

# PYTHON

## PROJET – Transport en Commun



### EQUIPE B

*Damien Marechal*

*Vincent Kerouel*

*Antonin Pourroy*

*Julien Lopez*

*Romain Ligavant*

*Marine Cheyssial*

*Julien Lefeuvre*

*Alexis Gillet*

*Vincent Kerouel*

*Steven Bodere*

*Marlo Kerleroux*

*Morgane Baysset*

*Frédéric Volant*

*Marc Questel*

# Table des matières

1. Présentation du projet et de ses besoins .....	2
2. Organisation du projet .....	2
2.1. Stratégie du groupe .....	2
2.1.1. Matrice de LED .....	2
2.1.2. Traitement / Interprétation des données .....	3
2.1.3. Gestion de configuration (Git) .....	3
2.1.4. Gestion de projet .....	3
2.2. Diagramme de Gantt.....	3
3. Méthodes et réalisations du projet .....	4
3.1. Partie traitement et interprétation des données.....	4
3.1.1. Présentation et fonctionnement du code.....	4
3.1.2. Présentation des codes des différents API .....	5
3.1.4. Développement de la page web.....	17
3.2. Partie matrice LED et affichage .....	20
3.2.1. Prise en main et mise en marche du matériel.....	20
3.2.2. Présentation et fonctionnement du code.....	23
4. Problèmes rencontrés et ressenti des membres de l'équipe .....	24
5. Conclusion .....	29

# 1. Présentation du projet et de ses besoins

Dans le cadre de notre formation en python et en gestion de projet, il nous est demandé de réaliser un panneau d'affichage. Celui-ci indiquera les données en temps réel de temps d'attente des transports en commun sur les arrêts de bus.

Le panneau sera configurable, afin de s'adapter à sa ville et à sa station de bus. (Brest, Caen, Rennes ou Nantes)

Son fonctionnement consiste à collecter les informations des prochains départs à un arrêt, de traiter ces données puis de proposer un affichage sur une matrice Led (panneau d'affichage), à la façon des écrans digitaux postés à certains arrêts.

## 2. Organisation du projet

Pour réaliser ce projet, un temps imparti de 15H tout compris nous a été attribué, comprenant le développement R&D, la rédaction des différents documents et la soutenance finale.

Un groupe de 13 étudiant de CIPA 5 a été composé pour répondre aux besoins de ce projet :

- |                     |                    |
|---------------------|--------------------|
| - Marine Cheyssial, | - Romain Ligavant, |
| - Antonin Pourroy,  | - Steven Bodere,   |
| - Julien Lefeuvre,  | - Marlo Kerleroux, |
| - Julien Lopez,     | - Morgane Baysset, |
| - Alexis Gillet,    | - Frédéric Volant, |
| - Vincent Kerouel,  | - Marc Questel     |

### 2.1. Stratégie du groupe

Nous avons réparti les tâches en fonction des compétences de chacun. Nous sommes divisés en 4 groupes. Nous avons défini un chef de projet , Damien, qui suit l'avancement de chaque groupe, et répond aux questions.

#### 2.1.1. Matrice de LED

Le groupe pour la matrice LED, ce groupe se charge de récolter des informations sur la matrice, la communication avec la carte, les couleurs d'affichage.

Cette équipe sera aussi en charge de la gestion du matériel physique (Raspberry, panneau LED ...) et de la gestion de configuration du Raspberry.

Ce qui correspond aux tâches 3, 5, 6 ,7, 10. Antonin, Vincent, Steven sont chargés de cette partie, ils font partie de l'option Système embarqué.

### 2.1.2. Traitement / Interprétation des données

Le groupe pour le traitement/ l'interprétation des données, ce groupe gère la récupération des données des API, le formatage des données pour qu'elles soient toutes sous la même forme.

Ce qui correspond aux tâches 2, 4, 6, 7, 10. Marc, Alexis, Romain, Frédéric, Julien se chargent de cette partie, ils font partie de l'option Génie logiciel

### 2.1.3. Gestion de configuration (Git)

Le groupe pour la gestion de configuration (Git), ce logiciel permet une gestion de version centralisée. Tout le monde travaille sur la même version.

Ce qui correspond aux tâches 4, 6, 7, 10. Marlo se charge de la création du Gt et de la rédaction du README.

### 2.1.4. Gestion de projet

Le groupe de gestion de projet, ce groupe réalise les Gantt, les rapports, le diaporama de soutenance.

Ce qui correspond aux tâches 1, 8, 9, 10. Morgane, Marine et Julien constituent ce groupe.

## 2.2. Diagramme de Gantt

	19/10/2021			21/10/2021			27/10/2021			28/10/2021					
	9h-10h	10h-11h	11h-12h	14h-15h	15h-16h	16h-17h	9h-10h	10h-11h	11h-12h	9h-10h	10h-11h	11h-12h	14h-15h	15h-16h	16h-17h
Présentation du projet et formation des équipes															
Tâche 1: Réalisation du dossier de première séance															
Tâche 2: Comprendre et récupérer les données															
Tâche 3: Comprendre le fonctionnement de la matrice LED															
Tâche 4: Utilisation et traitement des données															
Tâche 5: Affichage des premiers symboles sur la matrice LED															
Tâche 6: Affichage des premières données sur la matrice LED															
Tâche 7: Affichage des données souhaitée sur la matrice LED															
Tâche 8: Rédaction du rapport															
Tâche 9: Création du diaporama pour la présentation du projet															
Tâche 10: Présentation du diaporama à l'équipe															
Tâche 11: Présentation oral / soutenance															

Diagramme de Gantt

Dans un souci d'efficacité, nous avons souhaité faire un diagramme de Gantt prévisionnel afin de répartir les tâches pour ne pas de faire prendre de cours par le temps. Nous avons réparti dans ce diagramme les missions principales en tâches et nous leur avons imparti une durée.

Nous pouvons remarquer que chaque équipe (rédactionnelle, traitement des données et affichages sur la matrice LED) a des trous dans son planning, lorsqu'une équipe n'est pas prévu sur la séance, les membres de cette équipe rejoindrons les autres pour soulager leurs charge de travail.



### 3. Méthodes et réalisations du projet

#### 3.1. Partie traitement et interprétation des données

##### 3.1.1. Présentation et fonctionnement du code

Les données concernant les transports en commun sont disponibles au public sur le web, elles sont partagées par les différentes sociétés de transports en commun des différentes villes comme Bibus, Twisto, TAN et STAR.

La base du programme consiste à requêter les APIs web en trouvant l'URL de requête le plus optimal. Le fichier recueilli est sous format JSON, afin de comprendre nos données et les traiter par la suite nous avons utilisé l'outil en ligne JSON Parser. Cet outil nous a permis d'ajouter une indentation lisible au code JSON afin de le comprendre.

```
[{"sens":1,"terminus":"Souchais","infotrafic":false,"temps":"2mn","dernierDepart":"false",
"tempsReel":"false","ligne":{"numLigne":"95","typeLigne":3},"arret":{"codeArret":"ICAM1"}},
{"sens":2,"terminus":"HaluchÃreBatignolles","infotrafic":false,"temps":"18mn",
"dernierDepart":"false","tempsReel":"true","ligne":{"numLigne":"95","typeLigne":3},"arret":{"codeArret":"ICAM2"}},
{"sens":1,"terminus":"Souchais","infotrafic":false,"temps":"27mn","dernierDepart":"false",
"tempsReel":"false","ligne":{"numLigne":"95","typeLigne":3},"arret":{"codeArret":"ICAM1"}},
```

