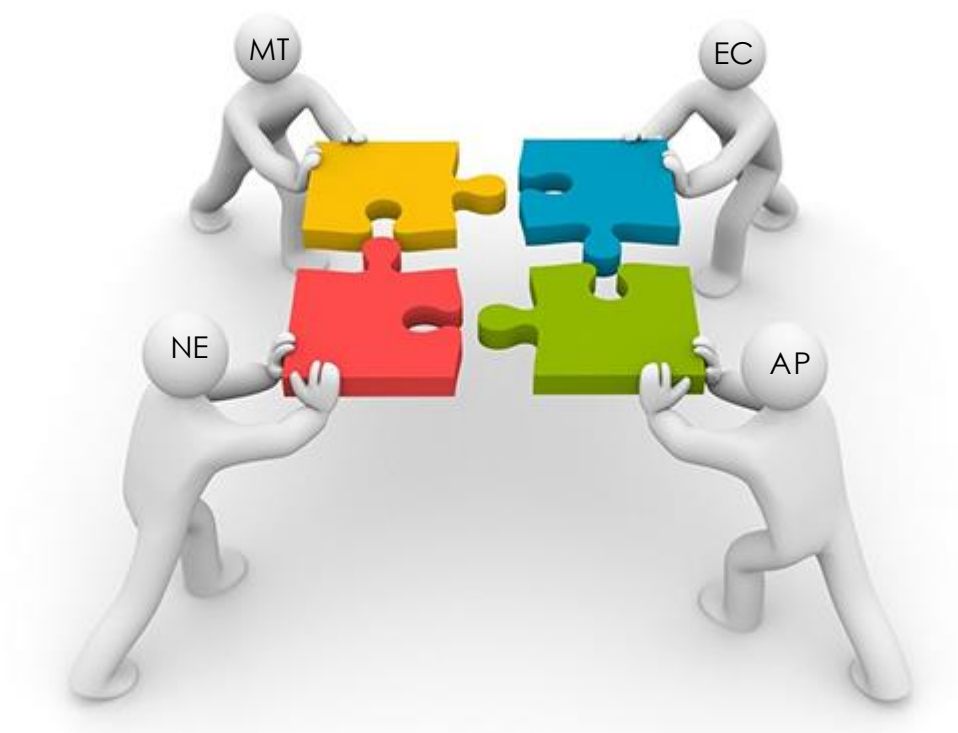




PROJET INTEGRATEUR



ANNEE 2019-2020

NICOLAS EVAIN, AXEL PHANOR, ESTEBAN CALLEJA ET MOUAD TAHTAOUI
L2 Informatique, Parcours Numérique pour les Sciences du Vivant
Collège Sciences et technologies pour l'Énergie et l'Environnement

Épigraphe

« Plus il a affaire à l'informatique, plus cela lui semble être comme un jeu d'échecs : un petit monde bien clos défini par des règles fabriquées, un monde dans lequel des garçons prédisposés par un type de tempérament se laissent prendre et deviennent à moitié fous, tout comme il est lui-même à demi-fou, de sorte que qu'ils se leurrent en croyant qu'ils jouent à ce jeu, alors que c'est le jeu qui se joue d'eux. »

Vers l'âge d'homme (2003) de J. M. Coetzee

Remerciement

Nous tenons à remercier Monsieur Martin Dieguez Lodeiro pour l'enseignement ainsi que toutes les informations qu'il nous a transmises pour nous aider à monter ce rapport. Nous tenons également à remercier l'ensemble de l'équipe pédagogique de l'UPPA pour son enseignement qui nous a permis d'acquérir les compétences informatiques nécessaire à la réalisation de ce projet d'intégration.

Avant-propos

Cette unité d'enseignement est destinée aux étudiant désireux de mettre en commun leurs compétences et connaissances pour la réalisation d'un projet de groups. Les étudiants seront mis à rude épreuve et confronté à moultes difficultés. Ils devront intégrer tout le savoir acquis durant 2 années éprouvantes afin de construire un projet de A à Z.

Ce rapport comporte notre projet intégrateur sur la problématique du réchauffement climatique.

Un projet de récupération, d'analyse et de traitement de données environnementales basé sur des expérimentations. Le but étant de construire une application web capable d'anticiper des événements à partir de données afin de proposer une carte interactive qui nous permettra de faire de la data visualisation.

Pour conclure cet avant-propos, je remercie, par avance, le lecteur pour toute remarque de fonds ou de forme.

Nicolas Evain, Axel Phanor, Esteban Calleja et Mouad Tahtaoui
Étudiants en L2 Informatique Numérique pour les Sciences du Vivant
Faculté des sciences et Techniques de la Côte Basque
Université de Pau et des Pays de l'Adour

Sommaire

Table des matières

Épigraphe.....	2
Remerciement.....	3
Avant-propos.....	4
Sommaire	5
1. Cahier des Charges.....	7
1.1. Présentation du projet	7
1.1.1. Contexte	7
1.1.2. Objectif	7
1.1.3. Description.....	8
1.2. Expression des besoins.....	8
1.2.1. Besoin fonctionnels.....	8
1.2.2. Besoins non fonctionnels.....	9
1.3. Contraintes.....	10
1.3.1. Délais	10
1.3.2. Coûts	10
1.3.3. Autre contrainte.....	10
1.4. Déroulement du projet.....	11
2. Modélisation	12
2.1. Diagramme de cas d'utilisation utilisateur	12
2.2. Diagramme de cas d'utilisation Administrateur	13
2.3 Diagramme de classe	14
2.4 Diagrammes de séquences	14
2.5. Diagramme d'activité.....	16
3. Récolte des données	17
3.1. Choix du langage et utilisation de l'API pyowm.....	17
3.2. Raspberry Pi.....	18
3.3. Amélioration possible.....	20
4. Base De Données.....	21
4.1. Importation des données.....	21
4.2. Structure de données	21
5. Application Web.....	24

5.1. Structure du site globalWarming.....	24
5.2. Incorporation de la map Leaflet (open street map).....	25
5.3. Les interfaces.....	28
5.3.1. Interfaces d'inscription.....	28
5.3.2. Interfaces de connexion.....	29
5.3.3. Les sessions PHP.....	29
6. Conclusion.....	31
7. Perspective dévolution.....	32
Annexes.....	34
Nuage de mot :.....	34
Structure base de données :.....	35
Tutogit :.....	36
Processus de data mining :.....	40

1. Cahier des Charges

1.1. Présentation du projet

1.1.1. Contexte

Ce projet s'inscrit dans le cadre du projet intégrateur, c'est-à-dire qu'il s'agit de travailler en groupe afin de mettre en commun toutes les compétences acquises durant le début de notre scolarité.

Le projet est en adéquation avec la filière Licence informatique numérique **pour** les sciences du vivant, il était donc nécessaire de trouver un sujet qui respecte cette condition.

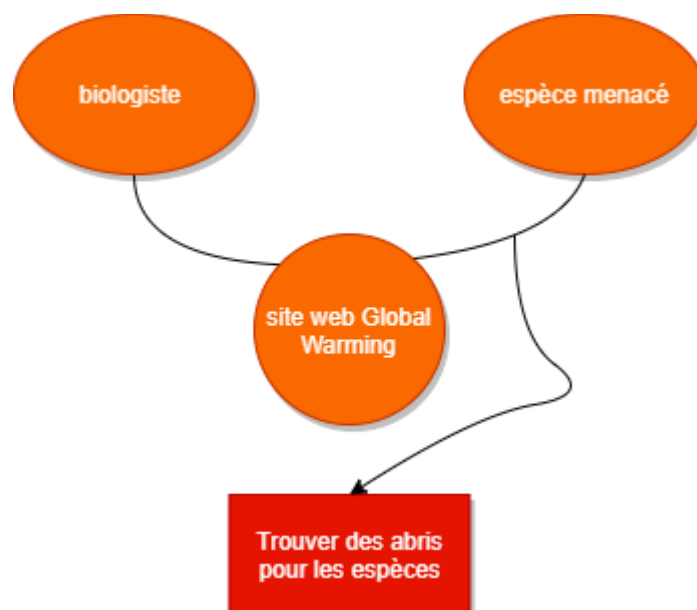
Pour le sujet nous avons décidé d'étudier le réchauffement climatique et de nous intéresser en particulier à la préservation des espèces animales à travers le monde.

Ce projet est dans le prolongement de la méthodologie scientifique vu au semestre 3

1.1.2. Objectif

L'objectif de notre projet « globalWarming » est de proposer aux utilisateurs de visualiser des données sur une carte interactive. Le but de la carte est de localiser des zones qui correspondent au critère de survie d'une espèce.

Voici la bête à corne du projet :



1.1.3. Description

Nous allons développer une application web en utilisant un serveur WAMP, le développement informatique utilisera les langage HTML, CSS, JavaScript et PHP. La base de données utilisera MySQL.

Afin que tout le groupe puisse collaborer sur le projet, nous avons mis en place un gitHub et réaliser un petit tuto pour son utilisation voir en annexe.

