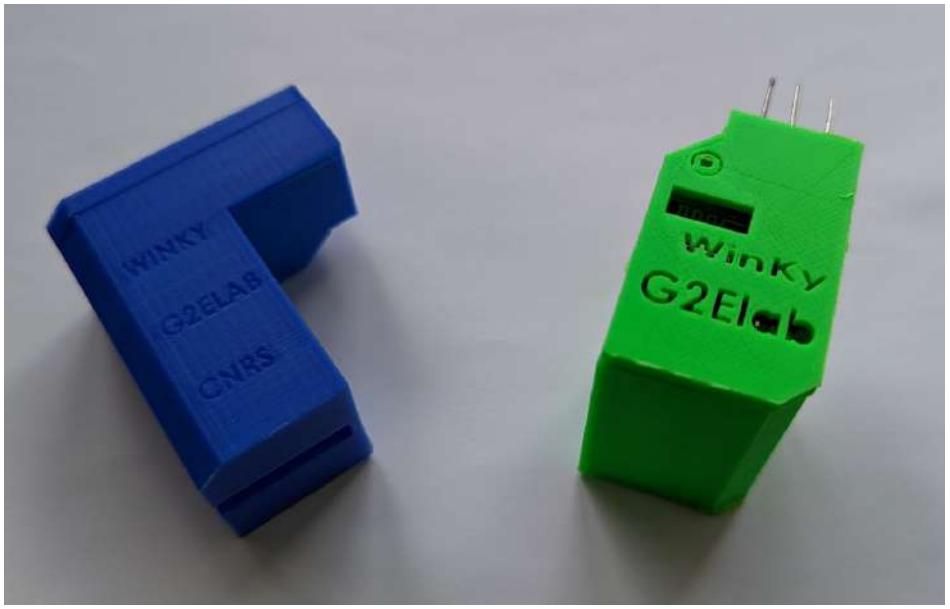
Projet d'ingénierie Grenoble INP - ENSE3
Février - Juin 2022

P003 - SmartRoom

Amélioration des habitudes de consommation d'énergie dans le secteur résidentiel grâce à la domotique



Coordinatrice: Céline BOURGEOIS

Porteurs de projet : Jérôme FERRARI, Benoit DELINCHANT, Frédéric WURTZ

Membres de l'équipe: Antonin ANDRE, Clara JENFT, Emma HOAREAU, Marie HALLING, Marine PHALIPPON, Guillaume DAUENDORFFER, Yukang WANG

Abstract

Energy is used everywhere in our lives, especially in the housing sector. Energy production, including electricity, is an important source of Carbon-dioxide (CO_2) emission, which is why reducing its consumption is essential. Autonomous houses are a good solution to participate in the energy transition. The objective of the SmartRoom project was to create a smart and reproducible system that recommends changes in the habits of the user's energy consumption. Previous work in this subject has come up with sensors to measure the electricity use in real time (Winky). The project has developed a system that uses real time production and consumption data of electricity from RTE combined with logged data from the sensors. With this information, we want to inform the user about his consumption (average value, evolution and CO_2 equivalent). The system also predicts the best time of day to consume according to the state of the network and the renewable energy ratio. These information are communicated to the user by SMS so that the alerts are received in real time. What has been achieved through this project is a better understanding of energy consumption that could lead to a change in people's behavior and better energy management.Nevertheless, the data transmission, due to problems with the Winky sensors, makes it a requirement to continue the study to achieve reproducible content, as was our objective.

Résumé

L'énergie est utilisée partout dans notre vie, en particulier dans le secteur du logement. La production d'énergie, dont l'électricité, est une source importante d'émission de dioxyde de carbone (CO_2), c'est pourquoi il est essentiel de réduire la consommation. La domotique est une bonne solution pour participer à la transition énergétique. L'objectif du projet SmartRoom était de créer un système intelligent et reproductible par tous, qui recommande des changements dans les habitudes de consommation d'énergie de l'utilisateur. Les travaux précédents sur ce sujet ont abouti à des capteurs permettant de mesurer la consommation d'électricité en temps réel (les Winky). Au cours du projet a été développé un système qui utilise les données de production et la consommation d'électricité en temps réel de RTE combiné avec les données enregistrées par les Winky. Avec ces informations, nous informons l'utilisateur sur sa consommation (valeur moyenne, évolution et équivalent CO_2). Nous pouvons également prévoir le meilleur moment de la journée pour consommer en fonction de l'état du réseau et de la part d'énergies renouvelables. Ces informations sont communiquées à l'utilisateur par SMS pour une réception de l'alerte en temps réel. Les résultats de ce projet sont une meilleure compréhension de la consommation d'énergie qui pourrait conduire à un changement de comportement des personnes et une meilleure gestion de l'énergie. Néanmoins, l'acquisition de données ne fonctionnant pas très bien, des travaux peuvent être réalisés sur celle-ci.

Sommaire

Table des figures

Partie 1: Aspects scientifiques et techniques

1. Introduction

1.1 Contexte

De nos jours, les effets du changement climatique se font déjà ressentir, il est donc essentiel d'agir, notamment dans le secteur des énergies. Dans le monde, 41% des émissions de CO2 sont dues à la production d'énergie électrique [1]. L'électricité est omniprésente dans nos vies: transports, systèmes industriels, habitations… De plus, la consommation d'énergie dans le monde est en constante augmentation. Dans ce contexte, la domotique (ou les maisons 'intelligentes') est un moyen de concilier efficacité et optimisation énergétique. La domotique correspond à l'ensemble des techniques d'automatisation en matière d'habitat. C'est dans ce contexte que les chercheurs du Laboratoire de Génie Electrique de Grenoble (G2Elab), ont mis en place une plateforme de démonstration avec de nombreux capteurs: le Living Lab. Cette plateforme ne se focalise pas seulement sur la consommation d'électricité mais aussi sur la commande vocale, le comptage du nombre de personnes… Elle s'inscrit dans le projet du futur *Observatoire pour la Transition Énergétique* afin de pouvoir capitaliser les connaissances autour des stratégies de pilotage de production et de consommation dans le bâtiment. Ce cadre est aussi inscrit dans une démarche Open Source et Open Science, c'est-à-dire que les méthodes et résultats sont documentés et publics.

1.2 But général du projet

Notre projet est très étendu: contribuer à la transition énergétique dans le secteur résidentiel grâce à la domotique. Afin de cadrer notre projet, nous avons décidé de nous focaliser sur l'aspect électrique de l'énergie. En effet, c'est le vecteur d'énergie qui est le plus démocratisé dans le secteur résidentiel et également le plus facilement pilotable. Les solutions trouvées doivent être utilisables par le plus grand nombre et s'inscrivent dans un cadre Open Source, Open Science.

1.3 Objectifs du projet

Les objectifs que nous nous sommes fixés sont les suivants:

- Mieux informer l'utilisateur sur sa consommation et ses évolutions afin qu'il en prenne conscience.
- Inciter l'utilisateur à consommer au meilleur moment, c'est-à-dire, en dehors des pics de consommation et lorsque les énergies renouvelables sont importantes. Il sera donc nécessaire dans un premier temps de prédire la consommation et la production renouvelable (à l'échelle française).
- Aider au développement du capteur Open Science permettant l'acquisition des données de consommation en temps réel sur le compteur Linky.

2. Description de l'installation

Pour parvenir à nos objectifs, nous devons récupérer les données électriques. Pour ce faire, nous avons utilisé l'installation et le matériel qui sont présentés ci-dessous.

2.1 Installation

Comme mentionné dans nos objectifs, nous souhaitons utiliser un capteur nommé Winky (voir partie suivante). Ce capteur permet d'acquérir des données de consommation électrique. Ces données seront finalement envoyées par Wifi dans une application téléchargée dans une raspberryPi. Dans cette même Raspberry, une base de données sera implantée afin de stocker ces données. Enfin, ces données seront traitées et analysées grâce à des codes Python qui seront également téléversés dans la raspberry. Comme nous l'expliquerons tout au long du rapport, le traitement des données implique, dans notre cas, le traitement de données extérieures fournies par RTE (Réseau de Transport de l'Electricité). Enfin, le résultat du traitement des données sera envoyé à l'utilisateur via une Interface Homme Machine (IHM).

Le compteur électrique, le Winky et la raspberry doivent toujours être connectés à un Wifi commun et se trouver à proximité. L'utilisateur au contraire pourra recevoir des informations même s'il est loin de la maison hôte.

Un schéma récapitulatif est présenté dans la Figure 1.

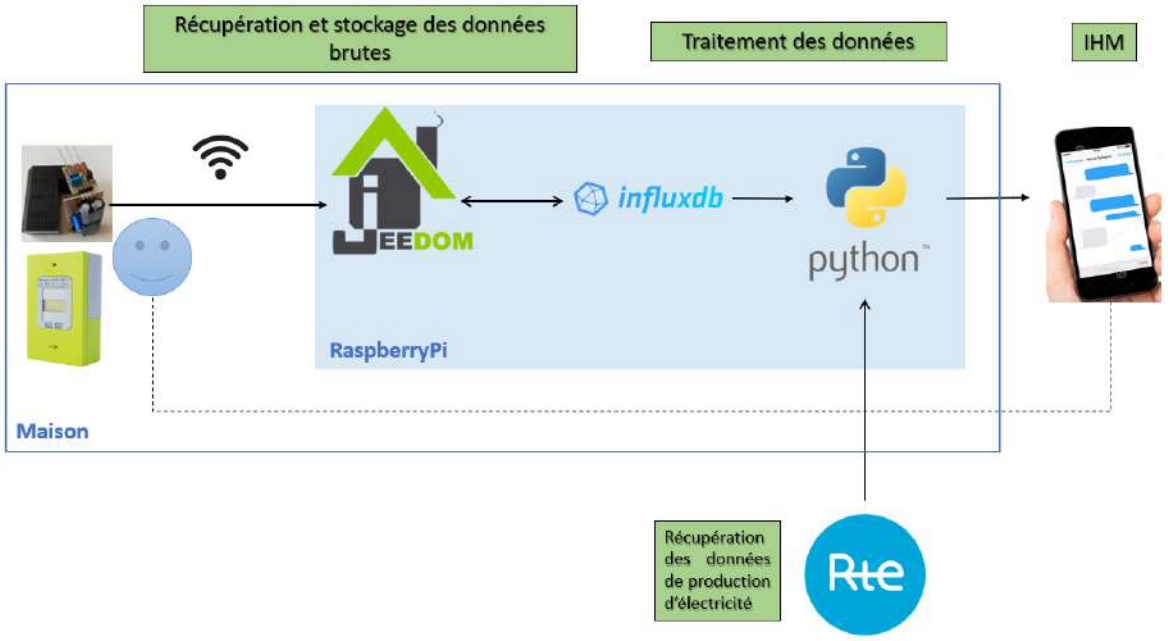


Figure 1: Schéma global d'une installation

Un tutoriel, présent en **Annexe 1**, permet de mettre en place cette installation.

2.2 Matériel

Winky

Les compteurs électriques intelligents comme les Linky permettent d'accéder aux données électriques d'un foyer à J+1 avec un pas de temps de 30min. Cela ne permet pas de surveiller la consommation en temps réel, on ne peut donc pas agir sur celle-ci à l'instant 't'. C'est dans ce contexte que les Winky prennent tout leur sens (photo d'un Winky en Figure2).
Les Winky sont des capteurs "fait maison" entièrement accessibles en Open Source et réalisables en FabLab. Ils se branchent via la prise TIC qui sert de passerelle de communication sur les compteurs Linky. Ils ont été mis au point par Jérôme Ferrari, porteur du projet, dans le cadre de ses recherches pour le G2Elab.
L'objectif de ces capteurs est d'acquérir les données en temps réel avec un pas de temps faible de l'ordre de quelques dizaines de secondes.

Figure 2 : Compteur Linky avec capteur Winky branché sur sa borne TIC

Comme mentionné précédemment, les Winky et leurs installations ont pour objectif d'être reproductibles par le plus grand nombre, moyennant quelques connaissances en électronique et informatique. Tous les membres du groupe avaient pour objectif d'installer ces capteur chez eux afin de vérifier la reproductibilité du capteur, en testant eux même la fabrication et l'installation complète. Ce qui comprenait le perçage des cartes et le soudage des composants sur cette dernière (l'impression étant faite au préalable), l'installation de Jeedom, le paramétrage de la Raspberry Pi, du micro-controlleur…. Nous avons ainsi fourni notre retour d'expérience sur ce processus à notre porteur de projet afin qu'il puisse mieux se rendre compte du degré de difficulté de la manipulation, des potentiels problèmes qui peuvent être rencontrés et des choses à améliorer.

Raspberry Pi

Une Raspberry Pi est un nano-ordinateur que nous utilisons sous Raspbian, système d'exploitation libre et gratuit basé sur Linux. Il est possible d'y installer des applications pour récupérer, traiter et transmettre des données. Il est aussi possible d'y rentrer des codes Pythons ainsi que des services qui vont lancer ces scripts à des intervalles prédéfinis. C'est sur une Raspberry Pi que nous allons effectuer toute la démarche, de la collecte des données à la transmission à l'utilisateur en passant par le traitement de ces données. Il est nécessaire d'avoir une Raspberry par installation.

3. Extraction des données

Comme mentionné précédemment, nous souhaitons collecter les données électriques des foyers via les capteurs Winky. Nous associerons cela aux données de production et consommation du Réseau de Transport d'Électricité (RTE). Pour pouvoir traiter ces données grâce à des scriptes Python, il faut premièrement les extraire.

3.1 Extraction des données fournies par le Winky

Explications du principe

Les données fournies par le capteur sont envoyées par Wifi sur Jeedom puis stockées dans la base de données InfluxDB (voir figure 1). Cette base de données contient plusieurs séries de données comme la puissance et l'énergie qui sont associées à une clé unique (appelée 'index' par la suite). Afin d'exploiter les données, nous devons extraire seulement la série de données sur laquelle nous portons de l'intérêt puis la convertir en fichier CSV grâce à un code Python. Les fichiers CSV ont un format texte fournissant des données tabulaires et sont facilement utilisables avec Python. Chaque fichier CSV contient une série de valeurs associée pour une durée donnée. Ces fichiers CSV contiennent des valeurs qui sont chacunes associées à un instant. Ce code doit être suffisamment performant pour extraire des milliers de données, nous devrons donc l'optimiser au maximum.

Remarque: pour utiliser ce code, il est nécessaire d'être connecté à la wifi liée à la raspberryPi de l'installation (celle-ci contient la base de données).

Description du code

Il faut dans un premier temps saisir toutes les informations relatives à la base de données comme le nom et le numéro de serveur. Puis il faut se connecter à la base de données. Une fois cela fait, un fichier CSV est créé avec uniquement la série d'intérêt (repérée par un index) grâce à une fonction: '*csv_creation(end, delta_time, name, index)*'. Une boucle 'for' est utilisée pour créer le CSV petit à petit, et non d'une traite, afin de ne pas faire mettre le code en erreur.

Les variables utilisées dans la fonction sont les suivantes:

- end: fin de la période, c'est à dire la date la plus récente
- delta_time: période de récupération de données
- name: nom du fichier CSV qui va être créé
- index: celui de la série de donnée qu'on souhaite extraire

Ainsi, de manière vulgarisée, pour obtenir les données de consommation sur une semaine à partir d'aujourd'hui: *csv_creation(now, 7jours, consommation_actuelle, 1234)*
De même mais pour la période précédente: *csv_creation([now-7j], 7jours, consommation_precedente, 1234)*

Remarque: ce code nous a été en partie fourni par notre porteur de projet, nous l'avons adapté pour qu'il puisse répondre à des requêtes longues et transformé en fonction pour faciliter son utilisation.

Résultats

La figure 3 montre comment les données obtenues sont présentées:

1	time,value
2	2022-05-04T12:58:11Z,11327970.0
3	2022-05-04T12:58:16Z,11327971.0
4	2022-05-04T12:58:16.099380Z,11328328.0
5	2022-05-04T12:58:21Z,11327972.0
6	2022-05-04T12:58:23.873367Z,11328329.0
7	2022-05-04T12:58:26Z,11327973.0
8	2022-05-04T12:58:32Z,11327974.0
9	2022-05-04T12:58:34.385354Z,11328330.0
10	2022-05-04T12:58:37Z,11327975.0
11	2022-05-04T12:58:39Z,11327976.0

Figure 3: Fichier CSV pour des données de consommation (Wh).

Il est nécessaire de traiter ultérieurement les données: mise en forme de la date, suppression des caractères… afin de pouvoir les exploiter.

3.2 Extraction des données RTE

RTE fournit des données et met à disposition un outil simple: éCO2mix [2]. Cet outil est disponible sur internet et permet de télécharger un fichier avec toutes les données de production/consommation française en temps réel sur un mois complet avec un pas de temps de 15 minutes sous la forme d'un fichier XLS. Grâce à un code python, nous récupérons automatiquement ce fichier et le transformons en un format plus facilement exploitable et compatible avec Linux, afin de pouvoir téléverser tous nos fichiers dans une Raspberry Pi.
Grâce à ces données sur la production et la consommation, nous pouvons alerter le consommateur sur le meilleur moment à consommer ainsi que lui fournir plus d'information sur son empreinte carbone.

Méthode

Expliquons rapidement la genèse du code et la forme sous laquelle se présentent les données de RTE. Le script est construit avec plusieurs fonctions, de manière à ce qu'il suffise de les appeler successivement, en fin de script, pour effectuer la récupération de données souhaitée.

Les différentes données publiques de RTE (production par filière, consommation, export…) sont téléchargeables à l'adresse suivante :
«https://www.rte-france.com/eco2mix/la-consommation-delectricite-en-france»
Si l'on veut effectuer l'action manuellement, il faudra cliquer sur les mentions *télécharger nos données* et *En-cours mensuel temps réel* présentent sur le site.

Les fonctions *initialisation* et *télécharger* permettent d'effectuer automatiquement ces actions. La fonction *initialisation* prépare le chemin d'accès suivant lequel nous allons stocker les données dans l'ordinateur, et elle supprime le dossier précédent, s'il existe, afin de pouvoir re télécharger le fichier sous le même nom. La fonction *télécharger* se charge de suivre l'URL de téléchargement des données.

Un fois cette action effectuée, un dossier zippé se charge sur la machine ayant envoyé la requête. La fonction *dezip* nous permet simplement, en utilisant le module python *zipfile*, de deziper le fichier pour que celui-ci soit lisible par l'ordinateur.
Le fichier ainsi obtenu contient les valeurs regroupées dans un grand tableau Excel. Ce tableau comprend environ 40 colonnes (une par série de données) et 7000 lignes (mais ce chiffre varie constamment). Les données sont réactualisées chaque 15 minutes et sont celles des 30 derniers jours.