Retrouvez ci-dessus un extrait d'un fichier JSON brut, on peut en effet constater que sa lisibilité par un utilisateur laisse à désirer. Nous utilisons JSON Parser pour simplifier la compréhension du fichier, retrouvez ci-dessous l'extrait de code indenté :

```
{
  "sens":1,
  "terminus":"Souchais",
  "infotrafic":false,
  "temps":"2 mn",
  "dernierDepart":"false",
  "tempsReel":"false",
  "ligne":{
    "numLigne":"95",
    "typeLigne":3
  },
  "arret":{
    "codeArret":"ICAM1"
  }
},
{
  "sens":2,
  "terminus":"HaluchÃre - Batignolles",
  "infotrafic":false,
  "temps":"18 mn",
  "dernierDepart":"false",
  "tempsReel":"true",
  "ligne":{
    "numLigne":"95",
    "typeLigne":3
  },
  "arret":{
```

```

        "codeArret": "ICAM2"
    },
    {
        "sens": 1,
        "terminus": "Souchais",
        "infotrafic": false,
        "temps": "27 mn",
        "dernierDepart": false,
        "tempsReel": false,
        "ligne": {
            "numLigne": "95",
            "typeLigne": 3
        },
        "arret": {
            "codeArret": "ICAM1"
        }
    },

```

Une fois triés, les données sont clairement lisibles, on peut apercevoir le sens de rotation des bus, le terminus, le temps nécessaire à l'arrivée du bus à un arrêt donné, le numéro de ligne et le nom de l'arrêt.

Traitement du fichier JSON :

Pour traiter les fichiers générés par l'API il est nécessaire d'utiliser les dictionnaires. Cette méthode nous permet de récupérer uniquement certaines informations du fichier JSON. Par la suite, il nous suffit de récupérer l'information dans le dictionnaire et de l'afficher dans le terminal.

Nous allons maintenant vous présenter les traitements effectués par les différents membres du groupe API, vous pouvez retrouver dans le chapitre suivant les traitements effectués pour les différents API.

### 3.1.2. Présentation des codes des différents API

#### - Bibus Brest

La documentation de l'API de Brest contient une requête en mesure de nous fournir les heures de passage du prochain bus à notre arrêt, getRemainingTimes. Afin de faire fonctionner cette requête il faut des paramètres, certains dont nous disposons, d'autre qu'il faudra déterminer :

- Le format, nous nous sommes mis d'accord pour utiliser du JSON, nous insérerons donc "json" en brute dans la requête
- Le route\_id, il s'agit de la ligne pour laquelle nous effectuerons la requête
- Le trip\_headsign, il s'agit du terminus, ce qui donne le sens de circulation du bus sur la ligne
- Le stop\_name, le nom de l'arrêt

L'API est beaucoup plus rudimentaire que celle de Rennes par exemple, il faut donc utiliser plusieurs requêtes différentes pour obtenir les informations dont nous avons besoin.

Le traitement des informations pour la ville de Brest se déroule donc en plusieurs étapes :

1. récupérer toutes les lignes qui passent par l'arrêt donné en paramètre
2. pour chaque ligne, récupérer leurs terminus (sens)
3. récupérer le prochain bus pour chaque sens de chaque ligne

En guise d'exemple, les prochaines explications seront basées sur l'arrêt "Patinoire".

Dans un premier temps, nous utilisons la fonction "getRoutes\_Stop" de l'API qui, pour un arrêt donné, nous renvoie la liste de toutes les lignes qui passent par cet arrêt, avec le numéro de ligne et le nom de la ligne. Nous récupérerons le numéro de ligne, qui nous sera utile pour les prochaines étapes de traitement.

```
[{
    'Route_id': '22',
    'Route_long_name': 'Guilers Roberval - Quatre Pompes'
}, {
    'Route_id': '05',
    'Route_long_name': 'Port de commerce - Provence'
}, {
    'Route_id': '01',
    'Route_long_name': 'Gare SNCF - Hopital Cavale'
}]
```

Un simple traitement sur ces données nous permet de récupérer uniquement les numéros de ligne dans un tableau.

```
['22', '05', '01']
```

Pour chacune de ces lignes, il faut maintenant trouver leurs terminus afin de déterminer les sens de circulation des bus. La requête getDestinations nous retourne les terminus de chaque ligne.

Exemple pour la ligne 01 :

```
[{
    'Trip_headsign': 'Gare'
}, {
    'Trip_headsign': 'Hôpital Cavale'
}]
```

Nous créons ensuite un tableau qui contiendra tous nos paramètres pour les requêtes qui récupéreront les prochains bus.

```
[{
  'route': '22',
  'terminus': 'BREST 4 Pompes'
}, {
  'route': '22',
  'terminus': 'GUILERS Roberval'
}, {
  'route': '05',
  'terminus': 'Port de Commerce'
}, {
  'route': '05',
  'terminus': 'Provence'
}, {
  'route': '01',
  'terminus': 'Gare'
}, {
  'route': '01',
  'terminus': 'Hôpital Cavale'
}]
```

Nous disposons maintenant de tous les paramètres nécessaires à l'exécution de la requête `getRemainingTimes` : la ligne, la destination et l'arrêt. La requête nous renvoie le résultat suivant :

```
[
  {
    'Advance': '00:00:00',
    'Arrival_time': '15:00:28',
    'Delay': '00:00:43',
    'EstimateTime_arrivalRealized': '15:01:26',
    'Remaining_time': '00:01:31'
  },
  {
    'Advance': '00:00:00',
    'Arrival_time': '15:08:28',
    'Delay': '00:00:00',
    'EstimateTime_arrivalRealized': '15:08:28',
    'Remaining_time': '00:08:33'
  }
]
```

On y voit un tableau contenant deux objets javascript, représentant chacun un bus. Pour le premier, on peut voir qu'il n'a pas d'avance, que l'heure d'arrivée prévue est à 15:00:28, mais il a 43 secondes de retard, son heure d'arrivée réelle est donc 15:01:26 et il reste 1 minute et 31 secondes d'attente.



Nous utilisons la valeur `EstimateTime_arrivalRealized` car elle donne l'heure réelle d'arrivée du bus à notre arrêt, en tenant compte de l'avance ou du retard. Il reste une étape avant de pouvoir enregistrer ces données, nous nous sommes mis d'accord avec l'équipe pour que les heures de passage soient envoyées sous forme de timestamp. Nous utilisons donc la fonction `mktime` du package `time` pour créer un timestamp à partir d'un objet `datetime`.

Suite à tous ces traitements, il suffit d'inclure les données nécessaires au panneau sous forme de tableau de dictionnaire ou objet Javascript et de trier ce tableau en fonction des heures d'arrivée des prochains bus :

```
[
  {
    'ligne': '01',
    'terminus': 'Gare',
    'temps': 1635346828.0
  },
  {
    'ligne': '05',
    'terminus': 'Provence',
    'temps': 1635346900.0
  },
  {
    'ligne': '01',
    'terminus': 'Hôpital Cavale',
    'temps': 1635347304.0
  },
  {
    'ligne': '05',
    'terminus': 'Port de Commerce',
    'temps': 1635347386.0
  }
]
```


Afin de vérifier que tout ce traitement renvoie bien le résultat attendu, nous avons comparé la sortie de notre programme aux données du site de Bibus. Pour un trajet de la Patinoire vers la Gare, le site de Bibus nous propose ce trajet par la ligne 01 :

Départ


Patinoire, Brest

Arrivée


Gare, Brest




14min




17min




1 trajet



38min



2 trajets

 01

11:56 > 12:10

14min

Nous sélectionnons le bus correspondant dans la liste des résultats de notre programme :

```
{'ligne': '01', 'terminus': 'Gare', 'temps': 1635335788.0}, {
```

Un simple convertisseur de timestamp en ligne nous confirme que les horaires correspondent bien :



Cela conclut la récupération des données depuis l'API de Brest.