Nous avons utilisé une API pour créer nos données à l'aide d'un code python et d'une raspberry pi

1.2. Expression des besoins

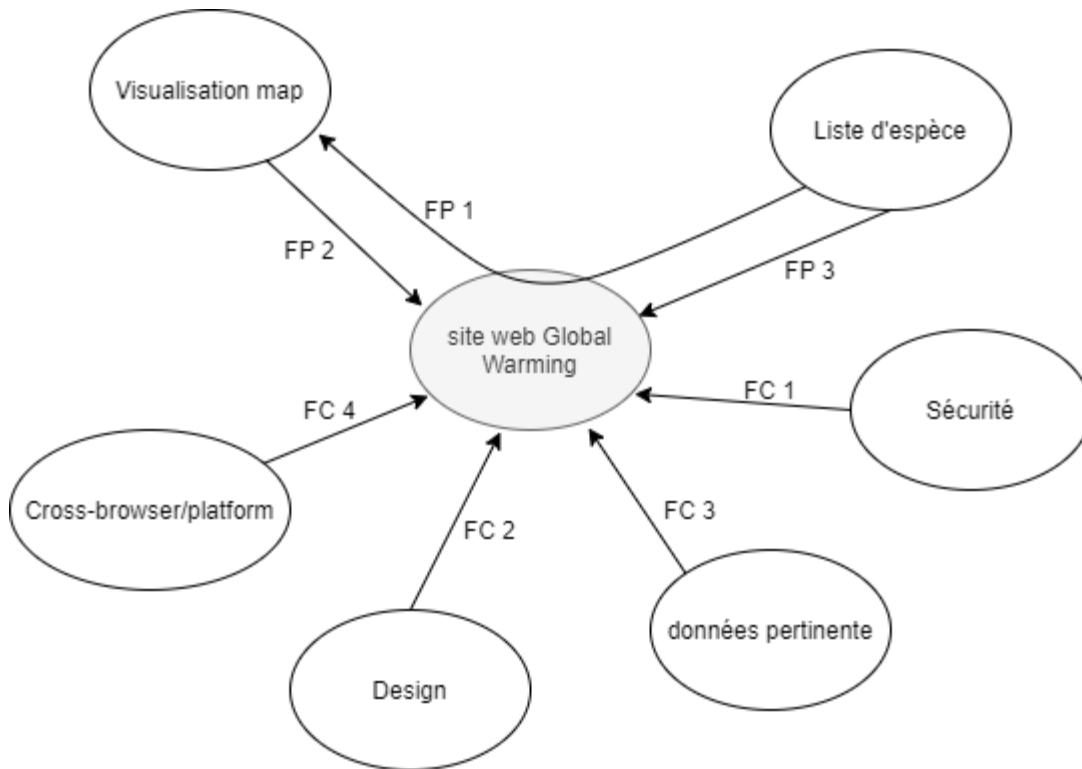
1.2.1. Besoin fonctionnels

Voici une liste des besoins qui seront absolument nécessaire :

- Carte interactive : possibilité de placer des marqueurs sur une carte
- Choisir une espèce : proposer une liste d'espèce à visualiser sur la carte.
- Ajouter une espèce : proposer dans une certaine mesure d'ajouter une. Espèce animale dans la base de données en précisant des critères de survie.
- Inscription : un utilisateur pourra, dans certain cas s'inscrire et obtenir un compte.
- Connexion : un onglet de connexion avec login et password qui vérifie l'authenticité dans la base de données
- Navigation : les utilisateurs pourront naviguer à travers quelque onglet

Note : Il est possible que par la suite d'autres besoins soient implémentés.

Ci-dessous le diagramme fonctionnel (diagramme en pieuvre).



FP 1 : Avoir un choix d'espèce à visualiser sur la carte

FP 2 : Avoir une map interactive avec des marqueurs qui sont placé

FP 3 : Possibilité d'ajouter une espèce

FC 1 : Possède un système d'authentification

FC 2 : Présentation visuelle correcte

FC 3 : Données solide récolté sur 1 an

FC 4 : Disponible sur tout les navigateurs, et différents téléphone portable

1.2.2. Besoins non fonctionnels

- Notre site internet devra être fluide et sans bugs.
- Capable de soutenir un certain trafic.
- Une présentation visuelle correcte.
- Fonctionner sur différents moteurs de recherches.
- Fonctionner sur un navigateur de téléphone portable.

1.3. Contraintes

1.3.1. Délais

En ce qui concerne du délai nous avons à notre disposition 15 séance de 1h30, ce qui fait un total **22h30**.

Mais il est important de rappeler que ce projet demande à travailler en dehors des séances prévu initialement.

La livraison finale du produit été prévu initialement pour fin avril. Mais elle a été reportée au 15 mai à cause du travail à distance causé par le covid19 qui ralenti l'efficacité de l'équipe.

- **Temps total : $4 \times 22,5h = 90h$**

1.3.2. Coûts

Pour le coût du projet, nous ne serons pas rémunérés mais si on doit être payés au smic sur les heures prévu nous devons être payé 175,95 euros (7.82×22.5) pour chacun des collaborateurs. Sachant que nous sommes 4, cela fait 703.80 euros.

Nous avons aussi besoin d'un Raspberry Pi coutant 39.90 euros

(<https://www.kubii.fr/home/1628-nouveau-raspberry-pi-3-modele-b-1-gb-640522710850.html?src=raspberrypi>). Heureusement, le Raspberry Pi a été fourni par la fac.

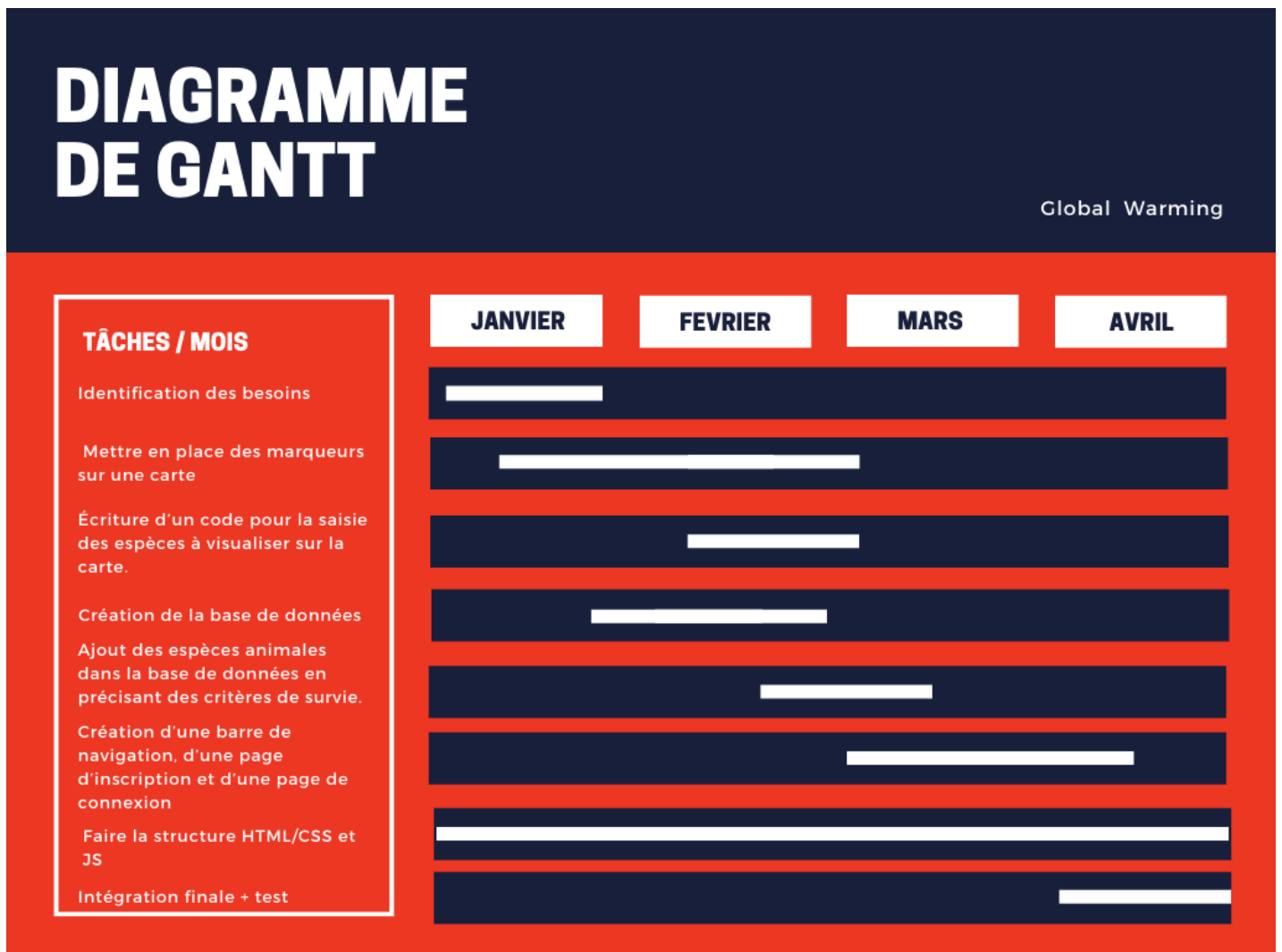
- **Budget total : $90 \times 7.82 + 39.90 = 743,70€$**

1.3.3. Autre contrainte

- Génération de la base de données : données météorologiques sur l'ensemble de la planète sont payante (quelque milliers d'€) donc nécessité de la faire nous même à l'aide d'un script (voir 3.2. Raspberry Pi)
- Pas de nom de domaine : obligation pour les étudiants de travailler sur un projet en local donc 1 seule personne peut développer le site à la fois
- Période de confinement : COVID_19
- Nombre de personnes disponibles pour le développement : 4 développeurs
- Lieu : domicile de l'étudiant donc obligation de télétravail
- Accès internet : parfois défaillant

1.4. Déroulement du projet

Voici la planification que nous avons mis en place avec toute les tâches à accomplir pendant le déroulement du projet :



Nous avons essayé de suivre au mieux l'avancement de projet mais les conditions de travail n'étaient vraiment pas à notre avantage.

2. Modélisation

Avant de passer à la modélisation procédons d'abord à l'identification des différents acteurs et des cas d'utilisation principaux de chacun de ces acteurs.