Un traitement est ici nécessaire (réalisé par la fonction « recup_xls ») car le tableur est au format XLS, ce qui correspond à l'ancien format de Excel, non lisible par l'ordinateur. Nous avons donc décidé d'opter pour une ouverture du fichier sous forme binaire (exploitable par la machine contrairement au format XLS), puis de le transformer en chaîne de caractères et enfin d'en extraire les valeurs une à une pour les placer dans une liste. Plus précisément, cette liste est une « liste de liste » dont chaque « sous liste » correspond à une ligne du tableau Excel.

La fonction *recup_valeur_conso* permet de récupérer la valeur de la dernière consommation nationale. Celle-ci est la dernière valeur non nulle de la colonne. Ainsi, pour la récupérer, il suffit de récupérer l'indice du premier 0 et de prendre le précédent. Si l'on veut récupérer une autre valeur, par exemple la dernière valeur de production de la filière éolienne, il suffit simplement de changer l'indice correspondant à la colonne dans laquelle sont rangées les données souhaitées. Pour connaître l'indice de la colonne dont on veut extraire la dernière donnée, il suffit de télécharger manuellement le fichier et de l'ouvrir. La ligne dans laquelle le changement d'indice est à effectuer est indiquée par un commentaire.

La dernière fonction, *recup_conso*, fonctionne exactement sur le même principe, elle permet simplement de récupérer toutes les valeurs de la colonne « prévisions de consommation » sur les 24 dernières heures. La fonction *recup_CO2* utilisée dans le code pour calculer l'équivalent CO_2 fonctionne également sur le même principe.

Pour plus de détails, voir **Annexe 2**.

Résultats

Des fonctions d'affichage nous permettent de constater que la récupération des données s'est bien faite et que celles-ci sont effectivement dans un format traitable par la machine (Figure4).

```
In [1]: runfile('/Users/clarajenft/Desktop/Bureau/projet_smart_house/
recup_valeur.py', wdir='/Users/clarajenft/Desktop/Bureau/projet_smart_house')


Dernière conso à 22:45 le 2022-05-12: 47120 MW
en attente
```

Figure 4: affichage de la dernière valeur de consommation.

Une fois les données de consommation du foyer et les données de production française extraites, nous devons les traiter puis transmettre le résultat à l'utilisateur.

4. Transmettre les informations à l'utilisateur

Notre objectif étant d'agir sur les habitudes de l'utilisateur et de lui faire prendre conscience de ce que représente sa consommation, nous allons utiliser une interface homme machine [IHM] dans ce but. Une IHM permet de mettre en relation les humains et les données informatiques recueillies par les capteurs. Dans notre cas, nous devions trouver un moyen de prévenir en temps réel l'utilisateur lorsqu'une anomalie dans sa consommation est détectée ou pour l'informer du meilleur moment pour consommer dans la journée. Un rapport mensuel ou hebdomadaire semble un bon moyen de mieux informer les habitants sur leur consommation de manière générale (moyenne, équivalent CO_2…). Pour finir, il est également nécessaire d'envoyer des messages à l'utilisateur pour l'informer du meilleur moment de consommation.
Afin de pouvoir communiquer à l'utilisateur les informations importantes, nous sommes passés par plusieurs solutions.

La première est l'envoi de SMS grâce à l'application Pushbullet, disponible sur le système d'exploitation Android. En téléchargeant cette application sur un téléphone, il est possible de s'y connecter avec un compte Google. Grâce à cette identification, on obtient une clé d'accès et un identifiant de l'appareil qui permettront d'effectuer des requêtes directement sur le téléphone lié qui servira d'émetteur pour les SMS. Ces requêtes peuvent se faire par l'intermédiaire d'un script Python. C'est le code *envoi.py* contenant la fonction *envoi_sms* qui permet de réaliser cela. Il suffit seulement d'indiquer en entrée de la fonction lors de son appel, sous la forme d'une chaîne de caractères, le message que l'on veut envoyer. L'avantage de cette méthode d'alerte est qu'il est possible d'envoyer des SMS à n'importe quel téléphone qui est relié au réseau téléphonique, il suffit d'avoir son numéro. Un autre intérêt des SMS est le côté instantané de l'alerte. L'inconvénient est que le nombre d'envoi de SMS est limité à 100 par mois par compte Google pour la version gratuite.

La deuxième méthode est substituable à la première, c'est l'envoi de notification grâce à l'application Pushbullet. De la même manière, il faut s'identifier sur l'application avec un compte Google et récupérer la clé d'accès ainsi que l'identifiant du téléphone. La fonction est aussi contenue dans le code *envoi.py* et s'appelle *envoi_notification*. Elle prend en entrée le titre et le message de la notification à envoyer sous la forme de chaînes de caractères. Les avantages de cette méthode sont que le nombre de notifications est illimité et qu'il n'y a ni besoin de carte SIM ni besoin d'être relié au réseau téléphonique. L'inconvénient de cette méthode est que l'application n'est disponible que pour le système d'exploitation mobile Android et donc que les téléphones sous un autre système d'exploitation (par exemple IOS) n'ont pas accès à cette méthode. Nous avons donc choisi, dans notre cas, puisque l'objectif est que notre solution soit accessible au plus grand nombre, d'utiliser l'envoi de SMS. Ce choix a aussi été basé sur l'hypothèse qu'il y aura moins de 100 alertes par mois et donc que la limite ne sera jamais atteinte.

La troisième méthode est l'envoi de mail à l'aide d'un script Python. Pour ce faire, nous avons créé une nouvelle adresse mail qui servira à cet effet. Ce script correspond à la fonction *envoi_mail* du code *envoi.py*, elle prend en entrée l'objet ainsi que le message du mail, tous deux sous la forme d'une chaîne de caractères. L'une des utilités de cette méthode est de pouvoir envoyer une plus grande quantité d'informations qui sont moins pénibles à lire par mail que par SMS. Par exemple, pour envoyer un rapport mensuel complet sur la consommation du mois passé et son évolution par rapport aux mois précédents ainsi que des commentaires et analyses sur différents aspects de cette consommation.

La quatrième méthode est l'affichage des données recueillies sur grafana qui est une application qui permet d'afficher des courbes à partir d'une base de données. Cet affichage permettrait à l'utilisateur d'avoir accès à toutes ses données en permanence et de pouvoir choisir à laquelle il veut s'intéresser. Malheureusement, nous avons été confrontés à un problème pour la mise en place de cette méthode. La transmission et l'affichage des données sur Grafana depuis la base de données InfluxDB se passe sans souci mais c'est pour l'importation des données sur InfluxDB que le problème apparaît. En effet, afin de pouvoir faire notre traitement de données, nous téléchargeons les données de consommation sous forme de fichier csv puis les données traitées sont à nouveau transcrites sur un fichier csv. Cependant, avec les versions des différentes applications qui sont utilisées lors du processus, pour importer les données d'un fichier csv sur InfluxDB, il faut utiliser une application supplémentaire, Telegraf. Mais il s'avère qu'un problème de reconnaissance du format de temps utilisé bloque l'importation des données et, en partie dû au retard pris auparavant, nous n'avons pas pu résoudre ce problème. Une solution aurait été de le changer lors de la récupération du fichier csv, avant le traitement des données, mais les fonctions utilisées par InfluxDB pour la création du fichier csv ne l'acceptent pas.

Ainsi, notre IHM repose principalement sur l'utilisation de l'application PushBullet. C'est une application disponible sous le système d'exploitation Android et téléchargeable également sur un ordinateur. Cette application permet d'effectuer différents types de requêtes telles que l'association de plusieurs appareils à un compte, la création de conversation entre différents appareils ou l'envoi de texte éphémère qui se suppriment une heure après avoir été envoyés. Dans notre cas, comme expliqué précédemment, nous nous sommes concentrés sur les requêtes d'envoi de SMS. Ces requêtes sont effectuées par l'interface de commande en ligne *curl* et sous le format *json*. C'est pourquoi, pour avoir la possibilité de les intégrer à notre chaîne d'informations, il a fallu convertir les requêtes en langage Python.

Un tutoriel portant sur le code *envoi* est disponible en **Annexe 3.**
De même une description de l'installation et l'utilisation de l'application est disponible en **Annexe 4**.

L'IHM que nous avons créée est entièrement numérique, nécessite peu de matériel et est facile d'utilisation ce qui la rend récupérable et réutilisable par le plus grand nombre.
Nous venons de voir comment informer l'utilisateur. Nous allons maintenant vous présenter les diverses informations que nous souhaitons lui transmettre ainsi que la manière dont elles ont été obtenues.

5. Informer l'utilisateur sur sa consommation

5.1 Valeurs moyennes et évolution

Contexte

Tous les foyers consomment de l'électricité mais généralement, la quantité exacte n'est pas connue (même si certains fournisseurs d'électricité ont mis en place des applications pour cela). Néanmoins, il paraît crucial de prendre conscience de sa consommation afin de savoir si elle pourrait être réduite. C'est pourquoi, grâce à des SMS, communiquer à l'utilisateur sa consommation en énergie (kWh) et sa puissance moyenne (kW) hebdomadaire est importante. Enfin, en comparant grossièrement deux périodes, l'utilisateur est informé sur l'évolution générale de sa consommation, ce qui lui permettrait de faire des efforts pour la réduire.

Méthode

Grâce à la fonction *csv_creation* du code *data_recovery_and_cvs_creation*, nous sommes capable de récupérer les données de consommation pour une durée donnée (pour nous elle est de 7 jours). Dans la base de données, la consommation ne se remet jamais à zéro, nous devons donc effectuer une soustraction entre la dernière valeur (la plus récente) et la première (la plus ancienne) de la période.
En connaissant le tarif réglementaire d'un kWh électrique en France (environ 0,1740 €/kWh), on peut estimer le coût de la consommation sur la période (voir figure 5).

```
############################## Actual energy consumption Wh ########################

#creation of the csv file using the index corresponding to the consumption: 1502
#we want value from 'now' to now-'delta_time'
Data_recovery_and_CSV_creation.csv_creation(now, delta_time, "Actual_Energy.csv", 1502)
df_actual_energy = pd.read_csv('Actual_Energy.csv')

delta_actual_energy = (df_actual_energy.iloc[-1,1]- df_actual_energy.iloc[0,1])/1000 #kWh

price_kWh = 0.1740
tot_price = price_kWh*delta_actual_energy
```

Figure 5 : Script du calcul de l'énergie consommée sur la période actuelle

Il est possible de réaliser la même chose pour la période précédente en modifiant l'appel de la fonction *csv_creation* (voir figure 6). La début de la période précédente correspond à la fin de la période actuelle.

```
Data_recovery_and_CSV_creation.csv_creation(end_previous_period, delta_time, "Past_Energy.csv", 1502)
```

Figure 6: Création du fichier csv pour la période précédente

Pour finir, en appelant la fonction *tendency (figure 7)* qui compare deux valeurs, il est possible de savoir comment la consommation a évolué entre les deux périodes.

```
def tendency (actual,previous):
    if actual > previous:
        tendance_energy = "augmenté"
        commentaire = "Soyez vigilants."
    else :
        tendance_energy ="diminué"
        commentaire = "Bravo."

    rapport_consommations = (abs(actual-previous)/previous)*100

    return (tendance_energy,commentaire, rapport_consommations)
```

Figure 7 : Script de la fonction permettant la comparaison des valeurs

Cette fonction permet de connaître l'évolution générale en pourcentage.
L'évolution de la consommation sert uniquement à titre indicatif car la variation de température n'est pas prise en compte. Or il est normal de consommer plus lorsque la température diminue ou augmente fortement.

Remarque: Nous avons fait quelque chose de similaire avec la puissance. La seule différence est la valeur de l'index pour la base de données et la réalisation d' une somme à la place d'une soustraction.

Plus de détails sur ce code sont disponibles en **Annexe 5**.

Résultats

Le type de message que l'utilisateur pourra recevoir est présenté sur la figure 8.

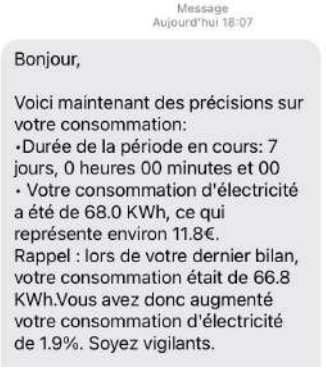


figure 8: Exemple de message que l'utilisateur reçoit pour l'informer de manière générale sur sa consommation

5.2 Équivalent CO_2 de la consommation

Contexte

Dans le monde, la production d'électricité est une source importante de CO_2. De plus, en fonction du moment de la journée, un kilowattheure d'électricité ne représente pas la même quantité de CO_2 (voir exemple en figure 9). RTE nous donne accès à l'équivalent CO_2 d'un kilowattheure d'énergie produit.

Figure 9: Allure de l'équivalent CO2 de la production durant une journée [2]

Informer hebdomadairement l'utilisateur de son équivalent CO_2 sera un bon moyen de lui faire prendre conscience de la pollution liée à sa consommation, ce qui l'incitera premièrement à la réduire mais aussi à consommer au meilleur moment.

Méthode

Pour calculer avec précision l'équivalent en CO_2 de la consommation, nous récupérons les données d'émission de CO_2 de RTE (qui ont un pas de temps de 15min). Pour cela, nous utilisons le code permettant de récupérer les données RTE en choisissant bien la colonne correspondant à la masse de CO_2/kWh. A partir de cela, nous construisons un vecteur '*CO2*' à 672 valeurs (car il y a 672 fois 15 minutes dans une semaine). Une composante du vecteur correspond à une équivalence en CO_2 d'un kilowattheure sur 15 minutes, la première valeur du vecteur étant la plus ancienne.

Nous calculerons pour une semaine, l'énergie consommée toutes les 15 minutes, grâce à une boucle 'for' qui s'exécutera 672 fois (*number_of_15min*). Cette boucle permet de créer un CSV et d'extraire l'énergie consommée sur une période de 15 minutes. A la fin de la boucle, nous effectuons un décalage temporaire pour extraire les 15 minutes précédentes et ainsi de suite.
Nous aurons donc 672 valeurs de consommation. Chaque valeur sera stockée dans le vecteur '*energy_vector_15*' pour lequel nous inversons l'ordre des éléments (afin d'avoir une cohérence entre ce vecteur et le précédent). Le détail du code est présent en Figure 10.

```
#loops over slots of 15 min over the time
for i in range(numbers_of_15min):
    #creation and reading of a CSV file
    Data_recovery_and_CSV_creation.csv_creation(now, dt15min, "Passed_Energy_15min.csv", 1502)
    df_kWh15min = pd.read_csv('Passed_Energy_15min.csv')
    #calculation of the value of consumption (last one - frist one) using the CSV file
    kWh15min = df_kWh15min.iloc[-1,1] - df_kWh15min.iloc[0,1]
    energy_vector_15[i] = kWh15min/1000
    now = now - dt15min #adding 15min intervall for the imported data
```

Figure 10 :boucle 'for' permettant de construire le vecteur de consommation.

Ce code a été écrit afin de n'avoir à changer que le temps global en modifiant *delta_time* pour faire les calculs sur une autre période.
Nous n'avons finalement qu' à faire les produits termes à termes des composants des vecteurs *CO2* et *energy_vector_15*. La somme de ces produits nous donnera l'équivalent en grammes de CO_2 de la consommation globale de la semaine. La figure 11 montre comment coder cela.

```
CO2_g = int(sum(energy_vector_15 * CO2))
```

Figure 11: Calcul final de l'équivalent CO_2 de la consommation

Plus de détails sur ce code sont disponibles en **Annexe 6.**

Résultats

Grâce à l'IHM, nous pouvons informer l'utilisateur de cela. Une illustration des mises à jour hebdomadaires est présentée dans la figure 12.

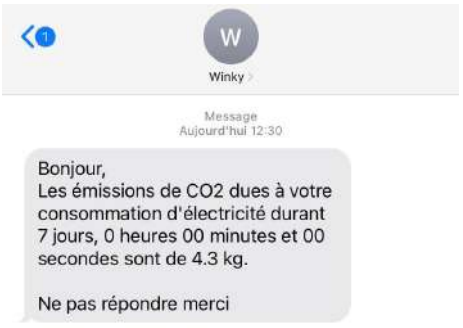


Figure 12: Résultat lié à l'équivalent CO_2

6. Détection des anomalies de consommation

Le but de ce code est de créer une routine , c'est-à-dire, un profil moyen de consommation par tranches horaires, sur les 7 derniers jours et ainsi obtenir une courbe de consommation type du foyer sur une journée (voir figure 13).

Une fois cette consommation type calculée, il sera possible de la comparer à chaque instant à la consommation du foyer. Si une surconsommation est détectée, l'utilisateur en sera alors alerté.

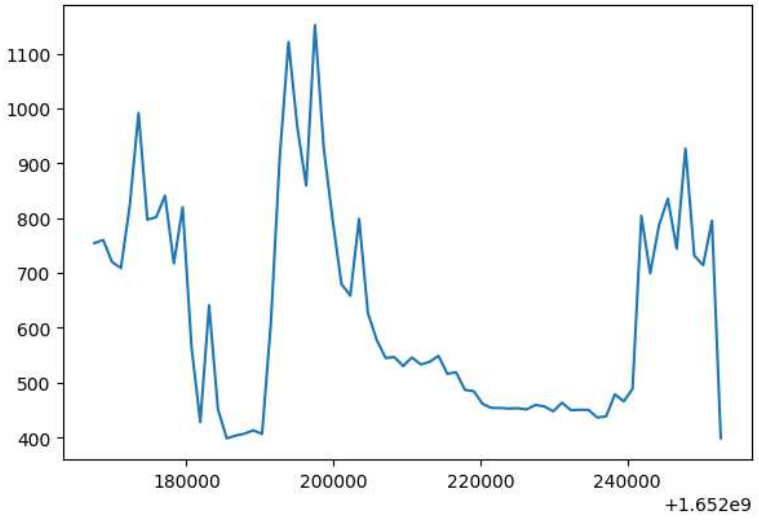


Figure 13 : Courbe de puissance de la consommation type (W) du 4 au 11 mai 2022

Nous somme parvenus à un script python calculant une fois par jour, dans la nuit, cette routine de consommation. Nous comparerons ensuite, en temps réel, au fil de la journée, cette dernière avec la consommation instantanée du foyer.

Il y a eu plusieurs étapes pour arriver à ce script. Au début, nous pensions pouvoir recalculer à chaque instant la nouvelle moyenne de consommation. Cependant il s'est avéré que la récupération de données et le calcul de la moyenne étaient bien trop long malgré les optimisations successives que nous avons effectuées sur le code. Il a donc fallu changer de stratégie. Nous avons ainsi décidé de calculer une fois par jour, dans la nuit, la routine pour pouvoir l'utiliser toute la journée suivante.

Cette solution permet de résoudre le problème du temps de calcul, cependant le calcul prend environ 1h (dépendant de la qualité du wifi et de la précision du pas de temps choisi). Durant le calcul, aucune comparaison n'est possible. De plus, si un problème de consommation survient, il ne sera pas détectable.