## - Nantes réseau TAN

```
# Requête vers l'api tan
def get_data():
    return requests.get('http://open.tan.fr/ewp/tempsattente.json/COMM')
```

La fonction `get_data()` permet de retourner les données de chaque arrêt à partir de l'api tan de Nantes. Dans cet exemple, nous récupérons les données de l'arrêt ICAM proche de l'Isen Nantes. Plusieurs liens sont mis à disposition pour renvoyer une structure de retour, ce choix a été pris en compte car c'est celui qui propose une variété de données détaillé pour chaque ligne.

En ce qui concerne l'arrêt, nous rajoutons non pas l'arrêt, mais le `codeArrêt` à la fin de l'URL, une problématique à prendre en compte notamment pour la synchronisation entre l'entrée de données sur l'application WEB et le traitement des données en python.

```
# Récupération des données
def get_information():
    data = get_data()
    json_string = data.text
    return json_string
```

La fonction `get_information()` permet de retourner les données renvoyé par la fonction `get_data` dans un format string que l'on va pouvoir manipuler par la suite.

**# Formatage des données (ligne terminus temps)**

```
data = get_information()
dataJson = json.loads(data)
dataList = []
for i in range(len(dataJson)):
    l = {}
    l["ligne"] = dataJson[i]["ligne"]["numLigne"]
    l["terminus"] = dataJson[i]["terminus"]
    l["temps"] = dataJson[i]["temps"]
    dataList.append(l)
```

Dans un premier temps, on récupère les données de la fonction `get_information` et on transcrit au format JSON. Cela nous permet de parcourir les données au format JSON et de récupérer les informations qui nous intéresse, à savoir le numéro de ligne, le terminus et le temps. Ces données sont ajoutées à la list `dataList`.

**# Traitement des données**

```
for d in dataList:
    if d["temps"]=="Proche":
        d["temps"]=str(60+current_time)
    elif "h" in d["temps"]:
        # Offset présent pour éviter un bug d'affichage de 59m alors que t>1h
        if ">" in d["temps"]:
            offset = 3600/2
        else:
            offset = 0
        d["temps"] = "".join([c*c.isnumeric() for c in d["temps"]])
        if len(d)>0:d["temps"] = str(int(d["temps"])*3600+current_time+offset)
    elif "m" in d["temps"]:
        d["temps"] = d["temps"][0:d["temps"].index("m")+1]
        d["temps"] = "".join([c*c.isnumeric() for c in d["temps"]])
        if len(d)>0:d["temps"] = str(int(d["temps"])*60+current_time)
    else:
        # We should not be here
        print("Error on data:")
```

La partie traitement des données est importante car l'api de Nantes renvoie les données de temps sous le format : « Proche », « 2 mn », « 4 mn 30 », « 1h » et « >1h ». Pour la problématique du « Proche », nous l'avons converti en 1 minute pour rendre le traitement plus simple. Concernant les heures, si la chaine de caractère possède le symbole « > », nous avons rajouté une variable « offset » afin d'éviter un bug. Il ne faut pas que dans le cas du « >1h », s'affiche 59m.

Offset est une variable qui vient s'ajouter au temps afin d'éviter ce problème. Dans le cas du « 1h » et « >1h » nous les transformons en seconde, ajoutons le temps actuel afin de les convertir en timestamp. Dans le cas du « 4mn », nous avons procéder de la même façon, nous avons convertit en seconde et ajouter le temps courant pour les convertir en timestamp. Un format de données pour le temps que nous avons décidé de renvoyer pour chaque API.

```
#Sorted by time
dataList = sorted(dataList, key=lambda j: float(j['temps']))
```

Pour la dernière partie du programme, nous avons décidé de rajouter une sécurité en plus, car l'api renvoie d'elle-même les données de chaque bus trié par temps. Nous avons donc rajouté un tri par temps restant avant l'arrivée du bus à l'arrêt.

## - Caen Twisto

Pour ce qui est de la programmation, nous nous sommes séparés les tâches par villes disponibles pour la programmation des scripts d'extraction et de mise en forme des données provenant des API.

Dans un premier temps, j'ai décidé d'intégrer les librairies « json », « requests » et « urllib ». « Json » pour l'utilisation des fonctions pour traiter du JSON et « requests » pour les fonctions de récupération des données. La librairie « urllib » permettra elle de proprement encoder une variable contenant du texte non ASCII afin de pouvoir être utilisé dans un lien URL.

```
import json
import requests
import urllib
```

Pour les futurs tests, j'ai ensuite ajouté une variable pour être utilisée directement dans la trame JSON afin de voir si les données sont bien récupérées pour un arrêt donné. J'ai utilisé ici la méthode « parse.quote » de la librairie « urllib » afin d'encoder la variable « ARRET » contenant le nom de l'arrêt pour un lien URL.

```
ARRET = 'Caen-Collège Desmeserets-COL. DESMESERETS'
ARRET = urllib.parse.quote(ARRET, safe="")
```

Je suis ensuite passé à la déclaration des fonctions qui seront utilisées dans la partie principale (main) du code.

J'ai commencé par la fonction « get\_data() », qui permet de retourner les différentes données provenant d'un lien avec l'utilisation de la méthode « get() » provenant de la librairie « requests ».

On remarque que l'URL a été modifié en son bout afin de se concentrer sur un seul arrêt (« refine.nom\_de\_l\_arret\_stop\_name= ») puis j'ai ajouté la variable « ARRET », déjà formaté pour une URL, contenant le nom de l'arrêt que je souhaite tester.

```
def get_data():

    return
requests.get('https://data.twisto.fr/api/records/1.0/search/?dataset=horairestr&q=&rows=100
&facet=ligne&facet=nom_de_l_arret_stop_name&facet=destination_stop_headsign&facet=et
at_de_la_course&refine.nom_de_l_arret_stop_name='+ARRET)
```

La seconde fonction, « `get_arret_information(api_string)` » permet de passer une chaîne en JSON afin de récupérer les données qui nous intéressent, ici la ligne, le nom de la ligne, le sens de la ligne et enfin le temps correspondant au passage du bus.

On récupère les données sous forme d'une chaîne de caractères provenant de l'API et on les insère dans une nouvelle fonction permettant de traiter le JSON avec la méthode « `loads` » de la librairie « `json` ». On déclare ensuite un dictionnaire (« `liste_arret_dico` ») qui récupère les données qui nous intéressent avec une boucle se basant sur le nombre de données présentes dans le JSON (« `nhits` »), les données en question sont « `ligne` », « `nom` », « `sens` » et « `temps` ». On déclare aussi une variable sous forme de chaîne de caractères « `liste_arret_char` » qui récupère les données provenant du dictionnaire pour chaque indentation de la boucle avec la méthode « `append()` ». Cette chaîne de caractères est ensuite retournée à la fin de cette fonction.

```
def get_arret_information(api_string):

    json_string = json.loads(api_string.text)
    dictionnaire_donnee = {}
    for i in range(json_string["nhits"]):
        dictionnaire_donnee[json_string["records"][i]["fields"]["nom_de_l_arret_stop_name"]]
= {
        "ligne": json_string["records"][i]["fields"]["ligne"],
        "nom": json_string["records"][i]["fields"]["nom_de_l_arret_stop_name"],
        "sens": json_string["records"][i]["fields"]["destination_stop_headsign"],
        "temps": json_string["records"][i]["fields"]["horaire_depart_theorique"],
    }
    print(dictionnaire_donnee)
```

Enfin je déclare la partie principale du code (« `main` ») dans laquelle j'ajoute les variables appelant les différentes fonctions déclarées précédemment.