Les acteurs :

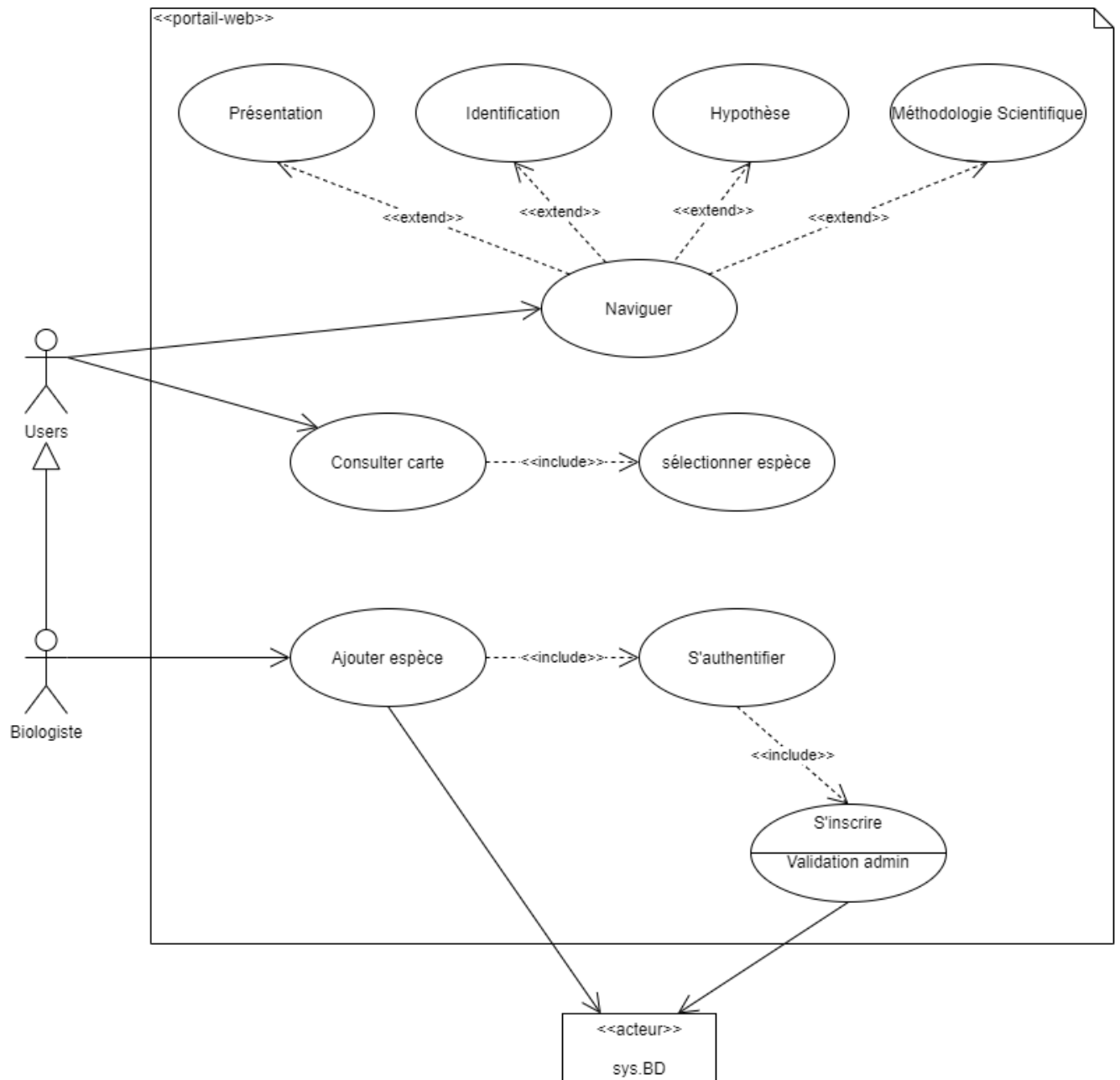
- Utilisateur : consulter carte, sélectionner espèce, naviguer (présentation, identification, hypothèse, méthodologie scientifique)
- Biologiste : hérite d'utilisateur mais avec une fonctionnalité en plus, ajouter espèces mais nécessite de s'authentifier (nécessite inscription).
- Administrateur : il est l'administrateur du système d'information et possède tous les droits, son rôle est de gérer les droits d'accès de tous les autres acteurs afin d'assurer le bon fonctionnement de l'application. Il est également responsable de l'évolution de l'application et de la maintenance : gestion utilisateur, gestion biologiste, maintenance, développement.

Les points importants :

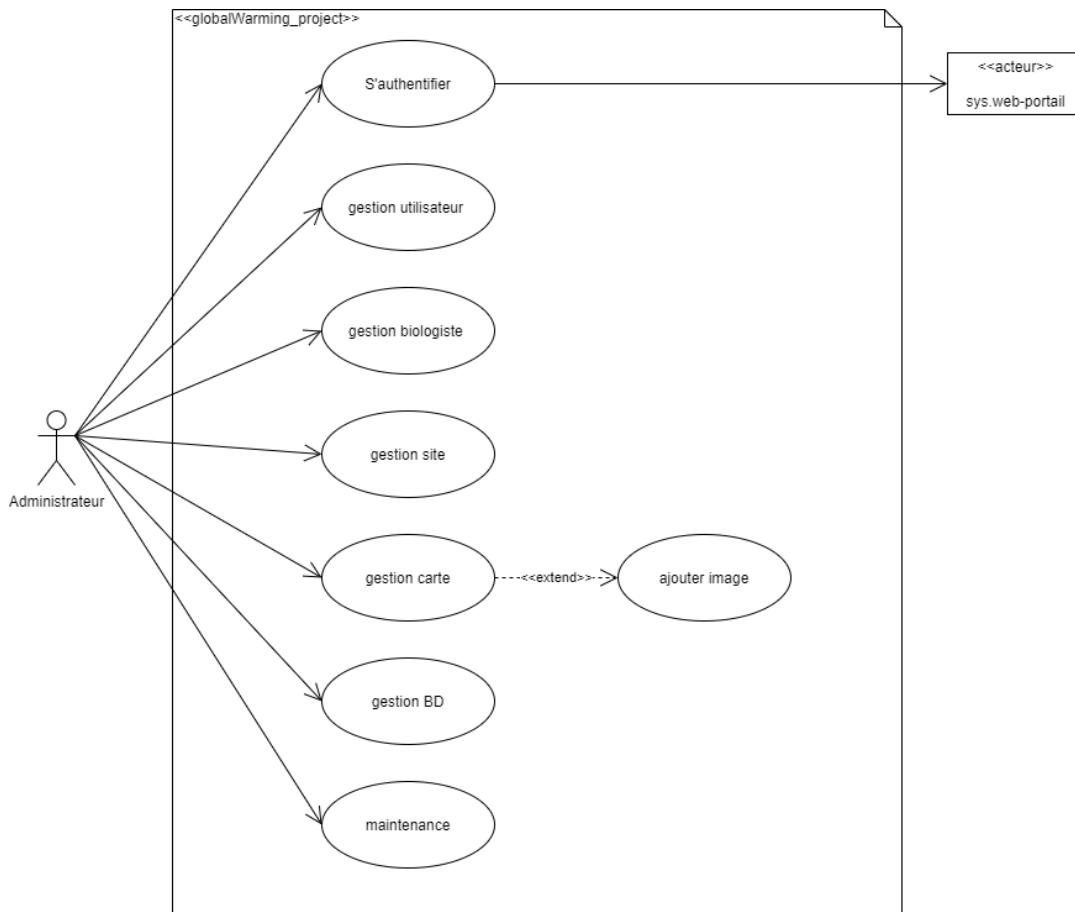
Avant d'accéder à l'application web, le biologiste doit avoir réalisé le processus d'inscription pour pouvoir s'authentifier. Celui-ci devra recevoir une clé d'inscription de notre part après avoir vérifié qu'il s'agit bien d'un biologiste. Ce processus est un processus humain pour l'instant.

Remarque : Afin d'accéder à l'application **le biologiste** devra **s'authentifier** car chacun n'aura pas les mêmes fonctionnalités. Cette authentification permettra d'attribuer les bonnes interfaces à tous les utilisateurs.

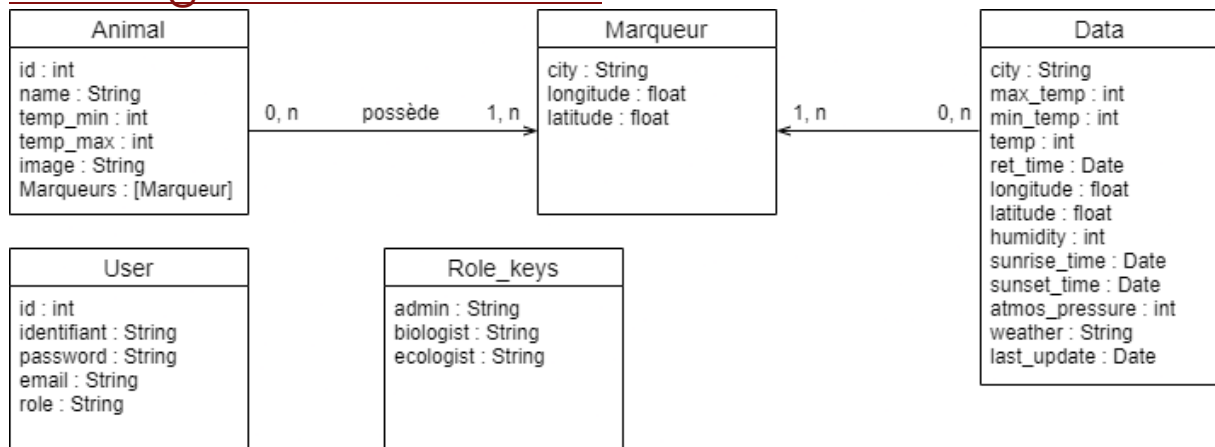
2.1. Diagramme de cas d'utilisation utilisateur