Un envoi de message est effectué lorsque la consommation actuelle à l'instant t est 3 fois supérieure à la consommation type (voir figure14). Cette valeur à été fixée après de nombreux tests. Elle permet d'éviter la surcharge en SMS mais également de ne pas perdre d'informations. Un temps de sécurité a aussi été ajouté. Ainsi si une notification est envoyée, il faudra attendre 1 heure pour que le script soit en mesure d'envoyer une nouvelle notification. Cette mesure a pour but d'éviter la saturation d'informations et de SMS auprès de l'utilisateur.

Figure 14: Résultat du code de détection des anomalies à 10h05 le 11 mai 2022

Les valeurs de consommation type sont calculées à partir des 7 derniers jours cependant on peut remarquer une importante différence de consommation entre le weekend et la semaine ou lors de vacances par exemple. L'idée d'un code prenant en compte seulement les jours de la semaine ou ceux du weekend nous a paru intéressante, cependant nous n'avons pas eu le temps de le mettre en place.

Pour plus de précision en ce qui concerne le code, voir **Annexe 7**

7. Consommer au moment opportun

7.1 Prédictions de production par un réseau de neurones

Le réseau électrique national peut parfois subir des pics de consommation et être surchargé. Il est donc intéressant de savoir quels sont les moments les plus propices à la consommation. On considère qu'il est plus adéquat de consommer hors des pics de consommation nationale et lorsque la production d'énergies renouvelables occupe une part plus importante dans le mix énergétique.
Afin de résoudre ce problème, nous proposons de réaliser la prédiction de production d'électricité et la prédiction de la part de production d'énergies renouvelables à l'aide de l'intelligence artificielle.

Le réseau de neurones LSTM

Un réseau neuronal artificiel décrit un algorithme d'apprentissage automatique qui tente d'imiter le réseau de neurones humains. Il s'agit d'une technologie très répandue dans divers domaines d'études. Le réseau de neurones artificiel est constitué d'un nombre important de processeurs, représentant les neurones, opérants en parallèle et organisés en couches.
Afin de traiter au mieux des séries temporelles, ce qui correspond ici aux données de production et de consommation d'électricité, deux choix s'offraient à nous : le réseau neuronal Long Short Term Memory (LSTM) et la méthode statistique Seasonal Autoregressive Integrated Moving Average (SARIMA). L'apprentissage de SARIMA étant particulièrement long, cela pourrait bloquer d'autres fonctions dans la raspberry, c'est pourquoi nous utiliserons plutôt le LSTM. Nous expliquons son fonctionnement de manière plus détaillée en **Annexe 8**.

L'entraînement LSTM - Méthodes

Nous avons choisi de créer deux algorithmes basés sur le LSTM expliqués de manière détaillée en **Annexe 9**. Le premier permet à l'utilisateur de connaître le moment où la production nationale sera la plus élevée et la plus basse. Pour cela, l'algorithme récupère les données de consommation de la dernière semaine sur le site RTE, soit une centaine d'heures. On assimile alors la courbe de production à celle de consommation, car elles sont similaires. Le réseau met à jour les poids de chaque neurone dans le réseau pour chaque *epoch* et ce pendant quelques centaines d'*epoch*. Si le coût converge et est assez petit, on peut arrêter l'entraînement. La précision du réseau n'étant fiable que pour une dizaine de valeurs consécutives, nous lançons la prédiction deux fois par jour, une fois à quatre heures et l'autre à seize heures. On indique finalement le minimum et le maximum de prédiction de consommation pendant une demi-journée en envoyant un message aux utilisateurs. Ils sont donc capables de prévoir le bon moment pour consommer l'électricité de manière à éviter la surcharge du réseau.

Le deuxième algorithme permet quant à lui, de trouver le moment, pendant les 12 heures suivantes, durant lequel le ratio ci-dessous est le plus élevé, afin que l'utilisateur puisse décider de consommer plutôt à ce moment-là (c'est-à-dire quand la production renouvelable est maximale).

$$\frac{Production\ d'ENR}{Production\ d'énergies\ totales}$$

Pour cela, nous utilisons une fois encore des données de RTE. Nous faisons la somme de toutes les productions d'énergies renouvelables et on la divise par la somme de toutes les productions d'énergie sur le territoire français. Il s'agit ensuite d'entraîner le réseau de neurones et de le tester. Nous avons choisi de construire un réseau de deux couches composées de 400 neurones chacune car il s'agissait du meilleur compromis entre rapidité et précision.

```
#définition des générateurs d'entraînement et test
train_generator = TimeseriesGenerator(X_train, X_train, length=seq_len)
test_generator = TimeseriesGenerator(X_test, X_test, length=seq_len)

#définition du réseau
model = Sequential()
#on ajoute deux couches d'LSTM de 400 neurones
model.add(LSTM(400, return_sequences=True, activation='relu', input_shape=(seq_len, 1)))
model.add(LSTM(400, activation='relu', input_shape=(seq_len, 1)))
#couche sortie
model.add(Dense(1))
```

Figure 15 : Définition du réseau de neurones et des générateurs d'entraînement et de test

Résultats

Généralement, les deux réseaux de neurones créés fonctionnent correctement comme nous pouvons le voir sur les graphiques suivants (Figure16). Ils présentent une comparaison entre les prévisions et les données réelles sur un peu plus de 80 heures. On constate que les courbes ont la même tendance.

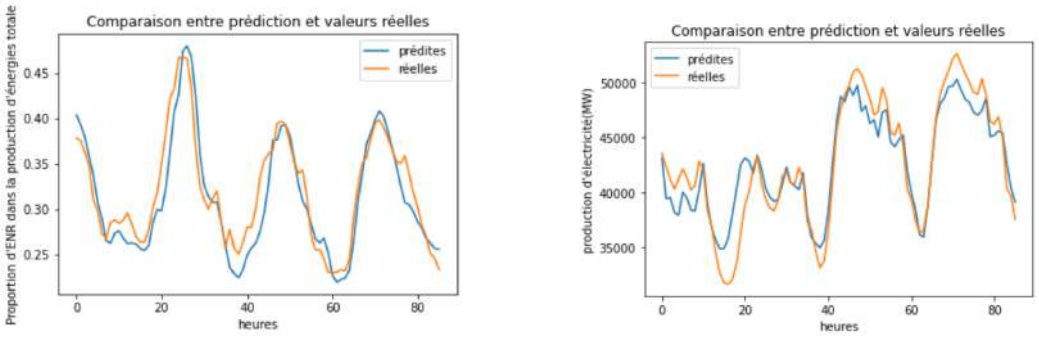


Figure 16: Comparaison de la prédiction en bleu et les vraies valeurs en orange pour la proportion d'énergie renouvelable à gauche et la consommation d'électricité à droite.

Finalement, grâce à ces réseaux de neurones, nous obtenons deux prédictions dans la journée pour deux tranches horaires. La première pour la période 4h-16h00 et la seconde pour la période 16h00-4h00. Des exemples de prédiction pour les deux algorithmes dans la première tranche horaire sont présentés ci-dessous (Figure17).

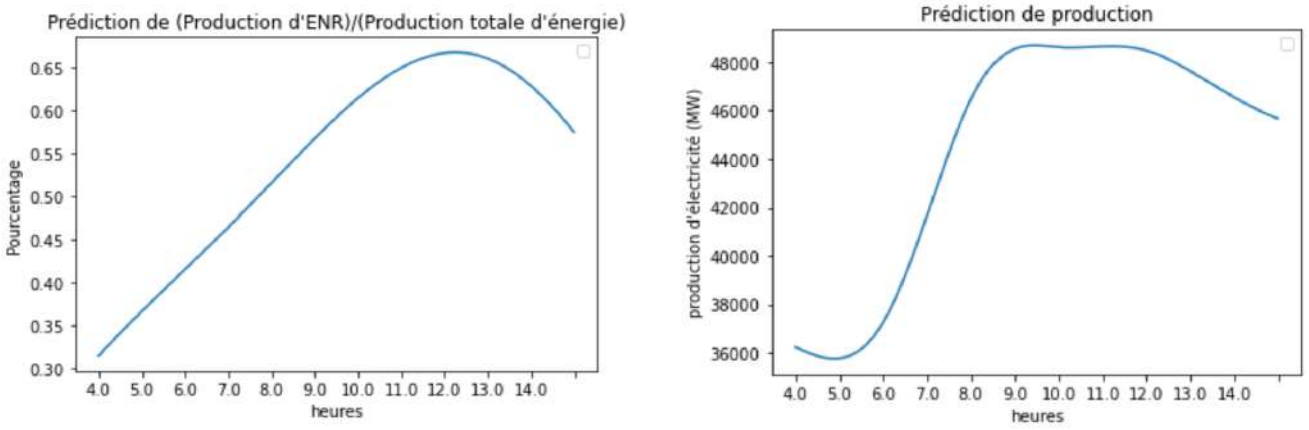


Figure 17: Les prédictions de part d'énergies renouvelables à gauche et de production d'électricité à droite.

L'utilisateur sera informé des moments adéquat de consommation via l'IHM comme présenté en Figure 18:

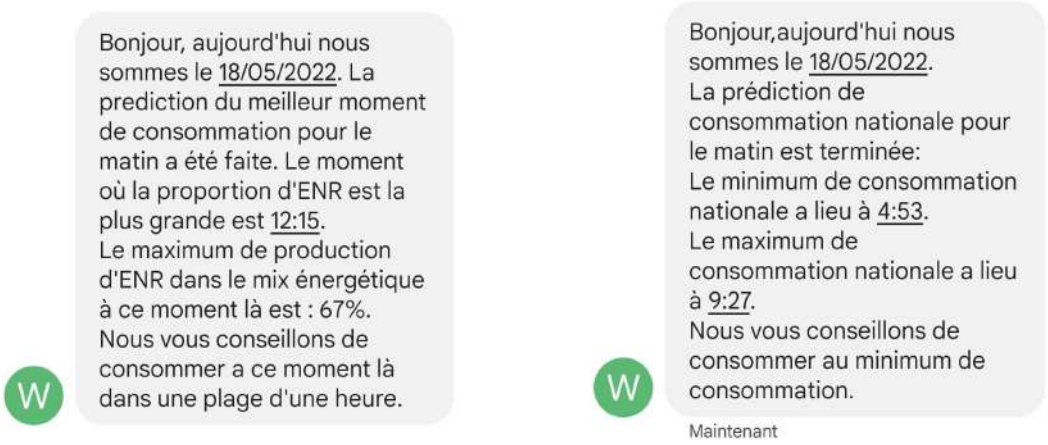


Figure 18: Exemple de SMS reçu par l'utilisateur concernant les prédictions

Les coûts des algorithmes sont ici de l'ordre de 0,0011 à la fin.

7.2 Prévision du pic de consommation

Le principe de ce code est de trouver (d'une manière différente), grâce aux prévisions de RTE, les moments de la journée où il y a le plus de consommation pour en informer l'utilisateur. Le but est que l'utilisateur prenne en compte ces informations dans sa consommation pour, qu'à terme, la courbe de consommation totale française s'aplatisse pour faire disparaître les pics et potentiellement réduire la demande en énergie.

Le code s'exécutera une seule fois par jour juste avant le réveil pour que l'utilisateur ait les données les plus précises possible et quand il en a besoin.

Nous allons considérer qu'un pic de consommation est présent lorsque la consommation est supérieure à 80% de l'écart entre le minimum et le maximum de consommation prévue de la journée (exemple en Figure 19).

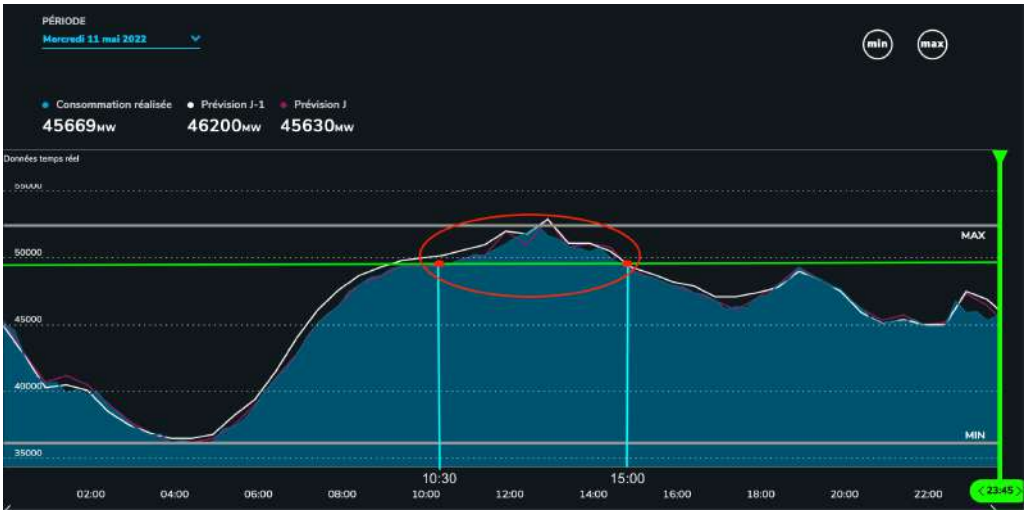


Figure 19: Courbe de la consommation électrique française du 11 mai 2022 (source: www.rte-france.com)

La valeur de 80% a été définie de sorte à obtenir des pics ne s'étendant pas sur une trop grande partie de la journée mais étant assez grands pour que cela soit représentatif de l'état du réseau.
Chaque matin à une heure qui est modifiable par l'utilisateur, le code exécute le calcul des heures de forte consommation. Le code est capable de segmenter les moments de forte consommation s'il y en a plusieurs dans la journée pour en informer l'utilisateur. De plus l'utilisateur est informé de l'heure à laquelle le pic de consommation est atteint en France ainsi que sa valeur pour qu'il fasse d'autant plus attention à cette période (Figure20).

Figure 20: Résultat du code de calcul de pic pour la journée du 11 mai 2022

A petite échelle, l'utilisation du code ne semble pas avoir un grand impact sur la consommation en France et semble être plutôt une contrainte pour l'utilisateur. Cependant, si une grande quantité de foyers sont équipés de ce dispositif, on peut espérer une diminution des pics de consommation en France. Or les pics de production/consommation induisent généralement l'usage de moyen génération d'électricité plus polluant, d'où l'intérêt de les réduire. L'utilisateur pourra voir l'impact de ses changements d'habitude grâce à la modification de son équivalent CO_2.
Pour plus de détails voir **Annexe 10.**

8. Transférer les codes dans la raspberry

Une fois nos codes terminés, il est nécessaire de les transférer dans une RaspberryPi (la même que celle qui communique avec le capteur) et d'y insérer un service. Cela permet aux codes de s'exécuter de manière automatique et en boucle (avec un durée déterminée entre chaque boucle). Lors de l'installation des codes dans la Raspberry Pi, il est parfois nécessaire d'installer des modules dans cette dernière. Cette étape étant complexe pour des personnes novices avec les systèmes d'exploitation Linux, nous avons écrit un tutoriel (disponible en **Annexe 11**).

9. Livrables

Dans cette partie, nous allons présenter ce que nous avons réussi à faire ainsi que les livrables que nous avons mis en place afin que notre travail soit Open Source, Open Science et puisse être repris par d'autres personnes.

9.1 Retour d'expérience sur les Winky

Après avoir fabriqué une série de Winky et d'avoir tenté de les mettre en place dans tous les appartements des membres du groupe, nous avons réalisé un retour d'expérience à notre porteur de projet (créateur du capteur). Cela permettra à notre porteur de projet de mieux se rendre compte du degré de difficulté de la manipulation et des choses à améliorer, ce qui est particulièrement important pour l'aspect de reproductibilité de la recherche scientifique.

Le coût individuel

Dans un contexte "Open Source/Open Science", ces capteurs sont réalisables avec du matériel de base en électronique (carte PCB, capacités, fer à souder...) afin qu'ils soient constructibles par des personnes non expertes dans un FabLab. Pour réaliser l'installation complète de récupération des données, du matériel supplémentaire est nécessaire: une raspberryPi, un câble éthernet, une box internet, éventuellement un routeur… Cette installation peut donc rapidement devenir "coûteuse" pour un particulier (même si le coût de revient pour le G2Elab est faible, dû aux grandes quantités et à la capacité d'impression des cartes PCB en interne) ce qui pourrait être un possible frein pour des particuliers. Nous avons calculé le coût approximatif pour un particulier: 214€ (voir figure 21).

	Prix en €
RaspberryPi en kit	79,99
Composants du Winky	22,69
Carte PCB	111,8
Total pour un particulier	**214,48**

Figure 21: Calcul du coût de l'installation pour un particulier [sans prise en compte de la box internet]

Dans notre cas, cela n'a pas été un frein car la totalité du matériel était fournie par le G2Elab. Pour eux, le coût de revient de d'une installation est d'une dizaine d'euros pour le capteur plus le coût d'une RaspberryPi.

Importance du compteur

Pour installer le Winky, dans l'état actuel, un compteur Linky en mode "historique" est indispensable, ce qui élimine une partie des intéressés. Dans notre cas, sur 7 membres, seulement 4 possèdent un compteur totalement compatible et 1 possède un compteur en mode "standard". Suite à cette constatation notre porteur de projet a décidé de travailler activement à l'adaptation du capteur pour s'adapter à tous les types de compteurs intelligents.

En définitive, ce capteur est un théorie réalisable par un particulier mais la contrainte financière ainsi que des contraintes pratiques liées au type et à la position du compteur (qui ne doit pas être trop éloigné de la box Internet) peuvent être un frein important.

Fabrication

Les 4 membres disposant des compteurs compatibles ont donc été les testeurs de la fabrication et l'installation de ce capteur qui a pour vocation d'être réalisable par tous.

Des Winky

Nous avons testé la fabrication du capteur. A la création du planning, nous avions prévu une journée pour la fabrication de tous les Winky. Les cartes PCB étant déjà imprimées, nous avons juste dû les percer, puis souder les composants. Néanmoins, ces deux étapes se sont avérées bien plus complexes que prévu, notamment la phase de soudage.
Pour souder un Winky par personne, il nous a fallu 3h chacun et l'aide de Jérôme était indispensable pour souder les composants complexes (comme une micro soudure sur le microcontrolleur ESP8266). A la fin de cette journée de soudage, tous les Winky avaient des problèmes liés aux soudures. Jérôme à donc conclu grâce à notre phase de test qu'une carte PCB double face était à bannir car elle compliquerait la fabrication. Il a donc réalisé une nouvelle version de carte, simple face, donc plus simple à souder.