Je déclare une variable « data » qui récupère les données provenant de la fonction « get\_data() » et une variable « liste\_arret » qui prendra la forme d'une liste grâce à la récupération de la liste provenant de la fonction « get\_arret\_information() ». On fini avec l'affichage de ce dictionnaire dans la console provenant du print de la variable « liste\_arret ».

```
if __name__ == "__main__":  
  
    data = get_data()  
    get_arret_information(data)
```

Pour conclure ce code n'est pas la version finale présent dans le programme complet du projet. En effet il a été retravaillé avec l'aide des autres membres du groupe afin de l'inclure dans le code complet concernant les API de chaque ville. Une partie formatage des données a été ajouté à ce code afin que ce dernier fonctionne avec les autres codes, selon un choix réalisé en groupe concernant l'harmonisation de toutes les données. Ce programme fut utile pour la détermination de la formation du code JSON pour savoir comment le traiter dans le programme principal.

## - Rennes réseau STAR

L'API de Rennes, comme Caen et Nantes, intègre un utilitaire d'API en ligne qui permet de part des critères, variables et autres, de construire une URL de requête voulu.

Dans un premier temps, il a donc fallu trouver les bons paramètres à utiliser dans l'API en ligne afin de construire la bonne URL, de manière à faire la bonne requête et récupérer dans un fichier de résultat uniquement les données utiles et nécessaires au traitement en répondant aux besoins. Une fois le bon URL construit il a fallu faire le traitement sous Python.

Pour ce faire, un fichier JSON externe de paramètres à été créé de manière à simuler la communication avec le site web en attendant que celui-ci soit développé, et lire les paramètres de villes et arrêts, ici "Rennes" et "//un\_arret\_voulu".

Ensuite se sont différentes fonctions qui vont faire les différentes étapes du traitement. On commence par lire le fichier json externe avec la fonction "get\_parameters", ensuite, avec la ville et l'arrêt récupéré on construit une URL avec le nom de l'arrêt de bus désiré avec la fonction "get\_url" sous la forme suivante :

```
"https://data.explore.star.fr/api/records/1.0/search/?dataset=tco-bus-circulation-  
passages-tr" \  
"&q=&rows=10&sort=-depart&facet=nomarret&refine.nomarret=" + bus_stop +  
"&timezone=Europe%2FParis"
```

Traduction de l'URL en langage simple : On cherche sur l'API de Rennes, dans le dataset (dossier de données) des passages de bus, à récupérer 10 résultats, triés par ordre chronologique, pour l'arrêt voulu, au fuseau horaire de Paris.

Ensuite il faut faire la requête sur l'API avec le lien précédent, puis récupérer le résultat en le convertissant en fichier JSON avec la fonction `"get_json"`. Une fois fait, le traitement des données se fait avec la fonction `"get_data"` de manière très simple et organisée de la manière suivante :

```
for bus in data_from_request["records"]:  
    bus_field = bus["fields"]  
    bus_line = bus_field["nomcourtligne"]  
    bus_destination = bus_field["destination"]  
    bus_departure = bus_field["depart"]
```

Ces quelques lignes de code, vont chercher pour chaque enregistrement du fichier JSON ("records) donc pour chaque bus, dans le champ "fields", les données de la ligne, la destination, et l'horaire de départ.

La fonction `"get_data"` fait tout ce traitement et renvoie un dictionnaire avec les données "ligne", "destination" et "temps restant" au format `TIMESTAMP`, pour chaque bus. Chaque dictionnaire de bus est mis dans une liste regroupant ces dictionnaires, liste qui sera envoyée au Raspberry pour transmettre les données sur la matrice de leds.

```
bus_dictionary = {  
    "ligne": bus_line,  
    "destination": bus_destination,  
    "temps": bus_timestamp_str  
}  
bus_list.append(bus_dictionary)
```

Le tout est géré par un main. Il y a également des données imprimées dans la console de manière à vérifier la validité des données avec l'existant sous le format suivant :

```
--- Réseau de bus de la ville de Rennes ---  
  
Liste de tout les bus passant à l'arrêt : Saint-Laurent  
  
*****  
  
Ligne : 51  
Destination : Rennes  
Départ à : 14h40  
Temps restant : 2 min  
  
*****  
  
Ligne : 9  
Destination : Cleunay  
Départ à : 14h41  
Temps restant : 3 min  
  
*****
```

### 3.1.3. Mise en commun des différentes API

Nous avons retenu deux solutions afin de réaliser la mise en commun des différents traitements API au sein d'un même programme Python.

Nous avons retenu la Solution 2 car nous avons réussi à l'implémenter plus facilement.

#### - Solution 1 :

Cette solution consiste à mettre les 4 traitements API à l'intérieur de 4 fonctions ayant pour argument les arrêts et de les appeler en fonction de la ville.

Vous pouvez retrouver la sélection de la ville ci-dessous :

```
def get_information_localisation(ville, arret):
    ville_fonction = ville.upper()
    if (ville_fonction == "BREST"):
        #appel de la fonction API Brest
        print("vous avez choisi la ville 1")
        return get_brest_information(arret)

    elif (ville_fonction == "NANTES"):
        #appel de la fonction API Brest
        print("vous avez choisi la ville 2")
        return get_nantes_information(arret)

    elif (ville_fonction == "CAEN"):
        # appel de la fonction API Brest
        print("vous avez choisi la ville 3")
        return get_caen_information(arret)

    elif (ville_fonction == "RENNES"):
        # appel de la fonction API Brest
        print("vous avez choisi la ville 4")
        return get_rennes_information(arret)

#Main
# Acquisition des donnees
ville_en_cours = input("Dans quelle ville vous vous situez ? ")
arret_en_cours = input("A quel arret vous situez vous ? ")

# Restitution des donnees a l'utilisateur
print("Vous avez sélectionné la ville " + ville_en_cours + " " + "vous vous situez à l'arret " +
arret_en_cours)

# Appel de la fonction
get_information_localisation(ville_en_cours, arret_en_cours)
```

## - Solution 2 :

L'idée de base de la solution 2 est de créer à partir des différents codes des différentes API, un seul et même code regroupant ces derniers. Chaque ville, et arrêt de bus est récupéré, via un fichier JSON renvoyé par le serveur web, et en fonction de cela le programme s'adapte et exécute de manière dynamique les traitements propres à chaque ville. Le programme est le même pour toutes les villes, mais chaque fonction prend en paramètre la ville et l'arrêt de bus pour son traitement.

Cette idée est tout à fait réalisable pour les API de Rennes, Caen et Nantes, mais n'est pas optimale pour Brest. En effet, les API de Rennes, Caen et Nantes sont différentes mais reposent sur une même mécanique de fond, à savoir, un lien initial envoie une requête, et l'API renvoie le résultat voulu sous forme d'un fichier JSON. La seule problématique était de poser la bonne question pour avoir la bonne réponse, donc de trouver les bons paramètres à envoyer dans le lien de requête pour avoir les résultats voulu. A partir de là, avec une seule requête on peut donc faire tout le traitement nécessaire du fichier JSON de résultat afin d'extraire les données voulues, ligne, destination, temps restant.

Cependant l'API de Brest étant plus rudimentaire, ce fonctionnement là n'est pas possible, et il est donc nécessaire d'envoyer plusieurs requêtes, et faire plusieurs traitements des fichiers envoyés par les requêtes pour arriver au même résultat. Nous avons essayé d'intégrer le traitement de Brest, de la même manière que pour les autres villes, mais cela ne s'avérait pas possible de part la mécanique de fonctionnement différente. Nous avons donc été contraints de laisser le traitement de Brest tel quel, dans une seule et même fonction propre à cette ville.