2.2. Diagramme de cas d'utilisation Administrateur



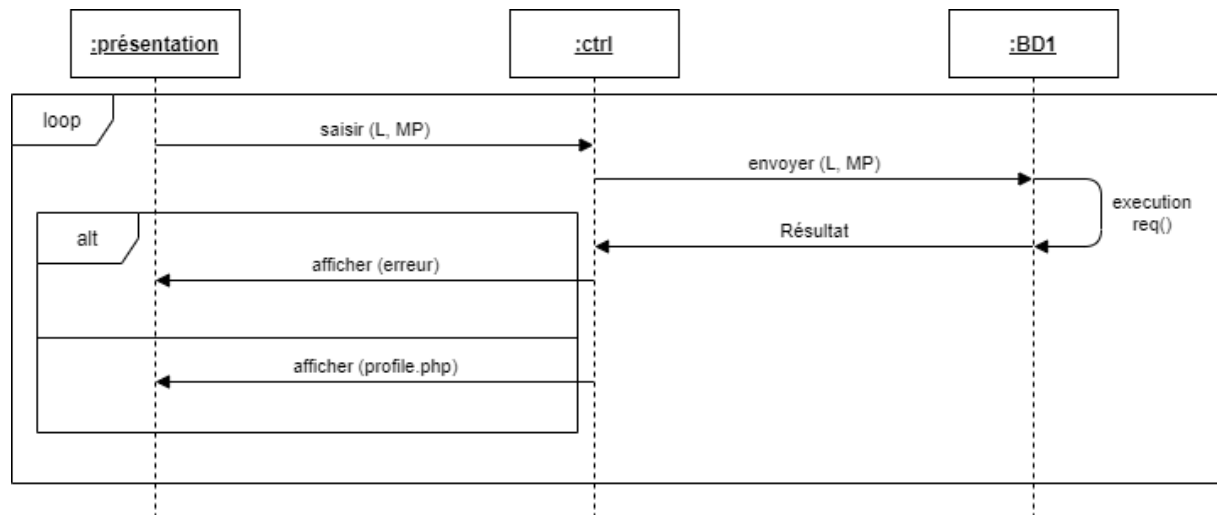
2.3 Diagramme de classe



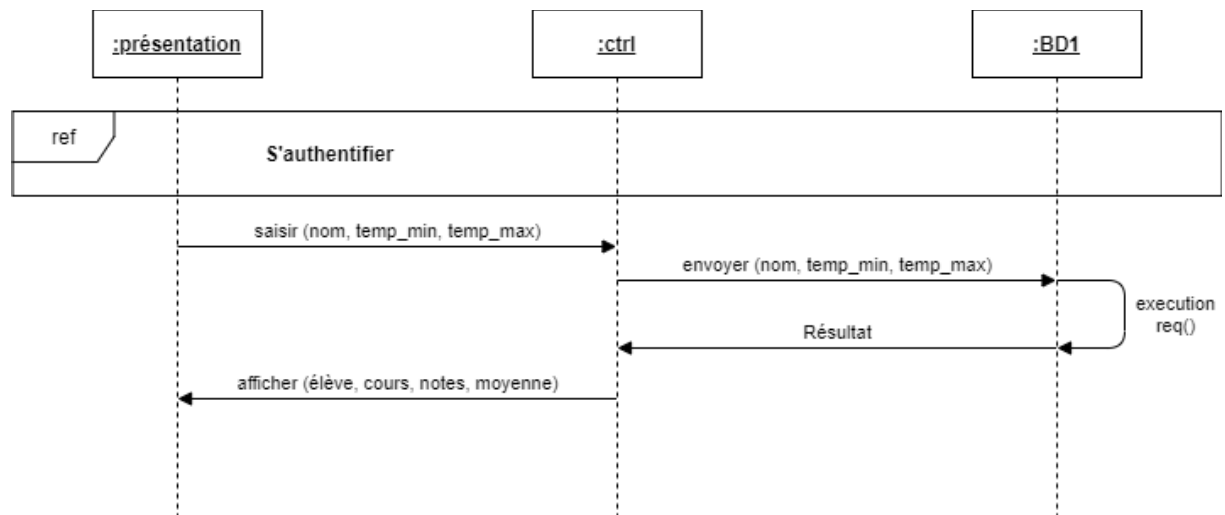
2.4 Diagrammes de séquences

Normalement, il faudrait réaliser un diagramme de séquence pour chaque cas d'utilisation. Mais nous n'allons pas tout développer et nous intéresser aux diagrammes de séquences les plus intéressants ajouter espèce et consulter carte.

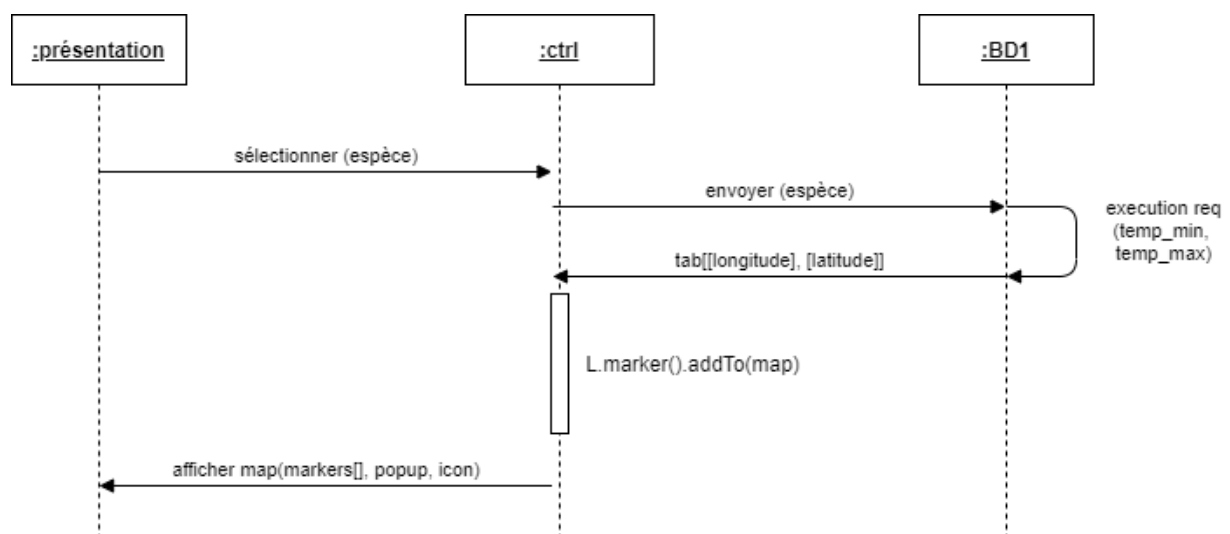
Avant de les réaliser faisons le diagramme de séquence le plus important car il est nécessaire pour chaque acteur, il s'agit du diagramme de séquence s'authentifier :



DS : ajouter une espèce



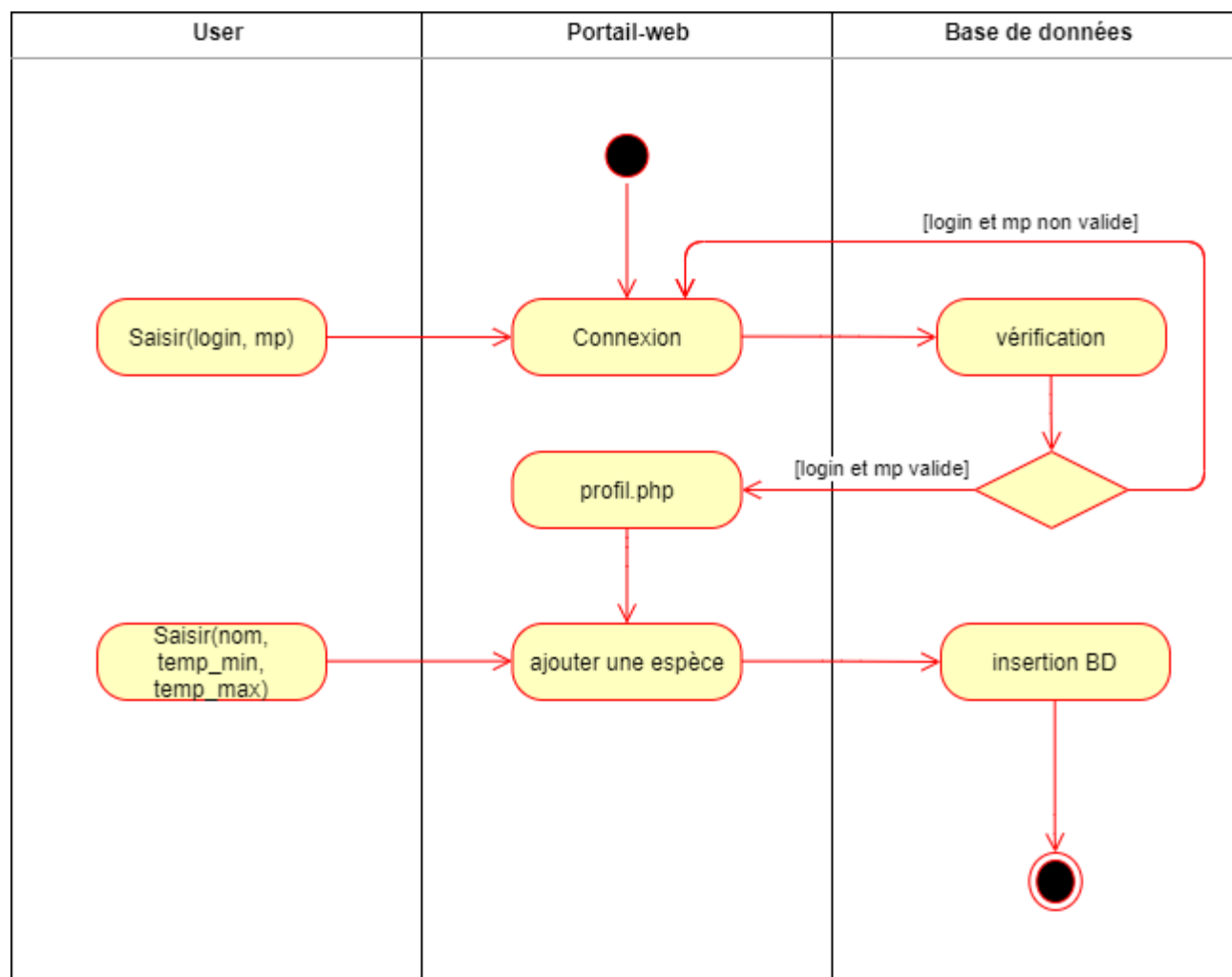
DS : consulter carte



Note : nous laissons ici le choix des noms aux développeurs

2.5. Diagramme d'activité

Nous allons nous intéresser au diagramme d'activité ajouter une espèce :



3. Récolte des données

Pour la récolte des données, nous voulions avoir des données solide et pertinente avec le plus d'informations possible. L'objectif était d'avoir un programme qui lorsqu'il s'exécute, il prend en paramètre une liste de ville et retourne un fichier correspondant avec tous nos objets « Ville » de notre diagramme de classe.