Du module de téléversement

Un module de téléversement est nécessaire pour implanter un code dans le micro-contrôleur présent sur le capteur (ESP8266).
Nous avons eu l'occasion de réaliser ce module, pour cela nous avons suivi les indications et schémas présents sur le site 'miniprojets' (site Web utilisé par notre porteur de projet [3]) afin de souder les composants aux bonnes bornes. Nous avons réussi à réaliser par nos propres moyens ce module cependant le tutoriel n'est pas adapté à des personnes n'ayant pas de compétence en électronique. En effet, nous avons eu quelques doutes en ce qui concerne la liaison des fils à l'arrière de la plaque car aucune photo n'est disponible sur le site. De plus, l'avant de la plaque est constitué de fils de même couleur ce qui peut perdre l'utilisateur lors de la conception de son module.

Nous préconisons donc l'ajout d'une photo de la face côté soudure et de changer la photo côté fils pour une plus lisible (avec des fils de couleurs). En **Annexe 12**, nous vous présentons un tutoriel pour la réalisation du module accompagné de photos pour la réalisation de ce capteur.

Installation

Parmi les 4 membres qui ont tenté l'installation complète des Winky, aucun n'est parvenu à acquérir des données. Ci-dessous, nous présentons nos difficultés:

Une fois les difficultés liées à la configuration de Jeedom surmontées, nous avons fait face à un problème de communication entre l'ESP8266 et la box internet de certains appartements. Cela a été résolu grâce à l'utilisation de routeurs.

Cependant, même avec la communication établie correctement certains capteurs n'arrivent pas du tout à acquérir de données. Dans le meilleur des cas, des données sont récupérées pendant quelques minutes après le branchement avant qu'un arrêt total survienne.

Avec l'aide de notre tuteur, nous avons essayé de résoudre ce problème de plusieurs manière:

- La première fut de tester les capteurs dans la salle 4A013 de Green-Er. Nous avons une incompréhension sur ce point car les capteurs fonctionnent parfaitement. Nous avons donc déduit que le problème venait des compteurs grenoblois.
- La seconde à été de vérifier les données électriques des compteurs grenoblois, dont, les tensions au bornes de la prise TIC du compteur et aux bornes des supercapacités grâce à des multimètres portatifs que nous avons amenés chez nous (Figure 22).

Figure 22: Vérification de la tension
Aux bornes des capacités

Rien ne paraissait anormal, nous avons donc décidé d'aller plus loin avec l'utilisation d'un oscilloscope. L'utilisation de l'oscilloscope nous permet de mieux visualiser la charge/décharge de la capacité. Nous obtenons les résultats suivant:

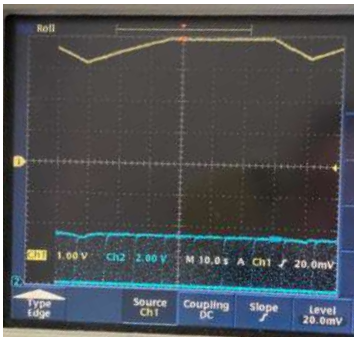
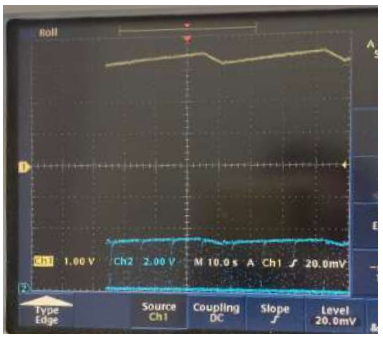
Acquisition des données: OUI	Acquisition des données: NON
	

Figure 23: Vérification du chargement des capacités grâce à un oscilloscope

On remarque sur le figure 23, que la tension au bornes des capacité (en jaune) n'a pas la même allure juste après le branchement [c'est à dire quand des données sont en cours d'acquisition] et quelques minutes après [quand il n'y a plus de données acquises]. On confirme donc qu'il y a bel et bien un problème. Pour rectifier cela, notre porteur de projet a pensé à augmenter le temps de chargement de la capacité.

Malheureusement, ce changement n'a pas permis à l'installation de fonctionner. Nous avons donc décidé d'utiliser les données de EXPSmartHouse afin de continuer notre projet. EXPSmartHouse est une maison de quatre personnes équipée de nombreux capteurs dont un Winky. Cette plateforme de démonstration fournit des données similaires à celle que nous aurions pu acquérir.

Pour conclure, suite à notre retour d'expérience le créateur de capteur a effectué 6 versions différentes du Winky. Il a changé les cartes PCB, la disposition des composants et à changer le micro-contrôleur pour un plus performant en pensant que les problèmes venaient principalement de ce dernier. Nous proposons également de nouveaux tutoriels.

9.2 Codes Python

Notre projet étant pour la majeure partie du traitement de données grâce à des codes Python, ces derniers constituent la partie la plus importante de nos livrables. Les codes livrés sont commentés de manière détaillée.

Pour finir, afin que le projet puisse être repris, chaque code est accompagné d'un descriptif détaillé sur son fonctionnement et son utilisation. Ces descriptifs sont disponibles dans les **Annexes 2, 3, 5, 6, 7, 9 et 10**.

9.3 Autres livrables

Tutoriels

Les autres livrables sont essentiellement des tutoriels qui ont été réalisés tout au long du projet. Ils ont pour objectif de faciliter la reprise du projet par d'autres étudiants mais également de permettre à notre tuteur de s'approprier notre solution.
Ces tutoriels sont également disponibles dans les **Annexes 1, 4, 11 et 12.**

Le livrable finalement non réalisé

Au début du projet, nous avions pour objectif de fournir de nouvelles données électriques à notre porteur de projet. Les capteurs ne fonctionnant pas, nous n'avons aucune donnée à fournir.

10. Conclusions technique et perspectives

Lors de ce projet, nous avions pour objectif de trouver des solutions afin de mieux informer l'utilisateur sur sa consommation ainsi que de l'inciter à changer ses habitudes afin de participer à son échelle à la transition énergétique.

Pour ce faire, nous souhaitons accéder à la consommation électrique en temps réel grâce à un capteur facilement réalisable dans un FabLab: le Winky. Nous avons donc chacun réalisé un Winky ce qui ne s'est pas avéré réellement accessible pour des personnes qui n'avaient jamais fait cela auparavant. Nous avons également installé chacun de ces capteurs dans nos appartements respectifs. Cependant, il s'est avéré que cette installation ne s'est pas passée comme prévu et que nous n'avons réussi à faire fonctionner aucun capteur. Néanmoins, cela nous a tout de même permis de contribuer à l'amélioration de ce capteur grâce à notre retour d'expérience sur la réalisation et l'installation.
Pour aller plus loin dans notre projet et tout de même trouver une solution afin de satisfaire nos objectifs, nous avons utilisé des données existantes et qui nous étaient accessibles, celles de EXPSmartHouse. Ces données sont présentées de la même manière que celles que nous aurions pu acquérir et sont représentatives de la consommation d'un foyer de 4 personnes.

Finalement, les solutions misent en place permettent :

- D'informer l'utilisateur sur sa consommation sur une période de temps donnée ainsi que son évolution d'une période à l'autre. De cette manière l'utilisateur prend conscience de ce que représente réellement sa consommation.
- De réaliser une prédiction du meilleur moment de consommation pour éviter les pics de consommation tout en consommant lorsque les énergies renouvelables sont importantes grâce à l'intelligence artificielle.
- De faire prendre conscience de l'impact du moment de consommation à l'utilisateur grâce à l'équivalent CO_2 de sa consommation. En effet, en l'informant du moment où la production renouvelable est la plus importante, nous l'invitons à réduire son équivalent CO_2. Cela l'aidera également à prendre conscience de son impact environnemental.
- De prévenir l'utilisateur lorsque sa consommation est anormalement importante, en prenant pour référence sa consommation habituelle.
- Informer l'utilisateur des heures où il est préférable de ne pas consommer en utilisant les prédictions de RTE. Cela permettrait à grande échelle de lisser la courbe de consommation/production.
- Une IHM permettant l'envoie de SMS a été mise en place afin de communiquer avec les utilisateurs. Cette solution permet de prévenir l'utilisateur en temps réel et est simple d'utilisation.

Toutes ses solutions sont fonctionnelles en possédant dès lors que l'accès à des données de consommation en temps réel fournies pour une installation Winky existent.

La prochaine étape de ce projet pourrait être de parvenir à solutionner les problèmes de fonctionnement liés au Winky afin de pouvoir tester nos solutions sur plus de foyers. Il serait également intéressant de créer plus d'intéraction entre les différentes solutions trouvées

Durant ce projet, nous avons appris à mieux manipuler la gestion et le traitement de données informatiques. Nous avons également pû découvrir comment fonctionne l'intelligence artificielle. Pour finir, ce projet nous a énormément appris à propos de la gestion de projet.

Partie 2: Gestion de projet

Ce projet était l'occasion pour nous, futurs ingénieurs, de mettre en pratique certaines pratiques de gestion de projet que nous avons apprises. Cependant, étant novices dans ce domaine, nous avons énormément appris.

1. Travail en équipe et organisation

Un des principaux objectifs des projets d'ingénierie est d'apprendre à travailler en équipe. Notre groupe étant composé de 7 membres avec des nationalités variées, des savoirs différents et des vécus divers, nous avons eu l'occasion de beaucoup apprendre les uns des autres que ce soit au niveau de notre approche d'un problème ou de notre organisation de travail.

1.1 Interculturalité

Premièrement, nous avons compris l'importance de l'interculturalité : Marie étant suédoise, Yukang chinois, et le reste de l'équipe français, nous avons dû admettre que chacun avait des habitudes de travail différentes. Notamment en ce qui concerne la rédaction des différents rapports et leurs structures. Chacun avait sa propre vision mais une approche "Française" a été privilégiée pour ce qui est de l'organisation mais surtout au niveau du langage utilisé pour la communication entre nous et la rédaction des différents documents. Ce choix s'est basé sur le fait que la grande majorité du groupe est français et que Yukang est plus à l'aise avec le français qu'avec l'anglais. Chacun avait néanmoins la possibilité de rédiger ses parties comme il lui semblait le plus pratique. Par exemple, Marie a préféré faire ses rédactions en anglais et nous les traduisions en français par la suite.

1.2 Changement de chef de projet

Au début du projet, nous avons défini les rôles de chacun, Guillaume a été nommé responsable technique et Antonin, responsable financier pour leurs intérêts respectifs à ce propos. Le rôle de chef de projet a été attribué à Marie pour sa rigueur et sa gestion du temps. Finalement, au cours du projet, Marie, de sa propre initiative, a préféré renoncer à ce poste en raison des barrières de la langue. C'est donc Marine, pour les mêmes raisons, rigueur et gestion du temps, qui a finalement pris en charge ce rôle.

1.3 Organisation

Concernant l'organisation du travail, nous réalisons toutes les tâches dont le développement des algorithmes sur les créneaux réservés au projet. Durant ces journées, nous avons décidé de commencer à 8h30 jusqu'à 12h15 puis de reprendre à 13h30 jusqu'à 17h. La cohésion d'équipe était favorisée par des pauses communes mais également par un travail autour d'une unique grande table ce qui facilitait les échanges entre les différents membres du groupe. Concernant le travail en dehors des séances, il était consacré aux essais d'installation des Winky chez les membres du groupe.

Concernant la répartition du travail, tous les membres de l'équipe n'avaient pas les mêmes points forts/faibles, certains étaient très à l'aise avec la programmation et d'autres non. Nous avons donc fait le choix, pour que chacun trouve sa place et ne soit pas mis de côté, de diviser le travail en petites équipes avec des tâches bien précises mais interconnectées. Ainsi, chacun a pu évoluer à son rythme et s'occuper de parties qui les intéressaient.

Les équipes étaient les suivantes:
-Marine et Marie: consommation moyenne ainsi qu'évaluation sur 2 périodes et équivalent CO_2 .
-Antonin et Clara: Récupération des données RTE, calcul des pics de consommation et comparaison de la consommation actuelle avec une consommation type basé sur la dernière semaine
-Emma et Yukang: Elaboration d'un réseau de neurones permettant la prédiction de production nationale et la proportion d'énergies renouvelables
-Guillaume: Interface Homme Machine

Ces petites équipes ont été un atout pour l'avancée du projet car chacun avait son petit projet sur lequel il pouvait avancer sans avoir besoin des autres. Mais cette organisation s'est également avérée être un désavantage. En effet, les équipes ne connaissaient pas précisément les tâches, les avancées, les changements d'objectifs, etc. des autres équipes. Cela est devenu un problème au milieu du projet, au moment de rassembler les résultats. En effet, les retards et les changements d'objectifs respectifs ont conduit à quelques tensions ce qui nous a valu de faire un point pour mettre les choses au clair au sein du groupe avec notre porteur de projet. Nous avons alors compris l'importance de la communication et de la connaissance de l'avancement global au sein d'une équipe. Nous avons donc décidé de renforcer le rôle du chef de projet et de mettre en place des réunions tous les mercredis à 9h puis à 15h. La première permet de définir clairement les objectifs du jour et la seconde de faire le point sur les problèmes et avancées de chacun. Cela nous a permis d'avoir une meilleure vision d'ensemble sur l'avancée du projet et de pouvoir s'entraider les uns les autres par exemple si certains rencontraient des problèmes que d'autres avaient déjà réussi à résoudre la communication permettait de s'en rendre compte. Finalement, nous avons réussi à résoudre ce problème en cours de projet. Néanmoins, cette prise de conscience tardive ne nous a pas permis de fournir une unique solution mais un panel de solutions possibles. Il aurait été très intéressant de rassembler ces solutions.

Cette constatation est due principalement à la répartition des tâches mais peut également être expliquée suite à l'apparition d'un imprévu qui nous a conduit à prendre du retard.

2. Gestion des risques/imprévus

Au début du projet, nous avons établit une liste des risques potentiels:

Description	Criticité	Responsable	Prévention	Solution
Gestion du temps	Moyen	Marine	GANTT et division des tâches Rapport hebdomadaire destiné au porteur de projet	Réduire les objectifs
Trop ambitieux	Faible	Marie	Discussion régulière avec les porteurs	Réduire les objectifs
Manque de connaissances	Faible	Antonin	Validation des objectifs par les porteurs	Réduire les objectifs
Problèmes liés aux capteurs	Moyen	Clara	Avoir conscience que ce risque est important	Utiliser les capteurs et/ou données de notre porteur de projet
Risque humain	Moyen	Emma	Echange très fréquent entre les membres sur leurs tâches	Changer la répartition des tâches
dépendance à la technologie	Moyen	Yukang	Sauvegardes à plusieurs endroits	/
IHM: visualisation des résultats	Haut	Guillaume	Solution de secours	SMS ou application déjà existante

Lors du démarrage du projet, les objectifs que nous avions définis étaient bien trop ambitieux, nous nous sommes rendus compte de cela assez tôt grâce à un échange avec nos porteurs de projet. Nous avons ainsi réalisé qu'il y avait un réel risque ici. Cependant, il a pu facilement être évité grâce à de la communication avec des personnes capables de faire un bilan objectif de nos compétences. Cela est finalement corrélé avec le risque de "manque de connaissances".

Le risque lié à l'interface homme machine avait été estimé comme haut car sans celle-ci, nous ne pouvions pas visualiser le résultat de notre travail. Finalement, nous avons abandonné l'idée de créer notre propre application car c'était trop ambitieux au vu de nos compétences et du nombre de personnes s'occupant de l'IHM. Nous avons alors gardé l'idée des SMS en utilisant une application préexistante pour permettre d'obtenir avec certitude une IHM fonctionnelle avant la fin du projet et de pouvoir l'utiliser pour visualiser et adapter les autres parties du projet.

Notre projet était composé d'une partie technique liée à la réalisation/installation des capteurs afin d'acquérir des données à traitées, et d'une autre partie uniquement informatique/algorithmique dépendante de ces données. Le risque majeur était donc de ne pas pouvoir récupérer les données dûes au non fonctionnement des Winky dans nos appartements. Ce risque était fort probable car nous devions les installer loin de l'école sans l'aide de notre porteur de projet. Nous étions, en règle générale, capables de trouver les causes de dysfonctionnement mais pas de les résoudre. Comme mentionné dans la partie technique, nous n'avons pas réussi à faire fonctionner les Winky. Nous avons donc utilisé notre solution de secours: utiliser les données de consommation déjà existantes, celles recueillies par Jérôme Ferrari dans sa maison. Sans cette solution notre projet aurait été un échec.

Finalement, nous avons compris que dans tout projet, la gestion des risques est importante. Il est nécessaire d'avoir des solutions de secours car tout ne se déroule pas toujours comme prévu, ce qui a été notre cas.

3. Gestion du temps

3.1 Planification

La planification du projet s'est faite principalement par un schéma du Gantt au début du semestre(voir **Annexe 13)**. Nous faisions également périodiquement des réunions pour mettre au point l'avancée des différentes parties.

3.2 Prise de retard

Lors des premières semaines du projets beaucoup de temps devait être consacré à la partie gestion de projet au travers de différents rendez-vous, livrables… Ces moments étaient évidemment nécessaires pour le bon déroulé de notre projet. Cependant, ce temps aurait pu être mis à profit d'une autre façon pour mieux cadrer le projet. En effet, les objectifs de notre projet n'étant pas réellement définis, nous avions alors eu l'opportunité de les choisir et de nous impliquer dans un projet qui nous intéressait. Néanmoins, nous n'avions pas le temps de nous plonger dans l'aspect technique pour savoir ce qui était réalisable et ce que l'on avait réellement envie car il était nécessaire de fournir rapidement nos objectifs à nos coordinateurs. Cela nous a donc amené à redéfinir plusieurs fois les objectifs et à prendre beaucoup de retard lors du lancement du projet.

Le point sur lequel nous avons pris le plus de retard par rapport à nos objectifs est l'installation des capteurs dans nos appartements. En effet, nous avions estimé initialement qu'en quelques semaines (deux ou trois) nous aurions réussi à faire fonctionner ces derniers. Finalement, rien ne s'est passé comme prévu, ce qui nous a conduit à nous égarer et à perdre du temps sur certaines tâches, notamment les nombreux essais d'installation, la résolution des problèmes rencontrés et surtout la réflexion sur une solution pour remplacer les données que nous ne pourrions pas acquérir.

Afin de parvenir à finaliser notre projet, nous avons fait le choix d'abandonner nos objectifs secondaires, notamment l'amélioration du capteur du nombre de personnes dans la pièce, afin de nous focaliser sur le cœur de notre projet.

Comme évoqué dans la partie gestion du travail, il y a eu un manque de communication qui a aussi causé de nouvelles pertes de temps. En effet, nous ne nous étions pas mis d'accord sur les besoins de chacun. Ainsi lorsque les travaux ont dû être regroupés il a fallu les réadapter pour les faire coïncider. Notamment en ce qui concerne l'IHM où tout le monde n'avait pas les mêmes idées des données à afficher et de comment les afficher. Ou encore certains codes qui ont été faits en doublons car plusieurs petites équipes avaient les mêmes besoins.