La solution étant optimale pour Rennes, Caen et Nantes, nous avons donc choisi de la conserver et de procéder à un traitement unique pour Brest. Le programme est donc "dynamique" pour les 3 villes, et dans le cas où il s'agit de Brest, il appelle la fonction spécifique à Brest. Finalement, peu importe la ville, les données d'entrées et de sorties du programme sont les mêmes, ce qui était le but principal d'un programme regroupant les 4 villes. On a donc bien en entrée la ville et l'arrêt de bus voulu, et en sortie, une liste de dictionnaires pour chaque bus, envoyée à la raspberry pour traitement avant affichage sur la matrice LED.

Pour les 3 villes au fonctionnement similaires, le traitement dynamique se fait donc de part un dictionnaire de clé, et en fonction des villes, le programme choisit la bonne clé dans le dictionnaire, pour le traitement des fichiers JSON de résultat renvoyé par les API :

```
def get_data(city, data):
    bus_list = []
    keys = {
        "RENNES": ["nomcourtligne", "destination", "depart", ""],
        "NANTES": ["ligne", "terminus", "temps"],
        "CAEN": ["ligne", "destination_stop_headsign", "horaire_de_depart_reel",
        "date_du_jour"]
    }
    Etc ...
```

Ensuite le traitement prend les clés nécessaires en fonction des villes :

```
if city == "RENNES":
    for bus in data["records"]:
        field = bus["fields"]
        line = field[keys[city][0]]
        destination = field[keys[city][1]]
        departure = field[keys[city][2]]
```

Etc ...

De la même manière, il y a un traitement de temps différent pour chaque ville. Et donc en fonction de la ville appelée le programme exécute la partie de la fonction de la ville concernée afin d'obtenir le temps sous forme d'un timestamp. Le timestamp est nécessaire, car cela permet un format universel pour les 4 API, lors de l'envoi du temps restant pour chaque bus.

```
def get_time(city, line, destination, departure, date):
    if city == "RENNES":
        departure = str(datetime.timestamp(datetime.strptime(departure, "%Y-%m-%dT%H:%M:%S%z")))
    elif city == "NANTES":
```

Etc

### 3.1.4. Développement de la page web

Afin de faciliter le paramétrage global des données de réseaux de transports en commun en temps réel, c'est-à-dire le réglage de la ville et de l'arrêt, nous avons trouvé nécessaire de mettre en place un serveur web. Celui-ci a été réalisé par Marlo et Marc, et permet de sélectionner une ville et un arrêt à afficher sur l'écran.

Le site se compose d'une page web nommée *index.php* qui, comme indiqué ci-dessous, demande à l'utilisateur de sélectionner la ville, puis l'arrêt où l'on souhaite obtenir les informations des prochains départs.







Une fois la ville et l'arrêt sélectionnés, il suffit de cliquer sur le bouton 'Enregistrer'. La page *index.php* va alors traiter les données prises en compte, les structurer dans un format propre à JSON et les écrire dans un fichier *parameters.json*.

A la page suivante, vous trouverez le code source de la page *index.php*.

Le fichier *parameters.json* sert de 'passerelle' entre les valeurs entrées depuis le site et l'afficheur. Il se compose d'une simple requête composée de la ville et de l'arrêt de transport en commun, comme montré ci-dessous.

```
parameters.json 44 Bytes
1 {
2   "city": "Nantes",
3   "bus_stop": "ICAM"
4 }
```

```




1 <!doctype html>
2 <html lang="fr">
3
4 <head>
5   <meta charset="utf-8">
6   <title>Projet Python</title>
7   <link rel="stylesheet" href="css/bootstrap.min.css">
8   <link rel="stylesheet" href="css/style.css">
9   <script type="text/javascript" src="js/bootstrap.min.js"></script>
10  <script type="text/javascript" src="js/jquery-3.6.0.min.js"></script>
11  <script type="text/javascript" src="js/script.js"></script>
12 </head>
13
14 <body class="text-center">
15   <div class="modal-dialog" role="document">
16     <div class="modal-content rounded-5 shadow">
17       <div class="modal-header p-5 pb-4 border-bottom-0">
18         <h2 class="fw-bold mb-0">Horaires bus</h2>
19       </div>
20
21       <div class="modal-body p-5 pt-0">
22         <div id="val-act" class="alert alert-info" role="alert">
23           <?php
24             $strJsonFileContents = file_get_contents("data/parameters.json");
25             $array = json_decode($strJsonFileContents);
26             $result = $array->bus_stop;
27             echo "Arrêt enregistré : ";
28             echo $result;
29             $result = $array->city;
30             echo " - Ville : ";
31             echo $result;
32           ?>
33         </div>
34         <form action="<?php echo $_SERVER['PHP_SELF'];?>" method="POST">
35           <div class="form-floating mb-3">
36             <select id="ville" name="ville" class="form-control">
37               <option disabled selected value> -- Aucune -- </option>
38               <option value="Brest">Brest</option>
39               <option value="Caen">Caen</option>
40               <option value="Nantes">Nantes</option>
41               <option value="Rennes">Rennes</option>
42             </select>
43             <label>Choix de la ville</label>
44           </div>
45           <div id="container-arrets" class="form-floating mb-3"></div>
46           <button id="valider" class="col-5 mb-2 btn btn-lg rounded-4 btn-success" type="submit">Enregistrer</button>
47           <button id="effacer" class="col-5 mb-2 btn btn-lg rounded-4 btn-danger" type="reset">Effacer</button>
48         </form>
49       </div>
50     </div>
51   </div>
52 </body>
53
54 <?php
55
56 if ($_SERVER["REQUEST_METHOD"] == "POST") {
57   $file = fopen("data/parameters.json", "w");
58   $city = $_POST['ville'];
59   $bus_stop = $_POST['arret'];
60
61   $string_to_put_into = "{\n
62     \"city\": \"\" . $city . "\",\n
63     \"bus_stop\": \"\" . $bus_stop . "\"\n
64   }";
65   $fwrite = fwrite($file, $string_to_put_into);
66   fclose($file);
67   header("Refresh:0");
68 }
69 ?>
70
71
72
73 </html>

```

## 3.2. Partie matrice LED et affichage

### 3.2.1. Prise en main et mise en marche du matériel

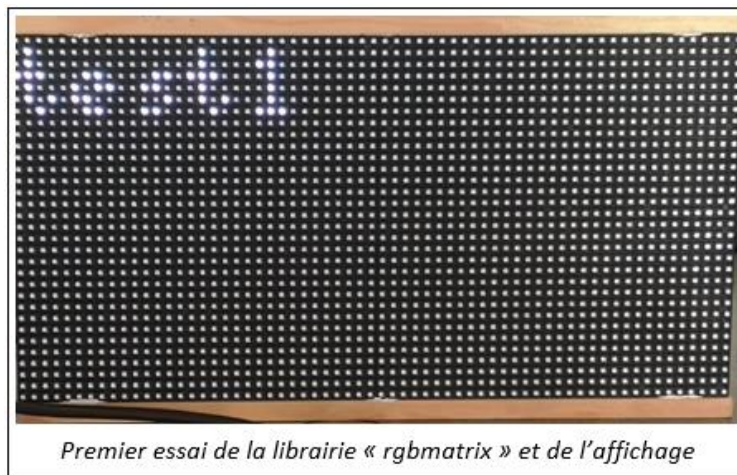
Avant de démarrer tout code, une analyse et des recherches ont été réalisées sur le matériel mis à notre disposition. Le responsable du matériel de notre groupe est Steven Bodere. Pour rappel, dans le cadre de ce projet, nous avons à notre disposition le matériel suivant :

<b>Raspberry Pi 3 B+</b> Il s'agit d'une carte ordinateur compacte qui offre des fonctionnalités diverses et des opportunités illimitées. Il suffit de brancher votre téléviseur, votre clavier, votre souris et votre alimentation, et vous pouvez commencer. Il existe également des cartes complémentaires disponibles pour permettre d'autres utilisations, telles que les modules d'afficheur LCD et de caméra.	
<b>Adafruit RGB Matrix Hat</b> Compatible avec le Raspberry PI 3 B+, cette carte complémentaire qui se plug directement sur le Raspberry permet de contrôler très facilement des matrices RGB et de créer un écran défilant coloré ou un mini mur de LED avec facilité.	
<b>64x32 RGB LED Matrix</b> Un panneau matriciel à LED RGB de 64x32.	