3.1. Choix du langage et utilisation de l'API pyowm

J'ai tout d'abord commencé à coder le programme demandé en C étant donné que nous en avions fait pas mal en L1 et L2. Mais j'ai perdu beaucoup de temps à vouloir optimiser le code en utilisant des pointeurs et en faisant des allocations dynamiques. Une fois le code terminé nous nous sommes rendu compte plus tard que le code avait encore des erreurs sur ubuntu, qui n'était pas affiché sur windows (le code avait été écrit sur windows).

Mais Martin Dieguez Lodeiro nous a filé un coup de main et réalisé la solution en python appelé ret.py. Voici le code :

```
#les chemin complet sont requis pour utiliser cron
filepath = 'places.txt'
import pyowm
import time
import datetime

owm = pyowm.OWM('b7c635aa6d4f700b3ffd7a54f01b4958') # You MUST provide a valid API key
ofile = 'out/data-' + str(datetime.datetime.now()) + ".txt"

f = open(ofile,"w")

with open(filepath) as fp:
    city = fp.readline().rstrip()
    while city:
        obs = owm.weather_at_place(city)
        o = city + ','
        o += str(obs.get_weather().get_temperature(unit='celsius')["temp_max"]) + ','
        o += str(obs.get_weather().get_temperature(unit='celsius')["temp_min"]) + ','
        o += str(obs.get_weather().get_temperature(unit='celsius')["temp"]) + ','
        o += str(obs.get_reception_time(timeformat='iso')) + ','
        o += str(obs.get_location().get_lon()) + ','
        o += str(obs.get_location().get_lat()) + ','
        o += str(obs.get_weather().get_humidity()) + ','
        o += str(obs.get_weather().get_sunrise_time('iso')) + ','
        o += str(obs.get_weather().get_sunset_time('iso')) + ','
        o += str(obs.get_weather().get_pressure()['press']) + ','
        o += str(obs.get_weather().get_detailed_status()) + ','
        o += str(obs.get_weather().get_reference_time(timeformat='iso'))
        o += '\n'
        f.write(o)

        city = fp.readline().rstrip()
        time.sleep(1)
    f.close()
```

On lui donne un fichier contenant une liste de 190 villes, et pour chaque ville il va récupérer 12 informations météorologique dont la température, la longitude et latitude sur la ville en question. Pour la récupération des informations d'une ville, l'api pyowm a été utilisé (doc : <https://pyowm.readthedocs.io/en/latest/index.html>)

Les API :

API est l'acronyme d'**Application Programming Interface**, que l'on traduit en français par **interface de programmation applicative** ou **interface de programmation d'application**.

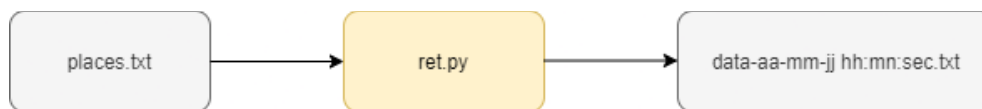
L'API peut être résumée à une solution informatique qui permet à des applications de communiquer entre elles et de s'échanger mutuellement des services ou des données. Il s'agit en réalité d'un ensemble de fonctions qui facilitent, via un langage de programmation, l'accès aux services d'une application. Nous, nous avons utilisé une API REST qui signifie Representational State Transfer. C'est une API réalisant des requêtes HTTP.

Le programme va nous créer un fichier de ce type :

```
Abuja,30.25,30.25,30.25,2020-04-18 10:00:04+00,7.49,9.06,49,2020-04-18  
05:18:34+00,2020-04-18 17:39:37+00,1012,scattered clouds,2020-04-18 09:57:19+00  
Accra,30.0,30.0,30.0,2020-04-18 10:00:06+00,-0.2,5.56,74,2020-04-18 05:52:07+00,2020-  
04-18 18:07:35+00,1012,scattered clouds,2020-04-18 10:00:06+00  
Etc...
```

Avec pour nom la date à laquelle il a été créé.

Diagramme des entrées et sorties du programme :

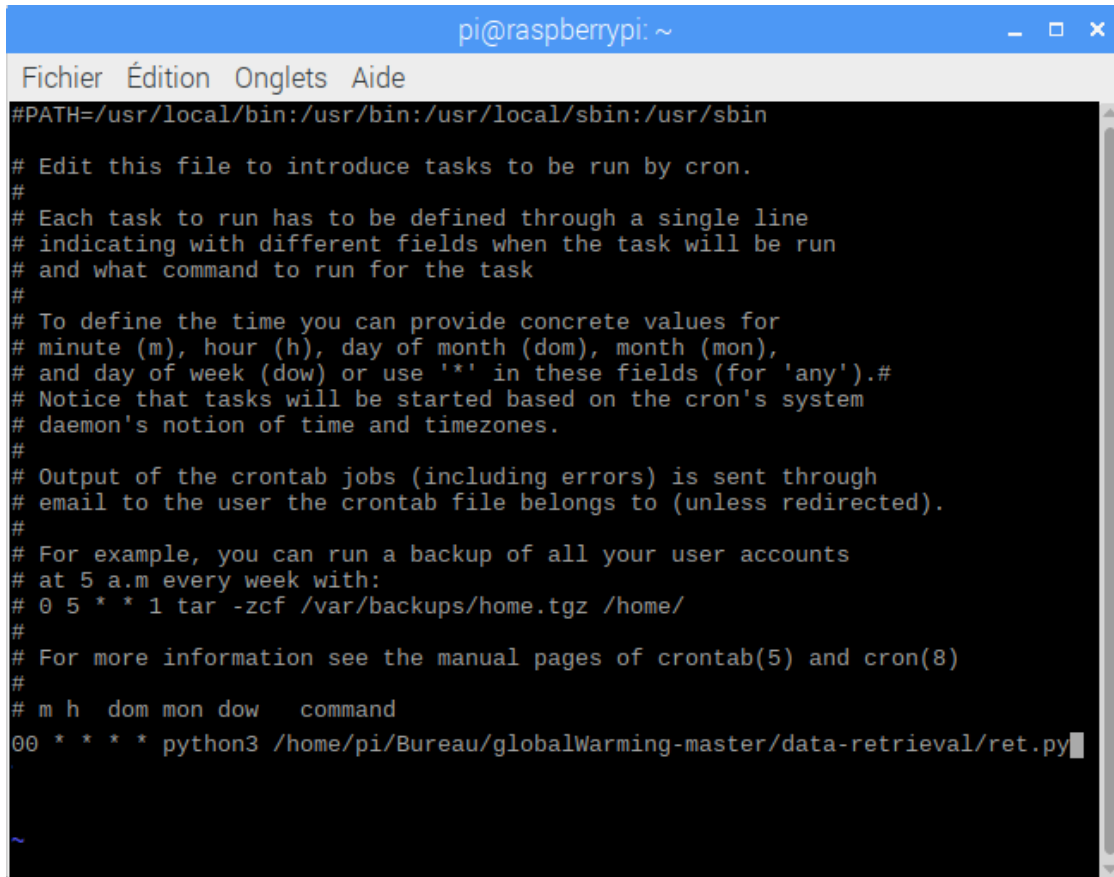


3.2. Raspberry Pi

Pour avoir une masse de données il nous fallait un moyen de lancer notre programme toutes les heures afin d'avoir une analyse des données pertinente par la suite.

Nous nous sommes donc munit d'un raspberry Pi prêté par la fac. Nous avons installé dessus le programme en python et nous avons aussi installé en plus un programme appelé crontab qui permet de lancer des applications de façon régulière.

Pour éditer les actions du fichier crontab, il suffit de faire la commande crontab -e. Voici ce que j'ai écrit dans le fichier pour que le programme ret.py s'exécute toutes les heures :



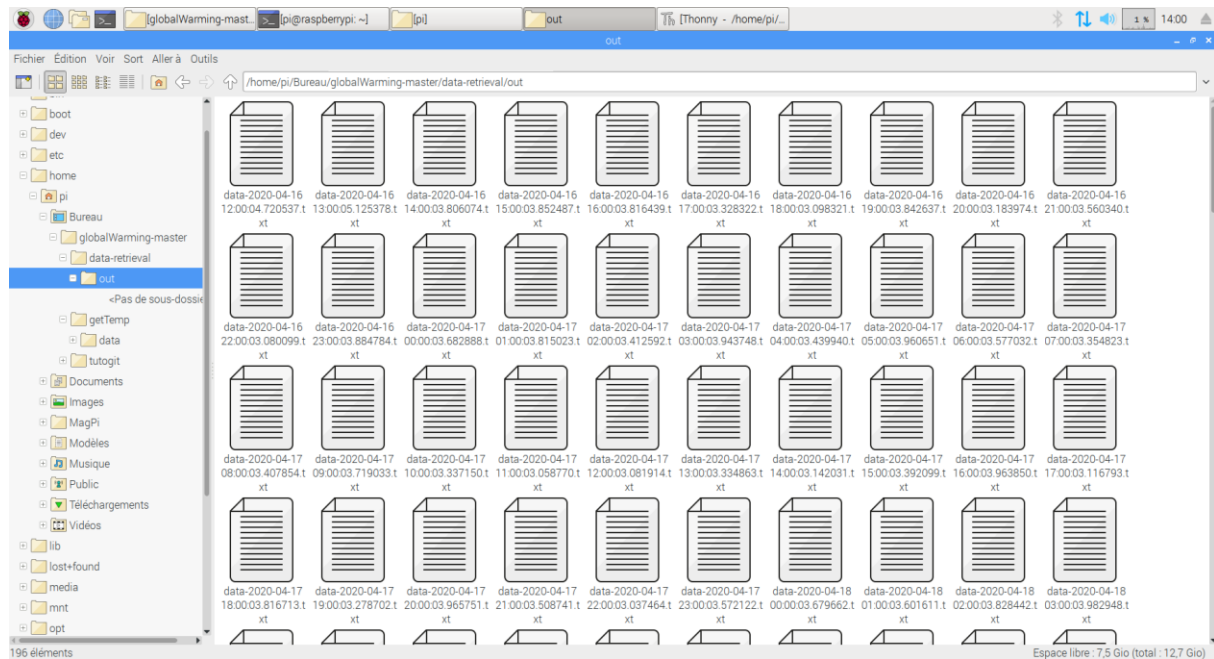
```
pi@raspberrypi: ~
Fichier  Édition  Onglets  Aide
#PATH=/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
00 * * * * python3 /home/pi/Bureau/globalWarming-master/data-retrieval/ret.py
```