Nous pensions pouvoir limiter le retard pris grâce à l'utilisation du Gantt. Malheureusement, nous ne l'avons pas mis à jour ni utilisé régulièrement pour faire part de l'avancée des différentes parties du projet et nous rendre compte si les délais étaient respectés. C'est sûrement un des aspects qui a fortement joué sur nos problèmes de communication car nous ne pouvions pas suivre l'avancée des autres équipes. Il aurait également pu être utile afin d'ordonner plus facilement l'échelle d'importance des tâches à réaliser et de potentiellement se concentrer plus sur une des parties qui serait plus en retard que les autres. C'est un des enseignements que nous avons pu tirer de ce projet, la planification et le suivi de chaque étape du projet est important pour ne pas se disperser et permettre au projet d'aboutir. Le diagramme de Gantt actualisé est disponible en **Annexe 14**.

Conclusion

Grâce à ce projet, nous avons énormément appris au sujet de la gestion de projet. Nous avons compris qu'il est nécessaire de prendre du temps en début de projet pour définir clairement les objectifs, pour faire la planification, pour faire une analyse des risques… afin que tout le monde soit au clair avec ce qu'il y a à faire et les délais impartis. En effet, nous avons été confrontés à des retards qui n'avaient pas été anticipés ainsi qu'à un imprévu: le non fonctionnement des installations Winky. Nous sommes tout de même parvenus à mener le projet à bien grâce à la mise en place de la solution que nous avions évoquée lors de l'analyse des risques.

La répartition du travail en groupe a été une bonne chose car elle à permis à chacun de trouver sa place. Néanmoins, nous nous sommes rendus compte que pour une organisation comme la nôtre, il est important de communiquer afin de savoir exactement qui fait quoi et où il en est.
Suite à notre problème de communication, nous avons réalisé que pour qu'un projet soit mené à bien et épanouissant pour tous les membres du groupe, l'interaction entre les membres d'une équipe est indispensable. En effet, cela permet d'échanger sur les problèmes techniques, les avancées, le ressenti de chacun, les nouvelles propositions. De cette manière les problèmes qui peuvent être rencontrés comme les incompréhensions, les redondances (deux membres travaillant sur la même tâche), les déviations d'objectifs peuvent être évités ou a minima résolus rapidement.

Enfin, nous avons réalisé que chaque membre d'un groupe a toujours quelque chose à apporter au projet, notamment dans une équipe avec des profils aussi hétérogènes. Il est donc important de prendre en compte l'avis et les besoins de tous. Néanmoins, les différences entre les membres sont parfois sources de mésentente. Il est très important d'être à l'écoute des habitudes et émotions des autres membres même si chacun doit s'adapter un peu.

Références

[1] Ministère de la transition écologique . Répartition sectorielle des émissions de CO2 dans le monde [en ligne]. Disponible sur: https://www.statistiques.developpement-durable.gouv.fr/edition-numerique/chiffres-cles-du-climat/7-repartition-sectorielle-des-emissions-de [consulté le 10/05/2022]

[2] Gestionnaire du Réseau de Transport d'Electricité français. éCO2mix - Toutes les données de l'électricité en temps réel [en ligne]. Disponible sur: https://www.rte-france.com/eco2mix [consulté le 19/05/2022]

[3] Jérôme Ferrari. WinKy Version 2– Open-source projet pour Linky avec WiFi [en ligne]. Disponible sur: https://miniprojets.net/index.php/2022/02/04/winky-version-2-open-source-projet-pour-linky-avec-wifi/ [consulté de janvier à mai 2022].

Annexes:

Annexe 1: Tutoriel: Installation Jeedom et Winky

Le tutoriel se compose de 4 étapes:

- I - Initialisation de la Raspberry pi
- II - Installation de Jeedom
- III - Installation du plugin Mosquitto (MQTT) sur Jeedom
- IV - Installation du Winky

I - Initialisation de la Raspberry pi

Sur votre ordinateur, télécharger l'image du système d'exploitation via le lien : https://downloads.raspberrypi.org/raspios_oldstable_armhf/images/raspios_oldstable_armhf-2022-01-28/2022-01-28-raspios-buster-armhf.zip

Ensuite, télécharger le logiciel etcher sur ce lien: https://www.balena.io/etcher/

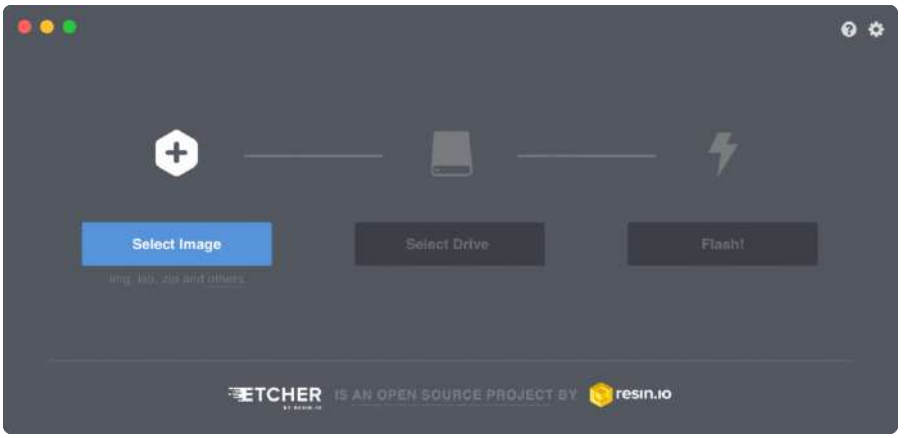


Figure 1: Une image de l'interface de ce logiciel.

Insérer une carte microSD dans l'ordinateur puis, dans le logiciel etcher, sélectionner l'image du système d'exploitation téléchargée plus tôt puis sélectionner la carte SD et enfin appuyer sur flash pour transférer l'image.

Dans la partie boot de la carte microSD (accessible depuis les fichiers windows), créer un nouveau fichier texte vierge nommé ssh.

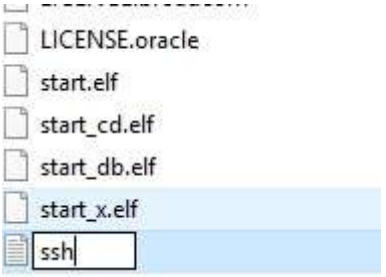


Figure 2: Nommé le nouveau fichier texte vierge 'ssh'.

Insérer ensuite la carte SD dans le Raspberry et connecter le Raspberry à la box internet via un câble ethernet et l'alimenter en courant avec le câble d'alimentation.

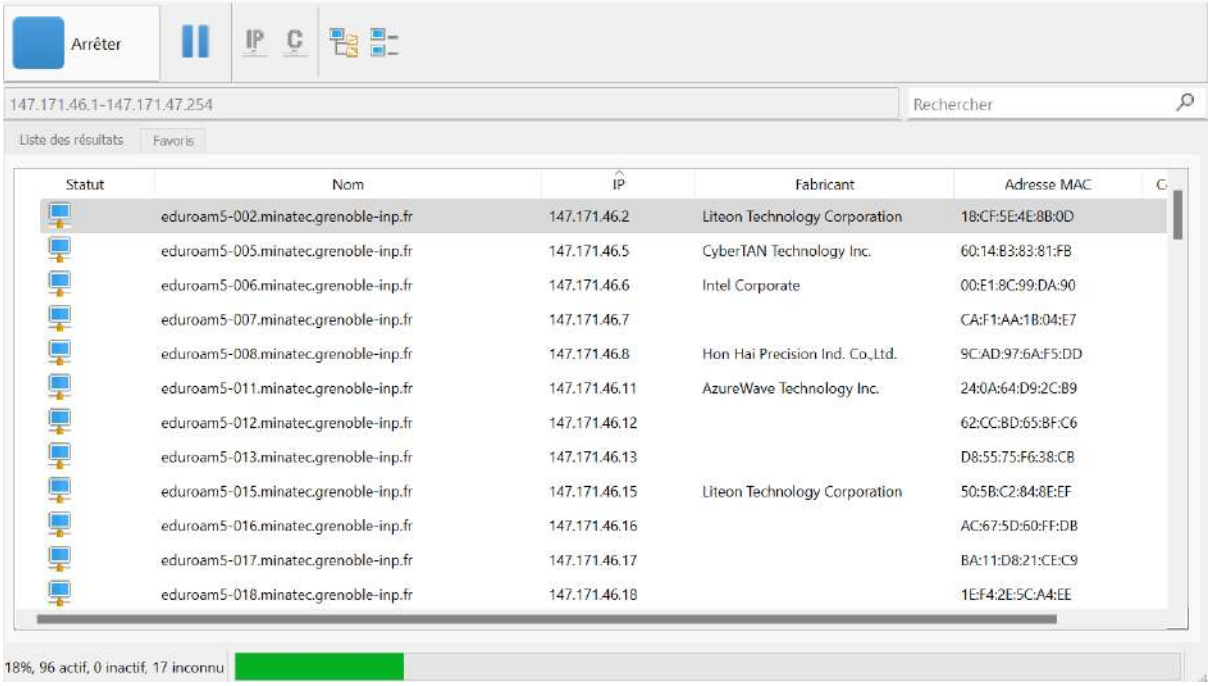
II - Installation de Jeedom

Télécharger le logiciel Putty : https://putty.fr.uptodown.com/windows

L'IP de votre Raspberry peut être trouvée avec deux méthodes.

Méthode 1 pour obtenir l'IP de votre Raspberry :
Télécharger le logiciel Advanced IP Scanner : https://www.advanced-ip-scanner.com/fr/

Grâce au logiciel Advanced IP Scanner, analyser et obtenir la liste des appareils connectés à la box internet et relever l'adresse IP du Raspberry.



Statut	Nom	IP	Fabricant	Adresse MAC
	eduroam5-002.minatec.grenoble-inp.fr	147.171.46.2	Liteon Technology Corporation	18:CF:5E:4E:8B:0D
	eduroam5-005.minatec.grenoble-inp.fr	147.171.46.5	CyberTAN Technology Inc.	60:14:B3:83:81:FB
	eduroam5-006.minatec.grenoble-inp.fr	147.171.46.6	Intel Corporate	00:E1:8C:99:DA:90
	eduroam5-007.minatec.grenoble-inp.fr	147.171.46.7		CA:F1:AA:1B:04:E7
	eduroam5-008.minatec.grenoble-inp.fr	147.171.46.8	Hon Hai Precision Ind. Co.,Ltd.	9C:AD:97:6A:F5:DD
	eduroam5-011.minatec.grenoble-inp.fr	147.171.46.11	AzureWave Technology Inc.	24:0A:64:D9:2C:B9
	eduroam5-012.minatec.grenoble-inp.fr	147.171.46.12		62:CC:BD:65:BF:C6
	eduroam5-013.minatec.grenoble-inp.fr	147.171.46.13		D8:55:75:F6:38:CB
	eduroam5-015.minatec.grenoble-inp.fr	147.171.46.15	Liteon Technology Corporation	50:5B:C2:84:8E:EF
	eduroam5-016.minatec.grenoble-inp.fr	147.171.46.16		AC:67:5D:60:FF:DB
	eduroam5-017.minatec.grenoble-inp.fr	147.171.46.17		BA:11:D8:21:CE:C9
	eduroam5-018.minatec.grenoble-inp.fr	147.171.46.18		1E:F4:2E:5C:A4:EE

Figure 3: L'interface du logiciel Advanced IP Scanner, avec les listes des appareils connectés à la box internet.

Méthode 2 pour obtenir l'IP du Raspberry pi :

Connectez le Raspberry pi à un écran, passez la souris sur l'icône du wifi et relever l'adresse IP.

Ensuite, ouvrir le logiciel Putty, entrer l'adresse IP du Raspberry obtenue auparavant, laisser le port 22 et le type de connexion sur SSH puis appuyer sur entrer. Une invite de commande va s'ouvrir. Se connecter grâce aux identifiants ID : pi et au mot de passe : raspberry. Puis copier/coller

wget -O- https://raw.githubusercontent.com/Jeedom/core/stable/install/install.sh | sudo bash

Dans l'invite de commande et appuyer sur entrer et l'installation se fait toute seule il suffit d'attendre.

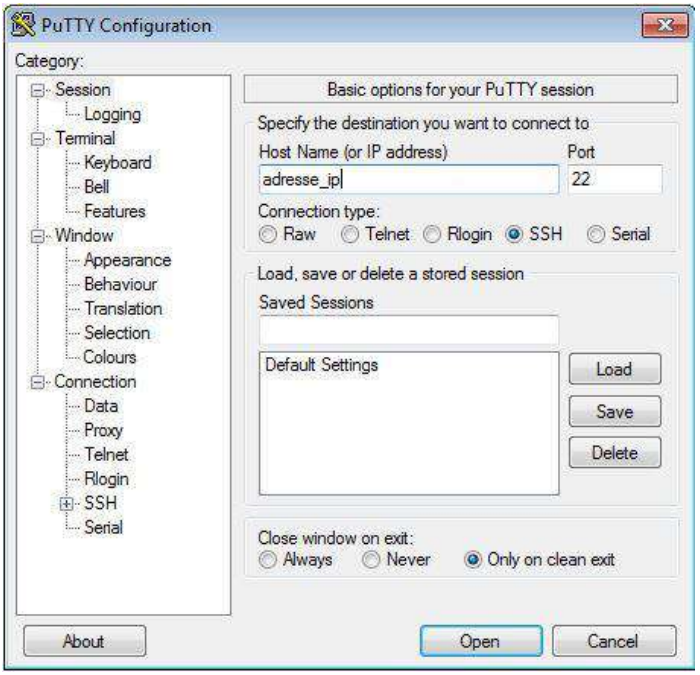


Figure 4: L'interface du logiciel Putty.

Une fois l'installation terminée, redémarrer le Raspberry à l'aide de la commande *sudo shutdown -r now*

Maintenant il faut ouvrir l'interface Jeedom sur un navigateur internet. Pour ce faire, entrer l'IP du Raspberry (obtenue plus tôt) dans le moteur de recherche. La fenêtre suivante s'ouvre peut être trouvée dans figure 5, ci-dessous.

Figure 5: L'interface de Jeedom pour se connecter.

Le nom d'utilisateur est **Admin** et le mot de passe est **admin**.

III - Installation du plugin Mosquitto (MQTT) sur Jeedom

Afin de pouvoir accéder aux données recueillies par le Winky, il faut installer le plugin MQTT sur Jeedom.
La première chose à faire est de se créer un compte Jeedom Market. Pour ce faire, aller sur le site https://market.jeedom.com/index.php?v=d&p=register
Une fois le compte créé, se connecter à l'interface Jeedom de la Raspberry et la lier au compte Jeedom Market : Réglages → Système → Configuration → Mises à jour → Market et entrez votre nom d'utilisateur et votre mot de passe pour votre compte Jeedom Market.
Ensuite, aller dans Plugins → Gestion de plugins → Market → puis rechercher jMQTT et l'installer.

Aller dans la configuration du plugin et à côté de Action, cliquer sur Activer. Dans l'interface du plugin, cliquer sur Ajouter un broker, le nommer et configurer comme ci-dessous. Enfin, sauvegarder cette configuration.

Figure 6: Dans la configuration du plugin de Jeedom. Nommer et configurer comme ci-dessus.

IV - Installation du Winky

La première étape est de programmer le Winky. Pour ce faire, télécharger l'IDE Arduino :
https://arduino.fr.softonic.com/
Copier et coller la dernière version du code suivant :
https://gricad-gitlab.univ-grenoble-alpes.fr/ferrarij/winky/-/tree/main/WinKy_Firmware
Remplacer les différents identifiants, mots de passe et IP dans le code par les vrais.
Il faut ensuite transmettre ce code sur le Winky :

- Tout en maintenant appuyer le bouton jaune du connecteur, le brancher en USB à l'ordinateur (vous pouvez ensuite relacher le bouton) et observer le Port supplémentaire qui apparaît, ce sera celui à utiliser
- Dans l'IDE Arduino, Outils →Type de carte → ESP8266 Boards → Generic ESP8266 Module

Si vous ne le trouver pas directement :
https://ambimod.jimdofree.com/2017/01/26/tuto-comment-programmer-un-esp-01-et-l-utiliser-%C3%A0-la-place-d-un-nodemcu/

Puis cliquer sur outils → gérer les bibliothèques → installer "pubsubclient by Nick O'leary"

- Cliquer sur Téléverser
- Attendre que le code ait été transmis puis déconnecter le port USB

Il ne suffit plus que de brancher le Winky à la prise TIC du compteur Linky et attendre qu'il s'affiche sur Jeedom pour pouvoir récupérer les données !

Annexe 2: Fiche explicative du code *Récupération des données RTE*

Pour rappel, ce code permet de récupérer la dernière valeur de la consommation nationale auprès du distributeur d'électricité français (RTE). Néanmoins, si l'on veut récupérer une autre donnée (production d'électricité des différentes filières de production par exemple), notons qu'il faudra simplement changer une ligne du code. Cette ligne modifiable est signalée par un commentaire.
Le script est construit avec plusieurs fonctions, de manière à ce qu'il suffise de les appeler successivement, en fin de script, pour effectuer la récupération de données souhaitée.

Expliquons rapidement la genèse du code et la forme sous laquelle se présentent les données de RTE.
Les différentes données publiques de RTE (production par filière, consommation, export…) sont téléchargeables à l'adresse suivante : « https://www.rte-france.com/eco2mix/la-consommation-delectricite-en-france » (se rendre en fin de page et ouvrir « télécharger nos données ». Une fois sur la page, si l'on veut effectuer l'action manuellement, il faudra cliquer sur la mention « En-cours mensuel temps réel ».

Les fonctions « initialisation » et « télécharger » permettent d'effectuer automatiquement ces actions. Plus précisément, la fonction « initialisation » prépare le chemin d'accès suivant lequel nous allons stocker les données dans l'ordinateur tandis que la fonction « télécharger » se charge de suivre l'URL de téléchargement des données.

Un fois cette action effectuée, un dossier zippé se charge sur la machine ayant envoyé la requête. La fonction « dezip » nous permet simplement, en utilisant le module python « zipfile », de deziper le fichier pour que celui-ci soit lisible par l'ordinateur.

Le fichier ainsi obtenu contient les valeurs regroupées dans un grand tableau Excel. Ce tableau comprend environ 40 colonnes (une par type de domaine) et 7000 lignes (mais ce chiffre varie constamment). Les données sont réactualisées chaque 15 minutes et sont celles des 30 derniers jours.
Un traitement est ici nécessaire (réalisé par la fonction « recup_xls ») car le tableur est au format xls, ce qui correspond à l'ancien format de Excel, non lisible par l'ordinateur. Nous avons donc décidé d'opter pour une ouverture du fichier sous forme binaire (exploitable par la machine contrairement au format xls), puis de le transformer en chaîne de caractères et enfin d'en extraire les valeurs une à une pour les placer dans une liste. Plus précisément, cette liste est une « liste de liste » dont chaque « sous liste » correspond à une ligne du tableau Excel.