Dans un premier temps, Steven s'est occupé de mettre en marche le Raspberry, et d'y installer Raspberry Pi OS (anciennement Raspbian). Il s'agit d'un système d'exploitation GNU/Linux spécialement conçu et optimisé pour la Raspberry Pi.

Pendant ce temps, Vincent et Antonin se sont penchés sur le panneau RGB et sur comment le mettre en œuvre facilement et rapidement. Pour se faire, ils ont recherché et trouvé une librairie python permettant de contrôler la matrice LED. Cette librairie se nomme "rgbmatrix", la source ci-après : <https://github.com/adafruit/RGB-matrix-Panel>.

Cette librairie permet de contrôler rapidement et facilement l’affichage du panneau LED, à l’aide de fonctions déjà créées. Suite à l’installation du système d’exploitation et de la librairie “rgbmatrix”, un premier essai de démarrage et d’affichage de données sur le panneau LED fût réalisé, qui s’est avéré concluant :



A partir de là, nous étions prêts à travailler sur la réalisation du programme d’affichage des données transmises par l’équipe de récupération de données de station de bus.

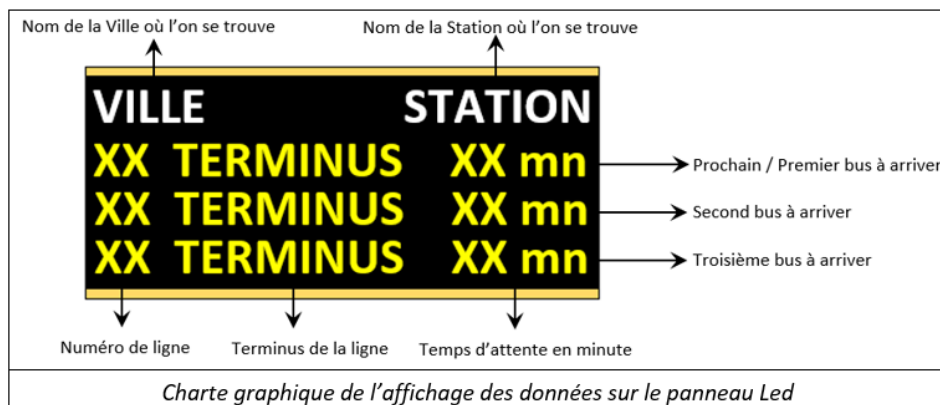
Lors des premiers essais comme celui ci-dessus, nous avons constaté que certaines Leds clignotaient légèrement (bruit), rendant le texte moins lisible et gênant à l’œil. Antonin et Vincent se sont donc penchés sur ce problème récurrent et désagréable, pendant que Steven et Morgane ont commencé à travailler sur l’affichage des données sur le panneau.

Après recherche, Antonin et Vincent ont constaté que la cause du bruit sur les LEDs généré par les GPIOs étaient dûs à l’utilisation d’une pin qui ne supportait pas le mode PWM. Pour corriger ce défaut, une modification Hardware a été nécessaire en jumpant le GPIO4 et le GPIO18, et une adaptation Software en ajoutant les arguments suivant dans les fonctions d’affichage :

--led-slowdown-gpio=5 --led-gpio-mapping=adafruit-hat-pwm



Pendant ce temps, Steven et Morgane ont défini la charte graphique du panneau d'affichage, avec confirmation du chef de projet et de l'équipe complète. Cette charte correspond à ceci :



Dans les cas où les noms de stations, de villes et de terminus seraient trop long pour passer dans leurs espaces attribués dans la charte graphique (risque de superposition sur les données affichées à proximité), les noms seront alors défilant, afin de permettre la lecture du nom entier et ne pas créer de superposition entre les données.

Suite à cela, un essai d'application de cette charte graphique sur le panneau Led a été réalisé, avec des données d'entrées aléatoires fixes que nous avons entrées à la main. Le but était de tester la faisabilité et la cohérence de l'affichage en suivant cette charte graphique avec de fausses données, avant d'interagir avec les données fournies par les API.

Le résultat, que vous trouverez ci-dessous, c'est avéré concluant :



Maintenant que nous avons bien pris en main le matériel, la librairie et aussi mis en place une charte graphique, nous pouvons nous lancer dans le développement du code final gérant la récupération et l'affichage des données reçues par les API.



### 3.2.2. Présentation et fonctionnement du code

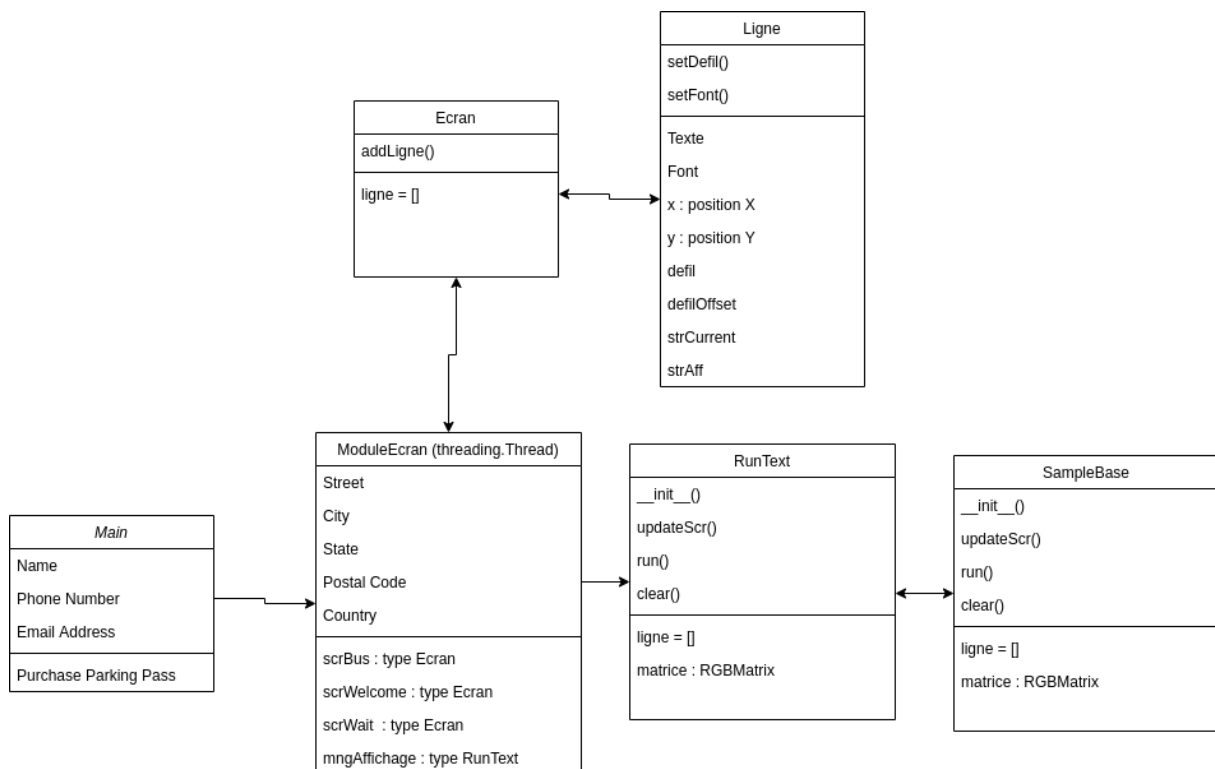
Le code qui gère l’affichage est sous la forme d’une classe héritée d’un thread et nommé ModuleEcran. Cette classe a une fonction run() qui recalcule les temps restant avant le passage du bus et si le bus est passé n’affiche plus le passage en question. Cette classe a une méthode appelée update(data) qui permet après la récupération des passages grâce au API une liste de dictionnaire.