Tout ce qui était en commentaire était déjà présent j'ai juste ajouté une ligne. Ici, je demande à ret.py de s'exécuter toutes les heures à 00min. (A noter que si on veut par exemple qu'une commande s'exécute toutes les deux heures il suffit de faire : 00 */2 * * * maCommande)

Bizarrement j'avais un problème, ret.py ne s'exécutait pas. Après mettre renseigner sur internet, j'ai compris qu'il fallait remplacer tous les chemins relatifs par des chemins absolus.

Une fois que le script est sauvegardé, il reste plus qu'une chose à faire, attendre que les données soit récupéré. Plus le programme tourne, plus nos données seront consistantes.

Après plusieurs jours de récolte de données, il fallait qu'à chaque fois je l'envoi sur github pour mes collaborateurs puissent l'importer dans la base de données.



3.3. Amélioration possible

Après avoir travaillé sur cette partie nous sommes rendus compte qu'il y avait quelques modifications nécessaires. Il faudrait faire une concaténation des fichiers du mois et ensuite envoyer les données à la base de données.

Une autre amélioration possible serait d'augmenter le nombre de ville donner au programme. Il est possible d'en trouver assez facilement sur internet.

4. Base De Données

Un fois la récolte des données terminée il fallait les importer dans notre base de données, un procédé complexe où il faut faire très attention à la structure de données. Une erreur et l'exploitation des données perd tout son intérêt.

4.1. Importation des données

Certains fichiers étant vides et d'autres d'une extension invalide il a fallu trier l'ensemble des fichiers textes issus du script de récupération de données.

Une fois tous les fichiers textes correctement triés et rassemblés dans un même dossier, il a fallu tous les concaténer. Pour cela il suffit d'utiliser l'invite de commande avec « **copy *.txt data.txt** ».

Nous obtenons donc en résultat un fichier texte « data.txt » contenant toutes les données récoltées prêtes à être importées dans la base de données à l'aide de phpMyAdmin.

```
Abuja,39.62,39.62,39.62,2020-04-16 11:00:07+00,7.49,9.06,12,2020-04-16 05:19:29+00,2020-04-16 17:39:35+00,1006,few clouds,2020-04-16 11:00:07+00
Accra,33.0,33.0,33.0,2020-04-16 11:00:09+00,-0.2,5.56,59,2020-04-16 05:52:51+00,2020-04-16 18:07:44+00,1010,few clouds,2020-04-16 10:56:24+00
Adamstown,22.81,22.81,22.81,2020-04-16 11:00:11+00,-130.1,-25.07,76,2020-04-16 14:55:48+00,2020-04-17 02:23:49+00,1013,light rain,2020-04-16 11:00:11+00
Algiers,21.0,21.0,21.0,2020-04-16 11:00:12+00,3.04,36.75,73,2020-04-16 05:11:56+00,2020-04-16 18:22:44+00,1019,scattered clouds,2020-04-16 10:56:26+00
Alofi,23.0,23.0,23.0,2020-04-16 11:00:13+00,-169.92,-19.06,94,2020-04-16 17:30:06+00,2020-04-17 05:08:01+00,1013,overcast clouds,2020-04-16 11:00:13+00
Amman,25.0,24.0,24.39,2020-04-16 11:00:15+00,35.95,31.96,27,2020-04-16 03:05:48+00,2020-04-16 16:05:38+00,1013,haze,2020-04-16 10:56:40+00
Amsterdam,20.0,13.33,17.4,2020-04-16 11:00:16+00,4.89,52.37,45,2020-04-16 04:39:48+00,2020-04-16 18:40:05+00,1016,broken clouds,2020-04-16 10:58:37+00
Ankara,13.89,11.11,12.26,2020-04-16 11:00:17+00,32.85,39.92,54,2020-04-16 03:08:50+00,2020-04-16 16:27:24+00,1018,broken clouds,2020-04-16 10:57:18+00
Antananarivo,23.0,23.0,23.0,2020-04-16 11:00:18+00,47.54,-18.91,56,2020-04-16 02:59:59+00,2020-04-16 14:38:45+00,1022,broken clouds,2020-04-16 10:57:13+00
Apia,28.89,28.89,28.89,2020-04-16 11:00:20+00,-171.77,-13.83,79,2020-04-17 17:33:32+00,2020-04-18 05:18:57+00,1016,light rain,2020-04-16 11:00:19+00
Ashgabat,24.0,24.0,24.0,2020-04-16 11:00:21+00,58.38,37.95,38,2020-04-16 01:29:19+00,2020-04-16 14:42:42+00,1011,clear sky,2020-04-16 11:00:20+00
Asmara,21.47,21.47,21.47,2020-04-16 11:00:22+00,38.93,15.33,17,2020-04-16 03:08:56+00,2020-04-16 15:38:40+00,1007,broken clouds,2020-04-16 11:00:22+00
Asunción,11.0,11.0,11.0,2020-04-16 11:00:23+00,-57.64,-25.3,87,2020-04-16 10:06:04+00,2020-04-16 21:33:58+00,1022,clear sky,2020-04-16 11:00:01+00
Athens,17.0,13.89,15.48,2020-04-16 11:00:24+00,23.72,37.98,42,2020-04-16 03:47:47+00,2020-04-16 17:01:28+00,1022,few clouds,2020-04-16 10:55:33+00
Avarua,22.0,22.0,22.0,2020-04-16 11:00:25+00,-159.78,-21.21,83,2020-04-16 16:51:17+00,2020-04-17 04:25:44+00,1014,overcast clouds,2020-04-16 11:00:25+00
Baghdad,30.0,30.0,30.0,2020-04-16 11:00:27+00,44.4,33.34,19,2020-04-16 02:30:32+00,2020-04-16 15:33:18+00,1012,overcast clouds,2020-04-16 10:57:03+00
Baku,17.0,17.0,17.0,2020-04-16 11:00:28+00,49.89,40.38,51,2020-04-16 02:00:08+00,2020-04-16 15:19:48+00,1012,scattered clouds,2020-04-16 10:57:12+00
Bamako,36.75,36.75,36.75,2020-04-16 11:00:29+00,-8.0,12.65,22,2020-04-16 06:18:41+00,2020-04-16 18:44:18+00,1011,scattered clouds,2020-04-16 11:00:28+00
Bangkok,38.33,28.89,33.54,2020-04-16 11:00:30+00,100.52,13.75,59,2020-04-15 23:03:55+00,2020-04-16 11:31:02+00,1007,few clouds,2020-04-16 10:59:49+00
Bangui,31.0,31.0,31.0,2020-04-16 11:00:31+00,18.55,4.36,62,2020-04-16 04:38:45+00,2020-04-16 16:51:51+00,1009,overcast clouds,2020-04-16 11:00:25+00
Banjul,28.11,28.11,28.11,2020-04-16 11:00:32+00,-16.58,13.45,34,2020-04-16 06:52:22+00,2020-04-16 19:19:15+00,1013,broken clouds,2020-04-16 11:00:32+00
Basseterre,25.0,25.0,25.0,2020-04-16 11:00:33+00,-62.73,17.29,78,2020-04-16 09:53:48+00,2020-04-16 22:26:57+00,1017,light rain,2020-04-16 11:00:33+00
Beijing,20.0,15.0,17.05,2020-04-16 11:00:34+00,116.4,39.91,47,2020-04-15 21:34:59+00,2020-04-16 10:52:57+00,1008,few clouds,2020-04-16 11:00:12+00
```

En prenant connaissance de la forme des données, il nous a été possible d'identifier la structure d'une table contenant ces données.