Finalement, la fonction « recup_valeur_conso » permet de récupérer la valeur de la dernière consommation nationale. Celle-ci est la dernière valeur non nulle de la colonne. Ainsi, pour la récupérer, il suffit de récupérer l'indice du premier 0 et de prendre le précédent. Si l'on veut récupérer une autre valeur, par exemple la dernière valeur de production de la filière éolienne, il suffit simplement de changer l'indice correspondant à la colonne dans laquelle sont rangées les données souhaitées. Pour connaître l'indice de la colonne dont on peut extraire la dernière donnée il suffit de télécharger manuellement le fichier et de l'ouvrir. La ligne dans laquelle le changement d'indice est à effectuer est indiquée par un commentaire.

La fonction « recup_conso » fonctionne exactement sur le même principe elle permet simplement de récupérer toutes les valeurs de la colonne « prévisions de consommation » sur les 24 dernières heures.

On peut réutiliser cette fonction pour extraire une série de données. Dans ce cas, il faudra simplement changer l'indice de la colonne considérée ainsi que le « 96 » visible dans le code. En effet, ce nombre permet de récupérer toutes les valeurs des 24 dernières heures (avec une valeur toutes les 15 minutes il y a bien 15 valeur à récupérer par jour). À titre d'exemple, si l'on veut récupérer des données pour la semaine passée il faudra remplacer 96 par 672 (96x7).

La fonction « afficher_conso_reelle_vs_prevision » est simplement une fonction d'affichage.

Les quelques dernières lignes du script permettant de lancer les différentes fonctions. Normalement les chemins d'accès pour stocker les différents fichiers n'ont pas besoin d'être modifiés. Grâce au module « time » de python, toutes les fonctions sont réexécutées toutes les 15 minutes.

Annexe 3: Fiche explicative du code *Envoi*

Objectif :
Ce code permet la communication d'informations à l'utilisateur. Cette communication peut se faire soit par mail, soit par SMS, soit par notification.

Fonctionnement :

Envoi_mail

Pour effectuer l'envoi de mail, on passe par le serveur SMTP qui est un protocole de relais qui permet d'envoyer un mail sans avoir à passer par un navigateur internet.

Il faut ensuite entrer une adresse mail et son mot de passe qui servira à envoyer les mails. Dans le cadre de notre projet, nous avons créé une adresse mail qui sera spécifique à cet usage. Puis, entrer l'adresse mail du destinataire.

Par l'importation du module *email.mime*, nous avons obtenu la fonction *MIMEMultipart* qui permet la création d'un email et la fonction *MIMEText* qui permet la création du message à associer à l'email. A cet email, on ajoute le sujet et le texte, qui sont les entrées de la fonction, à l'email qui a été créé.

Puis cet email est envoyé via le serveur SMTP à l'adresse mail du destinataire.

Envoi_notification

Pour effectuer l'envoi de notification, on utilise une application appelée PushBullet. C'est une application Android à laquelle il faut se connecter avec un compte Google. Son installation et son utilisation sont détaillées dans le document *Tutoriel installation et utilisation Pushbullet*.

Le code de cette fonction est converti de l'interface de commande *curl*. Pour effectuer la requête, il nécessite en en-tête, dans la partie *headers*, la clé d'accès (*Access Token*). Ensuite, la partie *json_data* correspond aux données qui vont être transmises avec la requête, sous le format de données *json* mais converti sur Python. Dans cette partie, il faut fournir le corps et le titre du message, qui sont les entrées de la fonction, le 'type' du message qui indique à l'application PushBullet que la requête est une notification ainsi que l'identifiant de l'appareil (*device_iden*) qui va recevoir la notification. C'est l'importation du module *requests* qui permet l'envoi des requêtes à l'application.

Pour obtenir la clé d'accès et l'identifiant de l'appareil, se référer au *Tutoriel installation et utilisation Pushbullet.*

Envoi_SMS

L'envoi de SMS se fait globalement selon le même principe que l'envoi de notification. Cependant, les données contenues dans *json_data* sont différentes. En effet, pour envoyer un SMS, il faut fournir l'identifiant de l'appareil qui va servir à envoyer le SMS, le numéro de téléphone du destinataire et le message à envoyer (qui est l'entrée de la fonction).

Pour plus d'informations sur l'envoi de requêtes à PushBullet, se référer à l'API de l'application : https://docs.pushbullet.com/#pushbullet-api

Annexe 4: Tutoriel installation et utilisation Pushbullet

Pushbullet est une application web ainsi qu'une application disponible sur Android. Elle permet l'envoi de notification directement sur l'application mobile et l'envoi de SMS sur n'importe quel téléphone portable. Le tutoriel se compose de 4 étapes:

- Etape 1: Installer l'application Pushbullet
- Etape 2: Clé d'accès
- Etape 3: Obtenir l'identifiant de l'appareil
- Etape 4: Codes

Etape 1: Installer l'application Pushbullet

Installer l'application Pushbullet sur le téléphone portable (uniquement Android) :
https://play.google.com/store/apps/details?id=com.pushbullet.android
Et se connecter avec un compte google.

Etape 2: Clé d'accès

Pour obtenir la clé d'accès, il faut :

- Aller sur le site internet Pushbullet : https://www.pushbullet.com/
- Se connecter avec le même compte google que l'application
- Aller dans *Settings*
- Cliquer sur *Create access token*

Bien noter cette clé d'accès, elle sera utile pour toutes les requêtes.

Etape 3: Obtenir l'identifiant de l'appareil

Il faut maintenant obtenir l'identifiant de l'appareil qui va servir à envoyer les SMS.
Pour ce faire, il suffit de lancer ce code Python :

```
import requests
import json

headers = {
'Access-Token': '<clé d'accès>',
}
response = requests.get('https://api.pushbullet.com/v2/devices', headers=headers)

print(json.loads(response.content.decode("UTF-8"))["devices"][0]["iden"])
```

En remplaçant *<clé d'accès>* par votre clé d'accès obtenue à l'étape 3.
Bien noter cet identifiant, sera utile pour l'envoi de SMS.

Etape 4: Codes

A l'aide de la clé d'accès et de l'identifiant de l'appareil, il est désormais possible d'utiliser les fonctions *envoi_sms* et *envoi_notification*. A noter que l'envoi ne se fait que lorsque l'application mobile est ouverte.

La fonction *envoi_sms* prend en entrée une chaîne de caractères qui sera le message à envoyer, il suffit de changer le numéro de téléphone du destinataire.

```
def envoi_sms(message):
    headers = {
        'Access-Token': 'o.QAZShPXZWBOTr331Gb77kY26hXdiLrPJ', #clé d'accès à adapter à l'appareil envoyeur
        'Content-Type': 'application/json',
    }

    json_data = {
        'data': {
            "target_device_iden":'ujwzqB8xKvYsjylcFKpCg0', #identifiant de l'appareil qui envoie le sms

            'addresses': ['+33685079501'], #numéro du téléphone receveur, à changer en fonction du destinataire
            'message': message, #message à envoyer, à changer en fonction de l'alerte
        },
    }

    #code qui communique à l'appli le type de message à envoyer, ne pas changer
    response = requests.post('https://api.pushbullet.com/v2/texts', headers=headers, json=json_data)
```

Figure 1: Une image d'une partie du code Envois. Ici le numéro de téléphone peut être changé dans 'adresses'..

La fonction *envoi_notification* prend en entrée deux chaînes de caractères qui seront le titre du message et le message à envoyer, il suffit de changer l'identifiant de l'appareil receveur.

```
def envoi_notification (message,titre):
    headers = {
        'Access-Token': 'o.QAZShPXZWBOTr331Gb77kY26hXdiLrPJ', #clé d'accès à adapter à l'appareil envoyeur
        'Content-Type': 'application/json',
        }

    json_data = {
        'body': message, #corps du message
        'title': titre, #titre du message
        'type': 'note', #type de message à envoyer, ne pas changer
        #identifiant de l'appareil qui recoit la notification, à changer en fonction du destinataire
        'device_iden':'ujwzqB8xKvYsjylcFKpCg0'
        }
    #code qui communique à l'appli le type de message à envoyer, ne pas changer
    response=requests.post('https://api.pushbullet.com/v2/pushes', headers=headers, json=json_data)
```

Figure 2: Une image d'une partie du code nommé Envois. Ici le l'identifiant de l'appareil receveur peut être changé dans ' device_iden'.

Annexe 5: Fiche explicative du code *Calculs*

Le code calcul.py est utile pour toutes personnes possédant une installation Winky reliée à une base de données influxDB.

Objective du code :

Ce code a pour objectif d'aider le consommateur à mieux connaître sa consommation. Les consommations en puissance et énergie électrique moyennes sont calculées pour une période choisie par l'utilisateur. Une estimation du coût de cette consommation sur la même période, ainsi qu'une comparaison avec la période précédente sont réalisées.
De cette manière, le foyer sera informé de manière globale sur sa consommation ainsi que son évolution. Dans notre cas, nous avons choisi une période de 1 semaine car c'est un pas de temps suffisamment long pour être révélateur de la consommation. Mais il n'est pas trop long, afin que l'utilisateur puisse agir sur sa consommation si elle est en augmentation.

Fonctionnement :

Ce code est constitué de plusieurs parties:

- Les "outils" nécessaires

Pour faire fonctionner ce code, il est nécessaire d'avoir installé au moins une fois le module 'influxdb'.
De nombreux autres modules Python sont à importer.
Les codes 'data_recovery_csv_creation' et 'envoi.py' sont nécessaires respectivement, pour pouvoir traiter les données de consommation et pour pouvoir informer l'utilisateur. Le détail de ses fonctions sera donné dans la suite de ces explications brièvement mais sont disponibles dans les fiches détaillées de chaque code.

remarque: ces deux codes doivent être télécharger au même endroit que le code actuel! I
Ils doivent être également adaptés à votre situation.

La fonction 'global_mean(csv_file)' est très simple: en entrée, elle prend le nom d'un fichier csv qui contient une colonne 'value' (si votre colonne a un nom différent, il est possible de changer cette ligne "csv_file['value']"). Grâce à une fonction python, la moyenne de la colonne entière est réalisée très facilement. Le paramètre de sortie de cette fonction est finalement la valeur moyenne de la colonne 'value' entière du fichier 'csv_file'.

La fonction 'tendency (actual, previous)' , à en paramètre d'entrée, deux floats. Elle permet de les comparer et de créer un message adapté au résultat ainsi que le pourcentage d'évolution. Le message est découpé en deux parties pour des questions d'affichage. Cette fonction à donc pour sortie:un message en deux partie: 'tendence_energie' et 'commentaire' ainsi que le rapport des variables: rapport_consommations

- Les variables de temps

Delta_time: cette variable permet de choisir la durée de la période sur laquelle les calculs seront faits. Il suffit d'indiquer le nombre de jours, minutes, secondes souhaité.

La variable 'now' réfère ici à la date et heure actuelle sous le même format que 'delta_time'.

end_previous_period= now-delta_time: la fin de la période en cours est le début de la période actuelle. Par exemple: [Actuelle_début]=07/05, [Actuelle_fin]=now, et [Passé_fin]=[Actuelle_début]=07/05, [Passé_début]=01/05 pour une période d' une semaine.

- Puissance

Dans un premier temps, on s'intéreste à la période actuelle de [now] à [now-delta_time], on va donc utiliser la fonction 'csv_creation' du fichier 'Data_recovery_and_CSV_creation' afin de récuperer les données correspondante à cette variable, et ce, de 'now' à 'now-delta_time:

Data_recovery_and_CSV_creation.csv_creation(now, delta_time, "Actual_Power.csv", 1503)

Une fois les données récupérées, il suffit de faire la moyenne de la colonne grâce à la fonction 'global_mean'.
Pour la période précédante, on fait de même en changeant les paramètres d'entrée de la fonction 'csv_creation'. La fin de la période précédente étant le début de la période actuelle. La durée de la période reste la même:

Data_recovery_and_CSV_creation.csv_creation(end_previous_period, delta_time, "Past_Power.csv", 1503)

- Energie

Pour l'énergie on récupérer les données actuelles et précédante de la même manière que pour la puissance en adaptant les noms de fichier et la valeur des index. Le raisonnement sur les durées reste le même.
Une fois les données récupérées, il faut faire une soustraction entre la donnée la plus récente et la plus ancienne du fichier (en effet, les kWh s'ajoutent sans jamais se remettre à zéro). Par exemple pour l'energye sur la période en cours:

delta_actual_energy = (df_actual_energy.iloc[-1,1]- df_actual_energy.iloc[0,1])/1000

Le coût de la consommation est calculé avec le prix d'un kWh moyen en France (cette valeur doit être mise à jour). Il suffit de multiplier le nombre de kWh consommé par ce prix.

En faisant appel à la fonction 'tendency' on connaît l'évolution de la consommation et le pourcentage de cette dernière.

- Envoie

Pour des question d'affichage toutes les variables sont arrondies de la manière suivante:
delta_actual_energy = Decimal(delta_actual_energy)
roundenergy = delta_actual_energy.quantize(Decimal('.1'), rounding=ROUND_HALF_UP)
De même, pour des questions d'affichage la variable 'delta_time' est remise en forme.

Le message est construit est envoyer ensuite grâce à la fonction 'envoi_sms'.

- Boucle temporelle

On réalise une boucle temporelle qui s'exécute à l'infini si aucune erreur est détectée:

```
k = 0
temp = time.time()
while True:
   try:
      if time.time() - temp > k*60*60*24*7: #Choose when the code will launch itself.
         k+=1

CODE #ici est présent la majorité du code évoquée ci-dessus.

      except:
         print("erreur")
```

De cette manière on pourra téléverser le code dans la raspberryPi et il suffira d'ajouter un service pour que le code soit autonome.

Annexe 6: Fiche explicative du code *CO2équivalent*

Introduction

Ce code utilise les informations du Winky sur la consommation de la résidence d'un certain temps et les valeurs de l'éCO2mix de RTE. Le script calcule l'équivalent CO_2 qui provient de la consommation. Pour voir comment la récupération des données de RTE a été faite, voir la description de l'annexe 2.

Pour faire fonctionner ce code, il est nécessaire d'avoir installé au moins une fois le module 'influxdb'.
De nombreux autres modules Python sont à importer.
Les codes 'data_recovery_csv_creation' et 'envoi.py' sont nécessaires respectivement, pour pouvoir traiter les données de consommation et pour pouvoir informer l'utilisateur. Le détail de ses fonctions sera donné dans la suite de ces explications brièvement mais sont disponibles dans les fiches détaillées de chaque code.

Objectif du code

Ce code a été créé pour calculer le CO_2 correspondant à l'énergie consommée par l'utilisateur pendant une semaine. L'information et après envoyer pour l'utilisateur par un SMS. De cette manière, le foyer sera informé de l'équivalent de sa consommation chaque semaine. Dans notre cas, nous avons choisi une période d' une semaine, car c'est un pas de temps suffisamment long pour se souvenir de ce qu'on a fait. Le temps n'est pas trop long, afin que l'utilisateur puisse agir sur sa consommation, s'il trouve qu'il y a contribué à trop d'émissions.

Comment il fonctionne?

Ce code est constitué de trois parties principal:

- La demande à RTE pour d'obtenir le l'équivalent de CO2 pour la dernière semaine.
- Le calcul de l'énergie .
- Une boucle temporelle

Récupération des donnée de CO_2

Grâce au code permettant de récupérer les données de RTE nous récupérons les données concernant le CO_2.
Le code spécifie également la colonne 17 pour récupérer le CO_2, ce chiffre fait référence au fait que c'est précisément la colonne 17 du fichier RTE qui contient les équivalents CO_2 pour la consommation. A partir de cela, nous construisons un vecteur '*CO2*' à 672 valeurs (car il y a 672 fois 15 minutes dans une semaine). Une composante du vecteur correspond à une équivalence en CO_2 d'un kilowattheure sur 15 minutes, la première valeur du vecteur étant la plus ancienne. Dans cet exemple nous mentionnons un vecteur à 672 valeurs car cela correspond à une semaine mais si vous souhaitez prendre un autre pas de temps, cette valeur se mettra à jour automatiquement dès lors que la variable *delta_time* est modifiée.

Le calcul de l'énergie

Nous calculerons pour une durée donnée *(delta_time*) , l'énergie consommée toutes les 15 minutes, grâce à une boucle 'for' qui s'exécutera le nombre de fois ou il y a 15 min dans la période (*number_of_15min*). Cette boucle permet de créer un CSV et d'extraire l'énergie consommée sur une période de 15 minutes. A la fin de la boucle, nous effectuons un décalage temporaire pour extraire les 15 minutes précédentes et ainsi de suite.

Nous aurons donc un nombre de valeurs de consommation égale à *number_of_15min*. Chaque valeur sera stockée dans le vecteur '*energy_vector_15*' pour lequel nous inversons l'ordre des éléments (afin d'avoir une cohérence entre ce vecteur et le précédent). Dans l'exemple d'une période de 7jours, ce vecteur aura 672 valeurs.

Le nombre de 15 minutes dans une semaine est calculé avec:

```
numbers_of_15min = int(int(delta_time.total_seconds())/int(dt15min.total_seconds()))
```

pour avoir un numéro en int.
Un vecteur vide est créé pour contenir toutes les valeur:

```
energy_vector_15 = [0] * numbers_of_15min
```

Le CO_2 final de la période est calculé en retirant la somme de la multiplication de deux vector, energy_vector_15 et CO2.

Une boucle temporelle

On réalise une boucle temporelle qui s'exécute à l'infini si aucune erreur est détectée:

```
k = 0
temp = time.time()
while True:
   try:
      if time.time() - temp > k*60*60*24*7: #Choose when the code will launch itself.
         k+=1

#####################
CODE...
#####################
     except:
         print("erreur")
```

De cette manière on pourra téléverser le code dans la raspberryPi et il suffira d'ajouter un service pour que le code soit autonome.

Annexe 7: Fiche explicative du code *Routine*

Contexte :

Le principe de ce code est de créer un courbe de consommation type, basée sur les données des 7 derniers jours. On veut ainsi pouvoir détecter les consommations suspectes en les comparant à la journée type.

Code:

On commence par importer tous les modules nécessaires au lancement du code.
La fonction *envoi_sms* permet d'envoyer le message que l'on souhaite, le message est l'entrée de la fonction.

La fonction *secondes_to_date* prend en entrée un temps depuis epoch (le temps depuis la création du temps informatique) en secondes et une chaine de caractère (format de la date voulut). Avec les entrées ainsi définies, la fonction renvoie le chiffre de l'heure actuelle.

Le code *recup_1_minute* récupère les valeurs de consommation des *nb_minutes* dernières minutes (entrée de la fonction), et renvoie en sortie la valeur moyenne de la consommation sur ces dernières minutes et la date moyenne de prise de ces données.

recup_7_jours récupère les données des 7 derniers jours pour les mettre dans un CSV qui pourra ensuite être manipulé. Il prend en entrée le *delta_time* de 7 jours.