Après le calcul des temps restant la fonction run() qui est une boucle infinie avec un “time.sleep de 5 seconde”, les données sont formées dans une liste de ligne pour être envoyés à la class RunTexte

Puis on envoie l’écran à la classe RunTexte. La classe RunTexte va calculer en fonction des paramètres des lignes les données pour écrire sur l’écran. Cette classe va aussi gérer le défilement des textes.

Dans la classe RunTexte nous avons intégré 2 fonctions de défilement, une première qui permet le défilement du titre (Ville | Nom de l’arrêt de bus) et une deuxième permettant le défilement des noms d’arrêts de bus qui font plus de 7 caractères. En raison de la taille de l’écran, cette fonction permet malgré la taille de certains arrêts, de respecter la charte graphique imposée précédemment.

La classe SimpleBase permet principalement d’initialiser la matrice LED. Lors de l’exécution des programme nous devons ajouter les arguments “--led-cols=64” ainsi que “--led-slowdown-gpio=5” afin de définir la largeur de la matrice à 64 Leds et de définir la vitesse des transitions des GPIOs du raspberry. Afin d’éviter de marquer tous ces arguments à chaque exécution de programme nous avons modifié les valeurs par défaut dans cette librairie soit 64 leds de largeur et fixer le led-slowdown par défaut à 5.



## 4. Problèmes rencontrés et ressenti des membres de l'équipe

### ▪ Ressenti du chef de projet :

#### - Damien Marechal :

Ce projet n'a pas été facile à gérer. Premièrement la deadline était proche et nous disposions de peu de cours pour travailler sur le projet. Plusieurs d'entre nous ont donc dû travailler sur notre temps libre. Ensuite, nous sommes un groupe nombreux, 13 personnes nécessite une bonne gestion d'équipe ainsi qu'une bonne répartition des tâches.

Disposant de connaissances approfondies en Python (entreprise + personnelle). J'en ai profité principalement pour aider les autres membres de l'équipe. J'avais d'ailleurs une vision globale de l'avancement du projet ainsi que des choix technologiques de chacun (ex: choix entre liste ou dictionnaire...). J'ai aussi été en charge de régler les connexions entre les différentes parties : raspberry, Api...

### ▪ Ressenti et problèmes rencontrés équipe GL :

#### - Marc Questel :

Dès le lancement du projet, nous nous sommes attribués les rôles parmi les parties 'extraction des données', 'affichage depuis la carte Raspberry', et 'la rédaction'. Pour l'extraction et interprétation des données nous avons trouvé judicieux que chaque personne s'occupe d'une ville spécifique. J'ai décidé par la suite de m'occuper de l'élaboration de l'API de Brest. Ce qui était pratique pour les personnes qui s'occupaient des APIs de Nantes, Caen et Rennes, c'est que le modèle d'extraction de données est intuitif et pratique à utiliser.

Malheureusement pour le cas de Brest, non seulement les données de tous les arrêts n'étaient pas forcément mis à jour, mais le requêtage se faisait différemment par rapport aux autres villes. De plus, il existe plusieurs fonctions identiques qui laissent à indiquer que le réseau devrait mettre à disposition des fonctions de requêtage plus optimisées.

Heureusement, j'ai très vite reçu de l'aide venant de Frédéric qui s'occupait par la même occasion sur l'extraction des données du réseau Bibus. Un côté pratique que j'ai apprécié vis à vis de l'extraction des données, c'est que Bibus met en place une documentation précise sur les fonctions à employer pour formuler une requête JSON rapidement.

Après que la partie « gestion de l'API de Brest » soit maîtrisée, et ce, grâce à l'aide de Frédéric, je me suis occupé de la partie Back-End d'un site web. Ce site permet à l'utilisateur de paramétrer l'affichage plus facilement en indiquant simplement un arrêt spécifique à une ville. La partie Back du site consiste à créer un fichier PHP qui va récupérer les données entrées dans le site, les traiter et les écrire dans un fichier JSON.

En collaborant avec Marlo qui créait le site, et avec les avis très utiles qu'a apporté le chef de projet, le Back-End s'est complété assez efficacement. Enfin, pour la partie générale, je trouve que la gestion du projet a été bien maîtrisée, de part grâce à chaque membre qui ont participé à des tâches spécifique du projet, et d'autre part grâce au chef du projet qui a su cadrer l'équipe, communiquer sur l'avancement en cours et aider chaque personne de manière efficace.

Contrairement aux autres villes telles que Caen, Rennes ou Nantes, le réseau Bibus de Brest donne une approche plus complexe à l'extraction de données, ce qui compliqué davantage les choses. En effet, le réseau de bus de Brest renvoie un document qui donne accès à des requêtes json sauf qu'en fonction de la requête choisie, les données n'étaient pas forcément disponibles. Raison pour laquelle il était nécessaire de travailler en collaboration avec Frédéric Voland.

- Alexis Gillet :

Au tout départ je n'avais pas vraiment d'idée précise sur comment tout cela allait se dérouler. La direction suivie par le groupe et le partage des idées m'ont vite rassuré et permis de trouver une première voie me permettant de me lancer dans le développement du code, avec l'aide des autres membres du groupes s'occupant du traitement des données provenant des API des différentes villes.

Un autre groupe s'occupait de l'affichage sur le panneau de LEDs, un autre du WEB et enfin un dernier s'occupait de la partie présentation/rédaction. J'ai eu complètement confiance en la capacité des membres des autres groupes à bien réaliser leur tâche, connaissant les bonnes capacités et expériences de chacun.

Pour ce qui est de la communication entre les différents groupes, le rôle endossé par le chef de projet a été très important lors des prises de décisions et du suivi des différents groupes. C'est lui qui mettait tout le monde au courant de l'avancée générale du projet.

Pour les problèmes rencontrés, le principal problème soulevé lors de notre traitement des données est de remarquer qu'aucunes des données obtenues n'étaient les mêmes que celles des autres villes, qui étaient aussi différentes les unes des autres. Il fallait alors trouver un moyen d'uniformiser les données entre elles afin de toutes les faire fonctionner avec un même programme.

Ceci fut le seul défi de groupe rencontré, mis à part les différents challenges des autres membres s'occupant des API des autres villes qui avaient différents problèmes propres à leur API. Pour les problèmes plus individuels, j'ai mis un peu de temps à comprendre comment la trame JSON pour la ville de Caen fonctionnait afin de manuellement choisir un arrêt dans la ville. Ce fut plaisant de constater que tout le monde se soutenait pour que chacun puisse avancer sur son travail à réaliser.

Au final j'ai un ressenti très positif de cette expérience de groupe, qui je pense est nécessaire pour de futur travaux à réaliser en groupe.

#### - Marlo Kerleroux :

Dès le début, le travail à réaliser a été bien réparti par le chef de groupe. Les personnes plus à l'aise en informatique se sont lancés dans le traitement des API ou sur le développement web, tandis que ceux qui s'y connaissent en systèmes embarqués ont travaillé sur la carte Raspberry et l'écran. Pour ma part, je me suis lancé dans la création du Gitlab afin que tout le monde puisse collaborer, et faciliter la mise en commun du code.