4.2. Structure de données

Notre base de données se compose de 4 tables :

- La table « **data** » qui rassemble les données importées à partir des fichiers textes dont les champs sont :
 - **city** : Correspond au nom de la ville où se trouve la station météo
 - **max_temp** : Température maximale enregistrée
 - **min_temp** : Température minimale enregistrée
 - **temp** : Température moyenne enregistrée

- **ret_time** : Heure de la récupération par le script
 - **longitude** : Longitude
 - **latitude** : Latitude
 - **humidity** : Taux d'humidité
 - **sunrise_time** : Heure du lever du soleil
 - **sunset_time** : Heure du coucher du soleil
 - **atmos_pressure** : Pression atmosphérique
 - **weather** : Météo
 - **last_update** : Date de dernière mise à jour de la station météo
- La table « **animal** » qui rassemble les informations relatives aux différentes espèces dont :
- **id** : Numéro d'identification
 - **name** : Nom de l'animal
 - **temp_min** : Température minimal supportée par l'animal
 - **temp_max** : Température maximale supportée par l'animal
 - **image** : Nom et extension du fichier contenant une image de l'animal
- La table « **membres** » qui rassemble les informations liées aux utilisateurs :
- **id** : Numéro d'identification
 - **identifiant** : Identifiant de l'utilisateur
 - **password** : Mot de passe de l'utilisateur
 - **email** : Adresse email de l'utilisateur
- La table « **role_keys** » qui comporte les clés permettant de s'inscrire avec un rôle spécifique :
- **rôle** : Correspond au nom du rôle
 - **role_key** : Correspond à une chaîne de caractère générées au préalable avec des requêtes SQL comme : (« INSERT INTO `role_keys` (`rôle`, `role_key`) VALUES ("admin", (SELECT SUBSTRING(MD5(RAND()) FROM 1 FOR 8))) »)

city	max_temp	min_temp	temp	ret_time	longitude	latitude	humidity	sunrise_time	sunset_time	atmos_pressure	weather	last_update
Abuja	34.7	34.7	34.7	2020-04-16 19:00:04	7.49	9.06	12	2020-04-16 05:19:29	2020-04-16 17:39:35	1004	few clouds	2020-04-16 19:00:03
Accra	30	30	30	2020-04-16 19:00:05	-0.2	5.56	84	2020-04-16 05:52:51	2020-04-16 18:07:44	1007	broken clouds	2020-04-16 19:00:02
Adamstown	22.78	22.78	22.78	2020-04-16 19:00:06	-130.1	-25.07	77	2020-04-16 14:55:48	2020-04-17 02:23:49	1015	light rain	2020-04-16 19:00:06
Algiers	18	18	18	2020-04-16 19:00:09	3.04	36.75	88	2020-04-16 05:11:56	2020-04-16 18:22:44	1015	scattered clouds	2020-04-16 19:00:08
Alofi	24	24	24	2020-04-16 19:00:11	-169.92	-19.06	94	2020-04-16 17:30:06	2020-04-17 05:08:01	1014	overcast clouds	2020-04-16 19:00:10
Amman	16	16	16	2020-04-16 19:00:12	35.95	31.96	55	2020-04-16 03:05:48	2020-04-16 16:05:38	1011	few clouds	2020-04-16 18:57:59
Amsterdam	17.78	10.56	14.04	2020-04-16 19:00:13	4.89	52.37	54	2020-04-16 04:39:48	2020-04-16 18:40:05	1016	broken clouds	2020-04-16 18:59:11
Ankara	12	8	10.02	2020-04-16 19:00:14	32.85	39.92	54	2020-04-16 03:08:50	2020-04-16 16:27:24	1018	scattered clouds	2020-04-16 18:59:01
Antananarivo	17	17	17	2020-04-16 19:00:15	47.54	-18.91	82	2020-04-16 02:59:59	2020-04-16 14:38:45	1023	few clouds	2020-04-16 19:00:15
Apia	28.22	28.22	28.22	2020-04-16 19:00:16	-171.77	-13.83	81	2020-04-17 17:33:32	2020-04-18 05:18:57	1012	moderate rain	2020-04-16 19:00:16
Ashgabat	17	17	17	2020-04-16 19:00:17	58.38	37.95	63	2020-04-17 01:27:56	2020-04-17 14:43:38	1012	clear sky	2020-04-16 18:58:25
Asmara	14.12	14.12	14.12	2020-04-16 19:00:19	38.93	15.33	28	2020-04-16 03:08:56	2020-04-16 15:38:40	1010	clear sky	2020-04-16 19:00:18
Asunción	26	26	26	2020-04-16 19:00:20	-57.64	-25.3	31	2020-04-16 10:06:04	2020-04-16 21:33:58	1023	clear sky	2020-04-16 19:00:02
Athens	15.56	10	13.04	2020-04-16 19:00:21	23.72	37.98	35	2020-04-16 03:47:47	2020-04-16 17:01:28	1019	clear sky	2020-04-16 18:57:31
Avarua	24	24	24	2020-04-16 19:00:22	-159.78	-21.21	83	2020-04-16 16:51:17	2020-04-17 04:25:44	1014	overcast clouds	2020-04-16 19:00:22
Baghdad	25	25	25	2020-04-16 19:00:23	44.4	33.34	27	2020-04-16 02:30:32	2020-04-16 15:33:18	1010	scattered clouds	2020-04-16 18:59:33
Baku	9	9	9	2020-04-16 19:00:24	49.89	40.38	76	2020-04-16 02:00:08	2020-04-16 15:19:48	1018	overcast clouds	2020-04-16 18:58:44
Bamako	37	37	37	2020-04-16 19:00:26	-8	12.65	29	2020-04-16 06:18:41	2020-04-16 18:44:18	1011	few clouds	2020-04-16 19:00:25
Bangkok	31.11	27.78	29.5	2020-04-16 19:00:27	100.52	13.75	79	2020-04-16 23:03:20	2020-04-17 11:31:10	1009	scattered clouds	2020-04-16 18:55:38
Bangui	27	27	27	2020-04-16 19:00:28	18.55	4.36	78	2020-04-16 04:38:45	2020-04-16 16:51:51	1005	thunderstorm	2020-04-16 18:58:20
Banjul	28	28	28	2020-04-16 19:00:29	-16.58	13.45	54	2020-04-16 06:52:22	2020-04-16 19:19:15	1010	clear sky	2020-04-16 18:59:50
Bassterre	30	28	28.73	2020-04-16 19:00:30	-62.73	17.29	69	2020-04-16 09:53:48	2020-04-16 22:26:57	1017	shower rain	2020-04-16 19:00:29
Beijing	13.89	10	12.44	2020-04-16 19:00:31	116.4	39.91	81	2020-04-16 21:33:31	2020-04-17 10:53:58	1010	scattered clouds	2020-04-16 18:56:39
Beirut	22	8.33	16.5	2020-04-16 19:00:32	35.49	33.89	46	2020-04-16 03:05:32	2020-04-16 16:09:34	1009	scattered clouds	2020-04-16 18:56:50
Belgrade	15	12	13.62	2020-04-16 19:00:33	20.47	44.8	41	2020-04-16 03:51:20	2020-04-16 17:23:55	1017	scattered clouds	2020-04-16 18:55:57

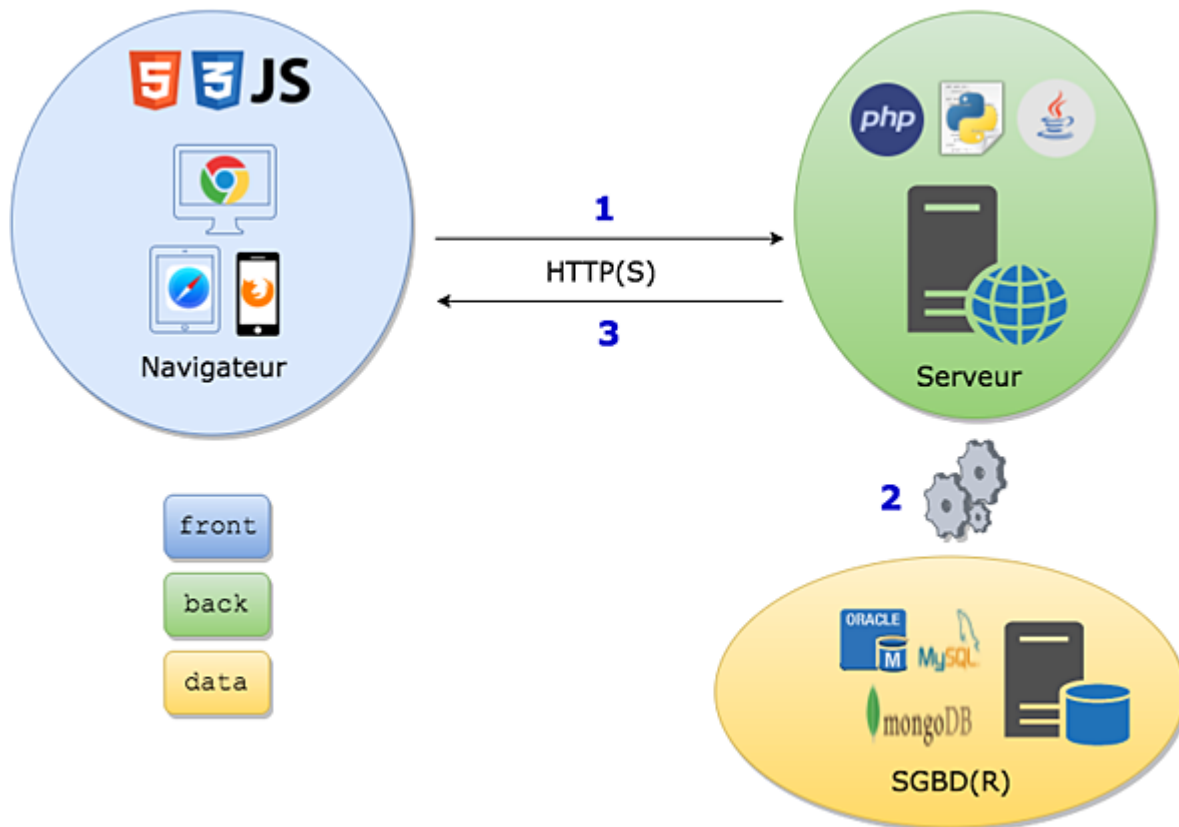
Cette base de données est exploitée grâce à différentes requêtes SQL transmises par le PHP. Tous les champs de la table « data » ne seront pas exploités mais cela pourra être amené à évoluer en fonction de fonctionnalités ajoutées plus tard.

Pour avoir un aperçu de la base de données, vous pouvez la trouver en annexe (Structure base de données :)

5. Application Web

5.1. Structure du site globalWarming

Avant de rentrer dans les détails notre application va suivre la structure suivante :



Afin de rendre l'utilisation de ce projet plus aisée, nous avons pensé à programmer en HTML, CSS, PHP et JavaScript une interface Web pour l'affichage. Celui-ci nous permettra donc de choisir une espèce, de visualiser la carte qui nous montre où l'espèce choisie peut survivre, d'ajouter une espèce ainsi que de se connecter (à venir). L'interface se compose classiquement d'une balise "head", d'une balise "body", et d'une balise "footer".