La fonction *date_to_seconde*s est la fonction inverse de la fonction *secondes_to_date,* elle prend en entrée une date et le format de celle-ci pour donner en sortie le temps depuis epoch de cette date.

recup_valeur_bonne_date prend en entré la liste des temps d'une journée découpée en pas de temps et un temps qu'il va falloir classer dans cette liste. Le temps qui doit être classé peut correspondre à une date d'une semaine avant (car calcul de la routine des 7 derniers jours). Lorsque l'on trouve ou est ce que ce temps doit être classé dans la liste Lx on renvoit son indice.

La fonction *recup_liste_moyennes* a pour but de renvoyer la liste des données sur 7 jours, regroupée en une journée et classée par heure. Elle prend donc en entrée le pas de temps, qui sera la précision du calcul de la routine. Elle crée une liste Lx correspondant à la liste des temps d'une journée, écartée du pas, et une liste Ly qui, grâce à la fonction *recup_valeur_bonne_date*, classe les données de consommation en fonction de leur heure dans la journée.

La fonction *calcul_nuit* correspond à toutes les actions qui vont devoir être effectuées pour créer la routine. Elle prend en entrée le pas de temps, elle récupère Lx et Ly grâce à la fonction *recup_liste_moyennes* puis, pour chaque terme de Ly, calcule la moyenne des données de chaque instant. Pour finir elle renvoie *Lx et Ly_bis* qui correspond à la liste des moyennes.

Pour le cœur du programme, on commence par initialiser la routine en appelant *calcul_nuit*. Une fois cela fait, on pourra rentrer dans la boucle infinie et dans le try (permettant de sécuriser le code une fois sur le raspberry). On vérifie s'il est 3h du matin; si oui, on lance le calcul de la routine. Sinon on commence par vérifier si une notification a été envoyée il n'y a pas longtemps. Si ce n'est pas le cas, on récupère les dernières valeurs de consommation, si elles sont 3 fois supérieur à la consommation « normale » (la consommation calculée dans la routine) on envoie un sms à l'utilisateur pour l'informer de ce problème.

Annexe 8 : Le réseau de neurones LSTM : Définition et utilisation

Un algorithme d'apprentissage automatique est un algorithme constitué de systèmes qui apprennent ou améliorent leurs performances, en fonction des données traitées. Un réseau neuronal artificiel est un algorithme d'apprentissage automatique qui tente d'imiter le système neuronal biologique.
Ces algorithmes peuvent parfois résoudre des problèmes difficiles plus rapidement que des méthodes informatiques classiques, au prix d'un entraînement parfois long. Le réseau de neurones artificiel est constitué d'un nombre important de processeurs, représentant les neurones, opérants en parallèle et organisés en couches de la manière suivante :

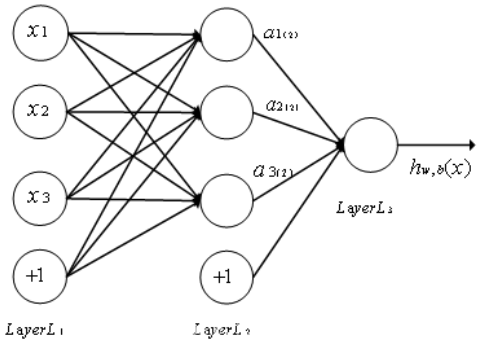


Figure 1: Schéma d'un réseau de neurones

Le Long Short Term Memory (LSTM) tire son nom du fait qu'il prend en compte non seulement des données nouvelles en entrée (Short memory) mais également des données anciennes jugées utiles ou nécessaires (Long memory) pour proposer la meilleure sortie possible. Cette architecture est constituée de plusieurs couches qui contiennent elles-mêmes plusieurs centaines de cellules.
Chaque cellule est composeé de trois portes : une porte d'entrée, une porte d'oubli et une porte de sortie. La porte d'entrée reçoit des informations et décide quelles nouvelles informations doivent être ajoutées à l'entrée. La porte d'oubli trie entre les informations les plus utiles et celles qui peuvent être oubliées. La porte de sortie choisie quelle information, de toutes celles reçues, doit constituer la sortie, afin que le neurone suivant reprend les informations importantes. Voici une illustration d'un neurone LSTM. Enfin, l'état du neurone constitue les données stockées dans la mémoire de ce dernier. Le réseau LSTM va donc utiliser une partie des données récoltées sur une certaine période pour s'entraîner, et une autre partie pour tester la prédiction.

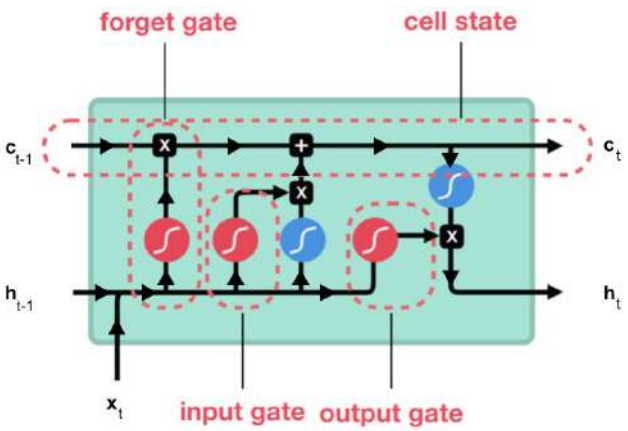


Figure 2: Une illustration d'un neurone LSTM.

On retrouve dans cette cellule un ensemble d'opérations qu'il est possible d'effectuer. A savoir la fonction sigmoïde (en rouge) et tanh (en bleue), des produits de Hadamard et des additions matricielles.

[définition (produit de Hadamard) : En syntaxe Python
Soient A = [[a, b], [c, d]] et B = [[e, f], [g, h]] deux matrices, alors le produit de Hadamard est la matrice Hadamard(A, B) = [[a*e, b*f], [c*g, d*h]].
]

A chaque neurone est attribué 4 poids qui pondèrent les valeurs d'entrées de chacunes des portes et de l'état de cellule. Ces poids sont initialisés au hasard et corrigés à chaque epoch (itération d'entraînement) en comparant la valeur prédite à la valeur réelle à l'aide d'une fonction de coût J.
Le résultat de cette fonction montre la précision de prédiction. En effet, plus elle est petite, plus la prédiction est proche de la valeur réelle.
Les poids de chaque neurone sont déterminés de la manière suivante, en utilisant la méthode de la descente du gradient :

$$\omega_{t+1} = \omega - \alpha\, Grad(J)$$

Avec ω le poids et α la vitesse d'apprentissage du réseau.
Il est nécessaire de normaliser les données afin d'éviter que le gradient n'explose.
Un réseau de neurones est constitué de paramètres, propres à lui-même, et d'hyper-paramètres, externes au réseau, qui doivent être définis. On note notamment : la proportion de données pour l'entraînement et le test, la durée qu'on utilise pour faire la prochaine prédiction, le nombre de couche, le nombre de neurones dans chaque couche, le nombre d'*epoch* d'entraînement, la taille du batch pour un entraînement, la fonction de coût et l'optimiseur.

Un optimiseur est une méthode utilisée dans le but de changer les attributs du réseau neuronal comme les poids, afin de réduire les coûts et rendre les prédictions plus précises. Il existe plusieurs types d'optimiseurs, mais le plus adapté pour un réseau LSTM reste le RMSProp parce qu'il accélère la convergence et oriente à un gradient qui fait descendre plus rapidement pour une série temporelle.

Annexe 9: Fiche explicative des codes d'intelligence artificielle

Nous avons construit deux algorithmes basés sur l'intelligence artificielle LSTM_Production. Le premier algorithme permet de prédire la production d'électricité sur le territoire national grâce en utilisant la colonne dédiée aux consommations sur le document Excel RTE. En effet, les courbes de production et de consommation nationales sont similaires. Le second algorithme permet de prédire la proportion d'énergies renouvelables produites dans le mix énergétique total de production en calculant le ratio entre la production totale d'énergie renouvelable et la production totale d'énergies.
Ces deux algorithmes sont construits de manière similaire grâce à un réseau de neurones de type LSTM et comportent deux majeures parties. La seule différence correspond aux données récupérées. Les informations suivantes sont donc valables pour les deux codes.
Au début du code de prédiction, trois types de modules nécessaires sont importés, le premier permet l'importation des données, le second permet de gérer la notion du temps et enfin le dernier permet de construire le réseau neuronal.
La première partie du code sert à télécharger, dézipper et transformer les données en format préférable en Python. La deuxième est pour gérer la notion de temps, et la troisième sert à construire le réseau de neurones.
Les fonctions de traitement de données (*get_data*, *unzip_file*, *recup_xls*) sont presque identiques à celles mentionnées avant.
La fonction *recup_valeur_conso_par_heure* sert à retirer les données de production d'électricité pendant une période voulue. Nous récupérons les données pendant les dix derniers jours en précisant l'heure de début et fin. De plus, puisque nous faisons deux prédictions par jour, on juge le moment actuel afin de distinguer deux cas. Les données sont retirées par heure étant donné que la prédiction sera faite pour les 12 prochaines heures. Si nous prenons les données par 15 minutes, le réseau ne pourrait pas prédire les 48 prochaines données précisément.

Après que les données sont transformées en format Pandas pour plus de visibilité, elles sont mises entre 0 et 1 par normalisation:

df = (df - df.min()) / (df.max() - df.min())

Dans la seconde partie, on construit un réseau de neurones de type Long Short Term Memory (LSTM). Voici les hyperparamètres que nous devons fixer pour le réseau :

- La proportion de données pour l'entraînement et le test : 60% et 40%
- Le nombre d'heures que nous utilisons pour faire la prochaine prédiction : 12
- Le nombre de couches : 2
- Le nombre de neurones dans chaque couche : 400
- Le nombre d'*époch* d'entraînement : 400
- La taille du batch pour un entraînement : 64
- Fonction de coût : MSE définie comme suit :
  $J = \frac{1}{2}(valeur\ prédite - valeur\ réelle)^2$
- L'optimiseur pour le renouvellement des poids : RMSprop

Nous utilisons l'optimiseur RMSprop car c'est le plus adapté pour un réseau de neurones de type LSTM. Après l'entraînement, si le coût converge et est assez petit, nous pouvons terminer l'entraînement pour gagner en rapidité. Nous traçons ensuite la courbe de la prédiction et des données

réelles sur le même graphique, et les deux sont proches (voir figure dessous) sur une période d'environ 80 heures afin de vérifier les tendances générales.

```
Epoch 499/500
2/2 [==============================] - 1s 502ms/step - loss: 0.0016
Epoch 500/500
2/2 [==============================] - 1s 452ms/step - loss: 0.0011
```

Après l'entraînement, la prédiction est faite en utilisant les douze dernières données. La première valeur prédite est ajoutée dans la liste, et la plus vieille est supprimée, puis nous refaisons cette itération. À la fin nous obtenons une liste de douze valeurs prédites.

Nous appliquons l'interpolation cubique pour retrouver les données en minute. Dans ce cas, la précision est garantie et l'utilisateur obtient un moment quasiment précis. De plus, les valeurs sont "dénormalisées" et nous retirons le maximum et minimum dans la liste. D'ailleurs, il faut bien distinguer le cas quand la prédiction est faite, et donner correctement le temps en format normal. Si c'est le matin nous ajoutons quatre heures de décalage pour l'abscisse, et seize heures pour l'après-midi. Si l'heure dépasse vingt-quatre, nous soustrayons vingt-quatre.

À la fin, la fonction *message* sert à rassembler les données nécessaires en un paragraphe complet. À l'aide de la fonction *envoi_sms*, le message est envoyé à l'utilisateur souhaité avec le bon compte et l'adresse par pushbullet.

Annexe 10: Fiche explicative du code *calcul des pics de consommation*

Avec ce script, nous cherchons à extraire les périodes de la journée durant lesquelles le transporteur d'électricité français (RTE) prévoit une consommation nationale (ici dénommée par « consommation ») élevée. En effet, chaque jour (vers 23h30 environ) RTE fournit une courbe prévisionnelle de la consommation pour le lendemain. Nous pouvons constater que ces prévisions sont extrêmement proches de la réalité dans la grande majorité des cas. L'ambition et la finalité de ce code sont de donner à l'utilisateur des plages horaires durant lesquelles il est intéressant de reporter sa consommation d'énergie pour permettre de diminuer la tension sur le réseau lors des pics de consommation.

Toute la première partie du script est composée de l'ensemble des fonctions permettant de récupérer des valeurs données par RTE. Pour plus de détails voir le document explicatif du script complet.

À l'issue de de code nous avons une liste des valeurs de prévisions pour les 24 prochaines heures, ainsi que l'heure et la date à laquelle sont émises les prévisions.
Si une exploration des variables est faite, on constate que la variable « Date » donne la date non pas du jour mais de la veille. Ceci n'est pas une erreur, et s'explique par le fait que les prévisions sont émises à 23h45 pour le lendemain, la date d'émission de la prévision est donc antidatée.

La fonction « seuil_min_max_conso » constitue le corps du script. C'est elle qui effectue les actions nécessaires à l'obtention des résultats escomptés pour ce code. En entrée, elle prend le fichier de valeur de RTE et un ratio, c'est-à-dire un pourcentage du maximum prévu de consommation tel que l'heure à laquelle la prévision atteint ce ratio soit considérée comme marquant le début d'une période de pointe de consommation. En d'autres termes : lorsque la prévision dépasse (ici) 80 % de la valeur maximale journalières de consommation prévue, l'heure correspondante marque le début d'une période durant laquelle la consommation d'électricité devrait être évitée. De manière analogue, lorsque la prévision passe en deçà de 80 % de la valeur maximale, l'heure correspondante marque la fin de cette période.

En sortie, la fonction retourne les heures de début et fin des périodes de report de consommation, l'heure prévue du pic de consommation, la valeur du pic de consommation prédit ainsi que le seuil qui a été considéré (ratio du pic de consommation).

Concernant le fonctionnement de cette fonction calculatrice :

Lors de l'exécution nous commençons par récupérer les variables retournées par le site de récupération des valeurs de RTE. Nous récupérons le maximum et le minimum de consommation, nous associons à ces extremums l'heure à laquelle ils sont censés apparaître. Pour cela nous avons utilisé une liste avec les heures pour lesquelles nous avons une valeur prédite.
Nous parcourons ensuite la liste de prédictions pour regarder lorsque les valeurs dépassent le seuil fixé. Finalement nous associons une heure à ces moments.

La seconde fonction du script est une fonction « main » qui exécute la fonction précédente ainsi que les fonctions de récupération de données. Nous avons également ajouté la pelle des fonctions

permettant l'envoi d'un SMS à l'utilisateur pour l'avertir des horaires de report de consommation pour la journée à venir.

Les dernières lignes du code permettent l'exécution du script une fois par jour à 7h du matin.

Annexe 11: Tutoriel: Installation des codes dans la RaspberryPi

Pour rendre le système autonome et pour lancer les codes à chaque période de temps, les codes ont été installés dans le RaspberryPi. Dans ce tutoriel, nous décrirons cette installation pour aider les installations futures.

Le tutoriel se compose de 4 parties:

1. Installation du code et des modules correspondants dans VNC Viewer
2. Faire en sorte que le Raspberry appelle automatiquement le script
3. Vérifier que les scripts sont en cours d'exécution
4. Modifier le contenu du script dans le Raspberry PI

Remarque: Cette installation utilise le programme VNC Viewer pour avoir accès au Raspberry.

1. Installation du code et des modules correspondants dans VNC Viewer

Description

Cette partie décrit comment ouvrir une fenêtre interactive du Raspberry Pi sur un ordinateur. Elle est suivie par une indication sur la manière de transférer les fichiers de l'ordinateur vers le Raspberry Pi. Et enfin de l'installation des modules utilisés.

Lorsque cette partie sera achevée, il sera possible d'exécuter votre script sur la Raspberry via le terminal de la visionneuse VNC. Notez que cela n'inclut pas encore l'autonomie du Raspberry.

Implémentation

Tout d'abord, connectez le RaspberryPi au wifi et écrivez l'adresse IP du Raspberry Pi connecté dans VNC Viewer. Utilisez les données de connexion de votre Raspberry Pi pour vous connecter.

(Si vous ne trouvez pas le Raspberry dans la visionneuse VNC, il est possible que le serveur VNC n'ait pas été activé dans le Raspberry Pi. Faites-le d'abord).

Dans VNC Viewer, utilisez la fonction "transfer files" (elle apparaît lorsque vous passez votre souris sur la partie supérieure de l'écran VNC) pour envoyer les codes de votre ordinateur local vers le Raspberry. Créez un dossier avec un nom approprié dans le dossier appelé "pi" (/home/pi).
Attention : Le nom du dossier doit être un mot unique ou lié avec des underscores, ex Project_SMART_ROOM et non Project SMART ROOM

Dans le terminal de la VNC, écrivez "ls" pour obtenir la liste des dossiers dans le Raspberry et donc l'emplacement de votre code. Cela donne quelque chose comme ceci :

```
Bookshelf           Desktop    Downloads  Pictures  Templates
name_of_your_folder Documents  Music      Public    Videos
```

Figure1: L'affichage de la commande "ls".

Utilisez ensuite la commande "cd nom_de_votre_dossier" pour trouver le chemin du dossier. Puis écrivez :

```
sudo python3 name_of_the_file.py
```

Ici, des modules seront demandés, par exemple "ModuleNotFoundError : No module named 'pandas', pour pouvoir exécuter le code. Installez-les (tous les modules) en utilisant la commande sudo pip3 install suivie du nom du module, par exemple :

```
sudo pip3 install pandas
```

Répétez l'opération jusqu'à ce que tous les modules soient installés. L'installation ressemblera à ceci

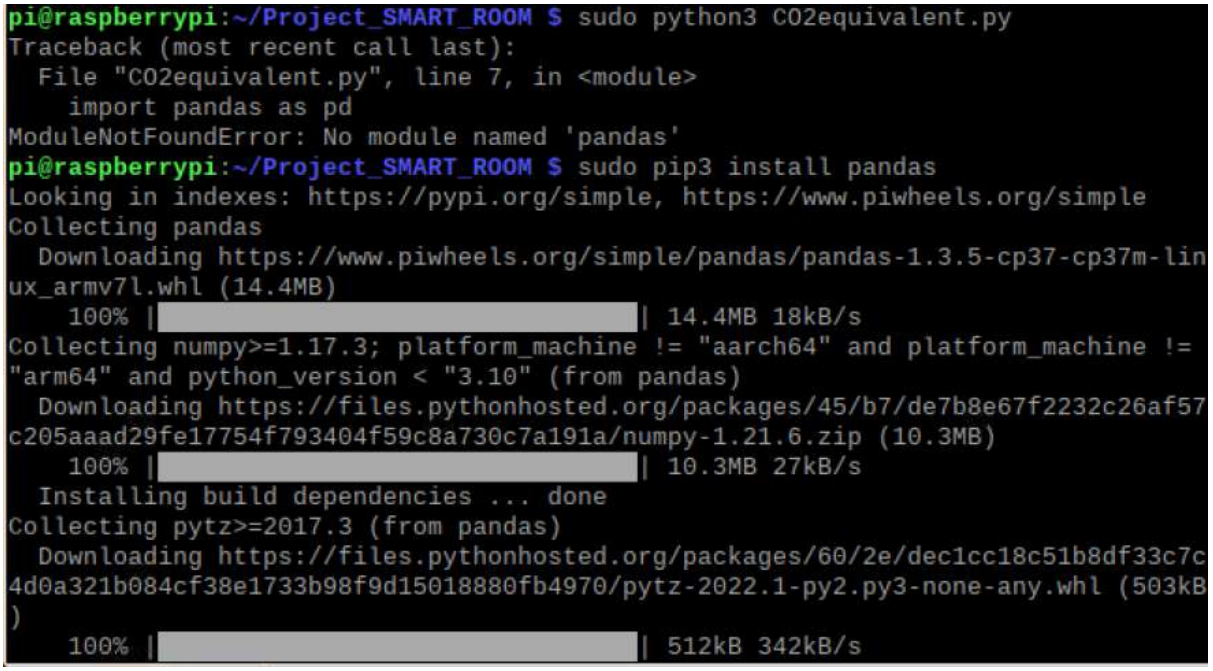


Figure 2: Une image du téléchargement des modules nécessaires.