De plus, j'ai rédigé le README du projet en markdown, afin que chacun puisse connaître le fonctionnement du projet, son utilité et les technologies utilisées. Je n'ai pas rencontré de problème spécifique sur cette partie de mon travail.

Par la suite, j'ai travaillé sur une page web permettant de configurer la ville et l'arrêt à laquelle on souhaite afficher les horaires sur l'écran. Ayant déjà utilisé les technologies web durant mon alternance, cela me semblait abordable au premier abord. J'ai utilisé du HTML, CSS, JS et les bibliothèques Bootstrap et JQuery.

Les difficultés que j'ai pu rencontrer concernent l'affichage des arrêts en fonction des villes. Certaines API ne permettaient pas de trier les arrêts dans l'ordre alphabétique, et d'autres affichaient les arrêts en double voire plus. Cette partie du projet m'a demandé de collaborer avec les personnes ayant réalisé les API, afin de me renseigner sur le fonctionnement de chacune, et de savoir les données qu'ils avaient besoin en retour.

#### - Romain Ligavant :

Le groupe s'est très vite divisé les tâches, une équipe a travaillé sur les APIs ce qui s'est avéré efficace car chacun échangeait et communiquait sur leur fonctionnement. En plus de cela, le chef de projet passait régulièrement prendre contact et apportait une aide.

Une autre équipe s'est occupée de la partie panneau LED avec qui nous avons échangé sur le type, le format de données que nos programmes devaient leur retourner. La communication entre chaque groupe était claire. En parallèle une autre équipe s'est occupée de la partie WEB avec qui nous avons également eu des échanges car il fallait coordonner le nom des arrêts sur l'API et le WEB.

J'ai apprécié travailler sur ce projet notamment pour le langage utilisé que je n'ai pas l'habitude de pratiquer mais aussi pour le travail en équipe qui s'est bien déroulé.

- Frédéric Volland :

L'équipe étant constituée d'étudiants dans les spécialités électronique et logiciel, la répartition des tâches a été plutôt intuitive et rapide. Pour la partie récupération des données, les étudiants en génie logiciel se sont répartis sur les différentes API. Tout le monde est arrivé dans ce projet avec des niveaux différents et des compétences différentes.

Je pense pouvoir affirmer que tout le monde était bien investi et avait bien compris son rôle dans l'équipe, avec les gens ayant le plus de compétences allant aider régulièrement les personnes en difficulté. L'hébergement du travail de chacun sur un Repository Gitlab s'est révélé très efficace car chaque personne publiait des corrections et des mises à jour régulièrement, ce qui a permis d'améliorer la stabilité globale de l'application.

Le fait d'avoir désigné un chef de projet a été très important dans les échanges entre les groupes, afin de déterminer le chemin à suivre pour le déroulement du projet, les solutions techniques à adopter. Cela nous a évité de partir sur des solutions qui seraient rentrées en conflit entre elles au moment de les rassembler. Globalement, le projet a été enrichissant d'un point de vue travail en équipe, la confiance que chacun s'est accordé dans la réalisation des tâches qui leur a été confiée a été très agréable.

- Julien Lefeuvre :

Le travail a été réparti correctement que ce soit en terme d'effectifs ou en terme de charge de travail, l'entraide est présente dans le groupe. Le diagramme de gantt a été respecté à la lettre ce qui nous a permis de tenir les délais en termes de production et de livrables.

Étant en charge de la production des livrables avec Marine et Morgane, nous avons eu comme première tâche de réaliser le rapport de première séance, par la suite j'ai été déployé sur l'équipe GL pour apporter un soutien logiciel. J'ai donc développé un code permettant l'intégration de l'ensemble des quatre API au sein d'un même programme.

Aux séances suivantes je suis retourné travailler sur la partie rédactionnelle car nous devons rendre des livrables conséquents. Chacun a su travailler en équipe et évoluer dans le partage de connaissances.



- Julien Lopes :

Pour ma part, j'ai travaillé sur l'API de Rennes. La difficulté initiale était de comprendre le fonctionnement de l'API, du système de requêtes et de voir comment traiter le tout. Une fois cette partie réalisée je me suis également confronté à des difficultés côté programmation.

En effet, je n'avais jamais coder pour ce genre de projet auparavant, et j'ai réussi à m'adapter rapidement et mettre à profit les expériences et compétences que j'avais en Python pour pouvoir réaliser le programme de l'API de Rennes. Il a fallu faire plusieurs choix techniques, méthodiques d'un point de vue Python, et j'ai préféré partir sur une méthode : "une fonction = une fonctionnalité" de manière à débayer plus facilement et avoir un programme plus structuré qu'une simple fonction.

Dans l'ensemble je n'ai pas rencontré trop de difficulté pour l'API de Rennes, en revanche, c'est pour l'intégration des différentes APIs entre elles que les problèmes sont apparus, mais vite résolu permettant une cohérence et une fluidité entre les APIs des quatre villes. Dans ce projet, la communication a été un point clé, et je pense que c'est grâce à cette bonne communication que nous avons pu tous nous entraider, résoudre nos problèmes et mener à bien ce projet.

- Marine Cheyssial :

L'équipe a su vite s'organiser en fonction des préférences et des compétences de chacun. Le projet a bien avancé, on a respecté notre diagramme de gantt de la première séance. Étant dans l'équipe gestion de projet, j'ai coordonné l'équipe de traitement des données pour avoir leur ressenti, les problèmes rencontrés et leur explications sur leur codes pour le rapport final.

- **Ressenti et problèmes rencontrés équipe SE :**

- Vincent Kerouel :

Ce projet m'a beaucoup plu car le raspberry PI est un univers que je ne connaissais pas beaucoup et donc ce projet python m'a permis de savoir utiliser cette carte et d'utiliser ses entrées sorties GPIOs. L'ambiance au sein du groupe est plutôt bonne.

- Morgane Baysset :

J'ai beaucoup apprécié l'organisation de notre groupe. La distribution des tâches selon notre option a été un grand avantage car on a pu travailler vite et bien. Chacun a su mettre ses compétences et connaissances en valeur, et les partager avec les autres. Chacun a su travailler en équipe, dans la confiance et évoluer dans le partage de connaissance.

- Steven Bodere :

Au travers de ce projet j'ai pu en apprendre l'intégration d'un projet sur une système embarqué. L'utilisation d'un panneau de LEDs et de la carte Adafruit m'ont permis de mieux comprendre le langage python au travers d'exemples visuels. Le travail en collaboration ma permis avoir plus expérience dans la gestion est les problématiques du travail en équipe, mais toujours très enrichissant.

Les débats pour les différents choix technique ont permis à chaque fois de choisir les meilleur solution avec les contraire du projet. Dans l'ensemble bien que techniquement les projet n'est pas compliqué, l'expérience du travail en équipe donne un vrai challenge au projet.

- Antonin Pourroy :

Au travers de ce projet j'ai pu en apprendre plus sur le fonctionnement des librairies python, le travail en collaboration avec les différents membres de l'équipe s'est révélé efficace, le groupe a fait face à quelques conflit qui se sont soldés par un retour au calme et l'omission des intérêts ayant mené au conflit. L'utilisation d'un panneau de LEDs et de la carte Adafruit m'ont permis de mieux comprendre le langage python au travers d'exemples visuels.

## **5. Conclusion**

Par rapport à notre cahier des charges initial, nous sommes satisfait du résultat. En effet, notre système réussi à récupérer les données de bus pour chaque station de chaque ville. Le panneau Led affiche proprement et en temps réel les données, en suivant une charte graphique fixe.

Un réel travail en équipe a été fourni, dans le partage et la confiance. Malgré les contraintes de temps, notre équipe a su gérer et partager les tâches entres les membres du groupe. Un véritable échange de méthodes et de connaissances a été réaliser sur ce projet, nous sommes très content du résultat.