Notez que, selon la connexion, cette procédure peut prendre un certain temps... Surtout pour les paquets numpy, car ils sont très gros.

Lorsque toutes les installations des modules sont terminées, vous devriez être en mesure d'exécuter votre code en appelant python3 puis votre fichier comme vous l'avez tenté précédemment. Ecrire :

```
sudo python3 name_of_the_file.py.
```

2. Faire en sorte que le Raspberry appelle automatiquement le script

Description

Cette partie sert à rendre le programme autonome. Le code que vous avez fait devrait avoir une sorte de boucle avec des délais pour lancer le script au bon moment. Le code que vous venez d'entrer lancera ce code une fois et ensuite il sera bloqué dans cette boucle.

Remarque: Cette partie doit être refaite pour chaque script qui est censé être lancé. Une suggestion est de faire une sorte de tableau pour le nom du script et le nom du service correspondant, un aperçu utile (mentionné plus bas).
Notation : Les scripts qui sont appelés, et non lancés, n'ont pas besoin de cette procédure.

Implémentation

Ouvrez le terminal dans le VNC Viewer et écrivez ce qui suit, où name est le nom du service correspondant à votre script :

```
cd /lib/systemd/system/
sudo nano Name.service
```

La commande ouvre une fenêtre, écrivez tout le texte suivant dans la fenêtre mais changez le nom du script et le dossier choisi (marqué en gras dans le texte ci-dessous) par le vôtre.

```
[Unit]
Description = *Add description (optional)*
After = multi-user.target

[Service]
Type = simple
ExecStart = /usr/bin/python3 /home/pi/Project_SMART_ROOM/Scriptname.py
Restart = on-abort

[Install]
WantedBy = multi-user.target
```

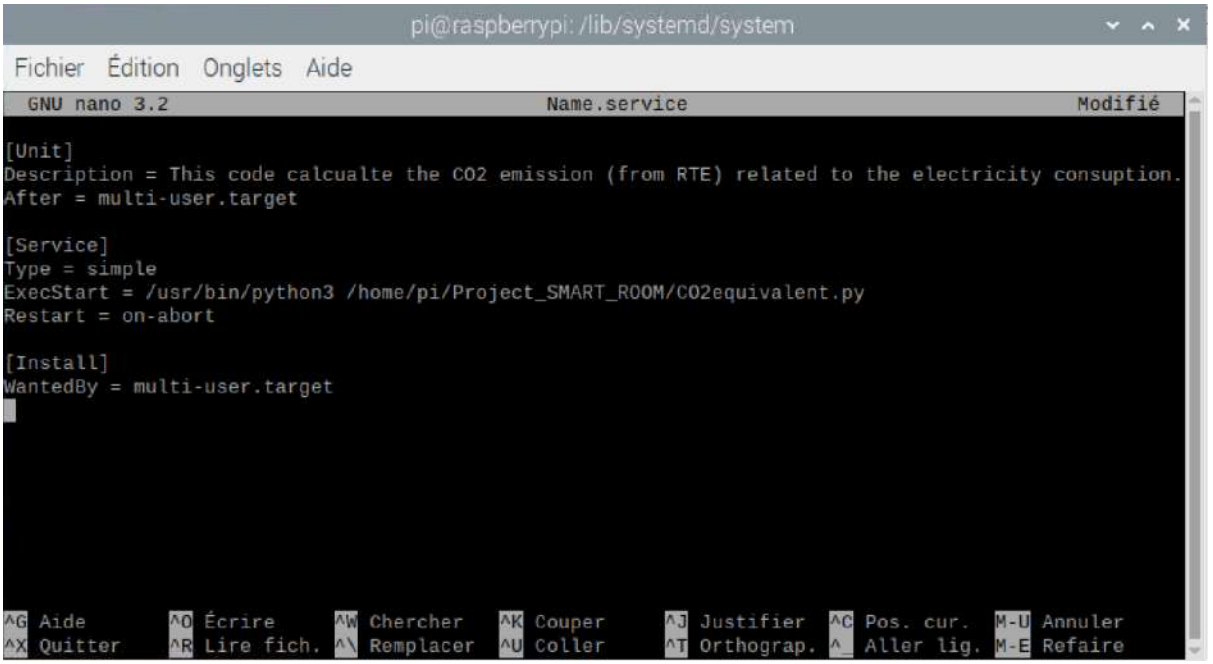


Figure 3: Une image de la fenêtre qui s'ouvre, y compris le texte mentionné.

Ensuite, quittez la fenêtre en appuyant sur ctrl + X puis ctrl + Y pour signifier OUI.
Changez le nom du service de nom.service à un nom approprié. Ici, chaque service est nommé d'après le script qui lui correspond pour faciliter la procédure.

Écrivez les commandes suivantes dans le terminal :

```
sudo chmod 644 /lib/systemd/system/Name.service
sudo chmod +x /home/pi/Project_SMART_ROOM/Scriptname.py
sudo systemctl daemon-reload
sudo systemctl enable Name.service
sudo systemctl start Name.service
```

Name est le nom du dossier que vous venez de créer, choisissez un nom approprié.
Répétez le processus pour chaque script qui doit être lancé.

3. Vérifier que les scripts sont en cours d'exécution

Description

Comme le titre l'indique, cette partie explique comment vérifier que vos scripts s'exécutent.

Implémentation

Dans le terminal, placez-vous dans le dossier de votre projet (en utilisant la commande *cd* et **nom_du_dossier**).
Puis écrivez "ls" pour obtenir la liste du dossier contenant les scripes du projet. Cela ressemble à l'image ci-dessous, figure 4.

Figure 4: Une image de la liste des scriptes en cours d'exécution.

Dans le cas du script en cours d'exécution, le nom de chaque script doit être marqué en vert et en gras, comme présenté dans l'image ci-dessus. Si le script est gris clair, il n'est pas en cours d'exécution.

4. Modifier le contenu du script dans le Raspberry Pi

Description

Cette partie décrit si l'utilisateur veut changer le contenu du script.

Implémentation

Ouvrez le dossier et ensuite le script. Effectuez les modifications et sauvegardez votre progression.
Si le script fait partie de ceux qui doivent être relancés, écrivez ce qui suit pour relancer le script modifié :

```
sudo systemctl restart name.service
```

Si le script ne doit pas être lancé (ex. seulement appelé), il suffit de sauvegarder le script modifié et vous avez terminé.

Si vous voulez ouvrir le datalog du fichier créé, utilisez la commande suivante :

```
sudo nano /home/pi/Project_SMART_ROOM/datalog_name.csv
```

Annexe 12 : Tutoriel fabrication module de téléversement

Ce tutoriel présente la fabrication du module de téléversement permettant de programmer l'ESP. Ce module est en vente sur Internet, néanmoins dans une démarche OpenScience il est intéressant de le fabriquer soi-même. D'autant plus que, cela demande peu de matériel et des compétences accessibles.

Matériel nécessaire :

- Fer à souder
- Etain pour les soudures

Éléments de fabrication:

- Une plaque à trous de 6×10 trous
- Une barrette femelle de 1×6 pins
- Une barrette femelle de 2x4pin
- Un bouton poussoir
- Un TTL FT232RL

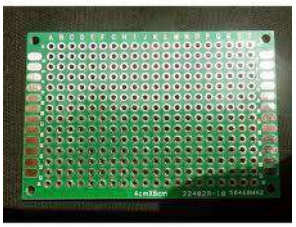



Figure 1: photo des différents éléments

Puis il faut relier les élément de la façon suivante:

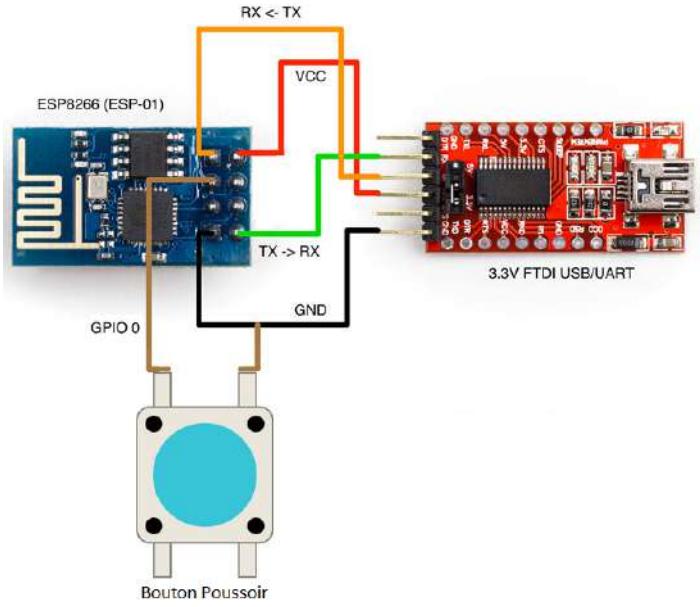


Figure 2: Schéma de principe des liaisons entre les composants

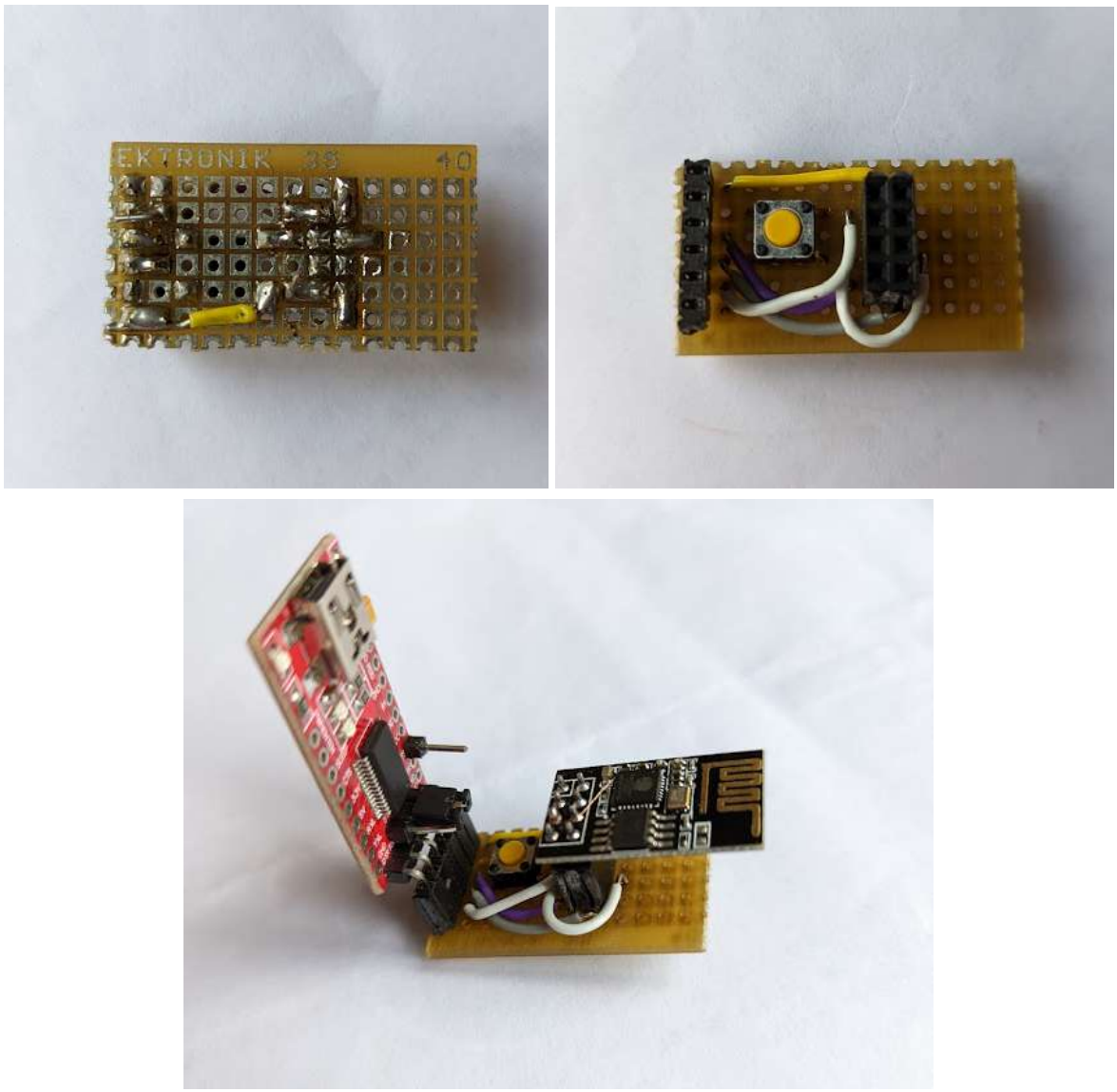


Figure 3: Photos du câblage du firmware

Maintenant, il ne reste plus qu'à le programmer, pour cela rendez vous sur le site mini-projet [3], un lien vers le code à téléversé est disponible.

Annexe 13: Diagramme de Gantt au lancement du projet

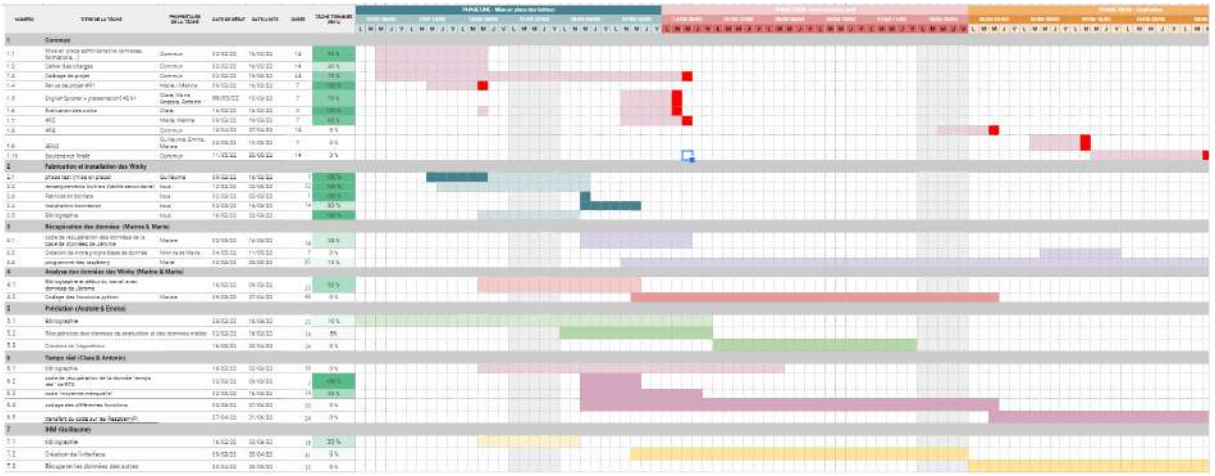


Figure 1: Diagramme de GANTT prévisionnel

Nous ne sommes pas parvenus à exporter la globalité des diagrammes de manière nette. Nous nous excusons pour cela et vous laissons vous rendre sur l'outil collaboratif suivant afin de mieux le visualiser.

https://docs.google.com/spreadsheets/d/17H6YsLyfyOk01QEsql_GAdNgOw6eYvUZaUpJvj8d7KY/edit?usp=sharing

Annexe 14: Diagramme de Gantt actualisé

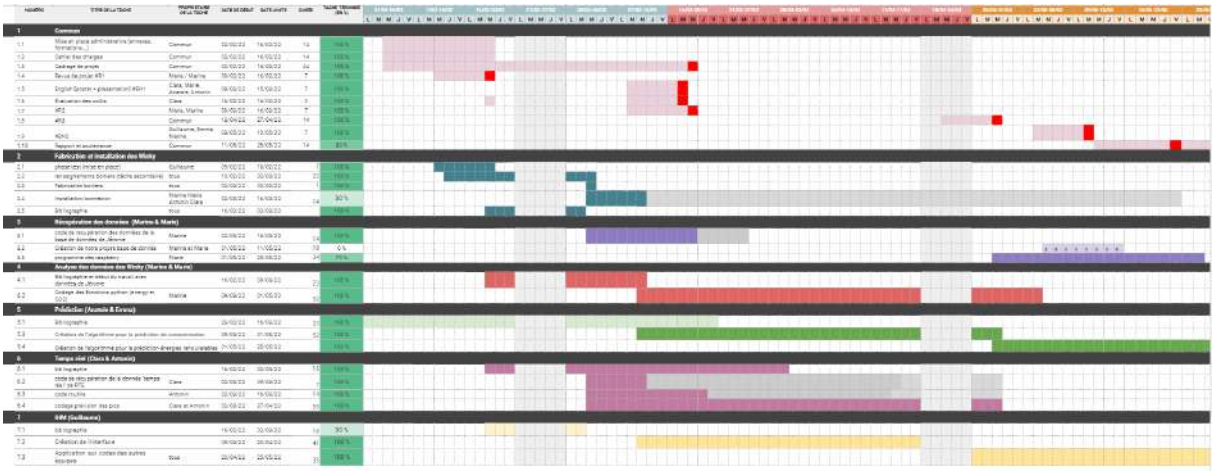


Figure 1: Diagramme de GANTT actualisé

https://docs.google.com/spreadsheets/d/1JD0IOpy2X2wzInan-DyKrXfRT48gh7iTdr9j2Vp5A9I/edit?usp=sharing

Comme mentionné dans le rapport, le facteur qui a engendré la majorité des retards est dû à la mise ne place des Winky.