La balise "head" nous permet d'importer les fichiers CSS et les scripts de notre application vers l'interface, ce qui nous permet d'appeler les fonctions associées à ces scripts JS lorsqu'on interagit avec l'interface Web.

La balise "body", quant à elle, contient des sous-balises. On distingue deux sous-parties, qui sont le bloc des commandes et le bloc graphique. Le bloc des commandes va justement nous permettre d'interagir avec notre interface, pour pouvoir appeler des fonctions contenues dans les scripts JS importés. On peut donc tout d'abord sélectionner une espèce et par la suite afficher la carte dans une nouvelle page. Nous avons également la possibilité d'ajouter des espèces et les

températures minimales et maximales où elles peuvent survivre, à savoir qu'il faut être connecter.

La barre de navigation contient 5 volets, un volet inscription, où les administrateurs et les biologistes peuvent s'inscrire, grâce à une clé d'invitation, ces derniers se connecte ensuite dans le volet connexion. Les deux onglet Global Warning et Recherche scientifique contiennent à la suite notre rapport et une brève explication en format PDF, et enfin le volet home qui est le volet principal.

Home Méthodologie Scientifique Global Warning

Connexion Inscription

Réussir l'affichage graphique était un défi, c'est pour cela qu'on a opté pour un « MagicScroll », une fonction qui joue la vidéo en arrière-plan en rapport des défilements de souris fait par l'utilisateur, cela veut dire que une fois l'utilisateur arrête de défiler sa souris la vidéo s'arrêtera. Cette fonction est très utilisée dans le site du géant Apple par exemple. Si l'utilisateur veut ignorer cette partie lors de sa prochaine visite, il peut juste cliquer sur le bouton rouge intitulé : « Défiler en bas ou cliquer ici »

Pour ce faire, on a téléchargé des librairies d'internet.

```
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/
  ScrollMagic/2.0.7/ScrollMagic.min.js"
  integrity="sha256-2p2tRZlPowp3P/04Pw2rqVCSbhyV/IB7ZEVUglrDS/c="
  crossorigin="anonymous"
</script>
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/
  ScrollMagic/2.0.7/plugins/animation.gsap.js"
  integrity="sha256-peenofh8a9TIqKdPKIeQE7mJvuwh+J0To7nslvpj1jI="
  crossorigin="anonymous"
</script>
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/
  ScrollMagic/2.0.7/plugins/debug.addIndicators.js"
  integrity="sha256-31FC/0T6XpfjAhj9FuXjw5/wPXXawCAjJQ29E23/XPk="
  crossorigin="anonymous"
</script>
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/
  gsap/2.1.3/TweenMax.min.js"
  integrity="sha256-lPE3wjN2a7ABWHbGz7+MKBJaykyzqCbU96BJWjio86U="
  crossorigin="anonymous"
</script>
```

Ensuite on insert le code dans le fichier dans app.js

5.2. Incorporation de la map Leaflet (open street map)

Pour l'implémentation d'une carte interactive, il a fallu choisir une API à utiliser. L'API de Google Maps était une possibilité mais nous avons préféré utiliser Leaflet, une librairie JavaScript open-source permettant la création de cartes interactives.

Nous avons ensuite suivi le tutoriel proposé par Leaflet et nous nous sommes aidés de la documentation pour créer la carte.

Nous avons commencé par inclure le CSS et le JavaScript de Leaflet dans le fichier « map.php » :

```
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.6.0/dist/leaflet.css"
integrity="sha512-xwE/Az9zrjBIPhAcBb3F6JVqxf46+CDLwflMHl0Nu6KEQCAWi6HcDUBE0fBIPtF7tcCzusKFjFw2yuvEpDL9wQ=="
crossorigin="" />
<script src="https://unpkg.com/leaflet@1.6.0/dist/leaflet.js"
integrity="sha512-gZwIG9x3wUXg2hdXF6+rVKLF6/Ovi9U8D2Ntg4Ga5I5BZpVxVlJWbSQtXPSiUTtC0TjtG0mxa1AJPUV0CPthw=="
crossorigin=""></script>
```

Pour « déclarer » la carte nous avons inclut le code suivant :

```
<div id="mapid">
<script type="text/javascript">
var mymap = L.map('mapid').setView([51.505, -0.09], 3);
L.tileLayer('https://api.mapbox.com/styles/v1/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
attribution: 'Map data &copy; <a href="https://www.openstreetmap.org/">OpenStreetMap</a> contributors, <a href="https:
maxZoom: 18,
id: 'mapbox/streets-v11',
tileSize: 512,
zoomOffset: -1,
accessToken: 'pk.eyJ1Ijo1Ym10ZTEyMyIsImEiOiJjajZlczJqMw0wNndoM2VzMVtZnlqMTJtIn0.G5rZHbpvLkxLChw0NIUc-Q'
}).addTo(mymap);
```

On déclare la carte avec L.map() et setView() nous permet de définir le centre de la carte lors de son chargement.

Ensuite, c'est là que nous pouvons paramétrer plus en détails la carte lors de son affichage :

Les différents attributs à spécifier permettent le bon chargement de l'API : de maxZoom à accessToken. Les valeurs indiquées viennent compléter l'URL de L.tileLayer() :

- **maxZoom** permet de définir le zoom maximal que peut effectuer l'utilisateur sur la carte.
- **Id** correspond au fond de carte utilisé pour la carte interactive.
- **tileSize** correspond à la taille de la grille de la carte.
- **zoomOffset** permet de modifier le zoom lors du chargement de la carte.
- **accessToken** correspond au token d'accès que nous avons dû obtenir en créant un compte sur le site de MapBox.

Pour que la carte s'affiche sur toute la page il a fallu rajouter dans un fichier « map.css » tout en l'incluant dans le « map.php » :

```
html, body {
height: 100%;
}
#mapid {
width: 100%;
height: 100%;
}
```

Nous avons ensuite ajouté le code php relatif à la base de données :

```
var longitude = [];
var latitude = [];
<?php
$globalwarming = new PDO('mysql:host=localhost;dbname=globalwarming;charset=utf8', 'root', '');
$verif = $globalwarming->query('SELECT temp_min, temp_max FROM animal WHERE name LIKE "'.$_POST['saisieMode'].'"');
$animal = $verif->fetch();
$verif = $globalwarming->query('SELECT longitude, latitude FROM data WHERE 1 GROUP BY city HAVING AVG(temp) BETWEEN "'.$animal['temp_min'].'" AND "'.$animal['temp_max'].'"');
while($d = $verif->fetch()){
longitude.push(<?php echo $d['longitude']; ?>);
latitude.push(<?php echo $d['latitude']; ?>);
}
<?php }
```

La première requête permet de récupérer les températures minimales et maximales de l'animal que l'utilisateur a choisi sur la page précédente.

La seconde requête permet de récupérer les longitudes et latitudes des enregistrements par ville dont la moyenne de température se trouve entre la température minimale et la température maximale supportées par l'animal choisit par l'utilisateur sur la page précédente.

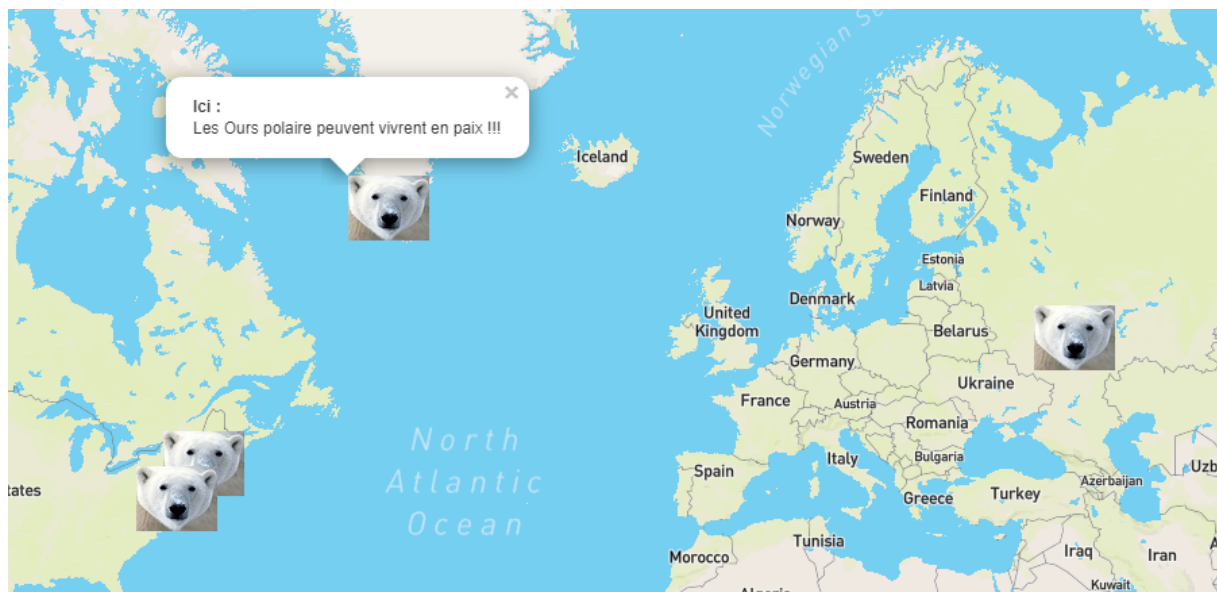
On récupère toutes les valeurs de longitude et latitude retournées dans deux tableaux avec une boucle while().

On va ensuite parcourir ces deux tableaux pour afficher tous les marqueurs aux longitudes et latitudes correspondantes à l'aide d'une boucle for et des fonctions liées à Leaflet :

```
for (var i = longitude.length - 1; i >= 0; i--) {  
  if ("<?php echo getAnimal('image') ?>" == "")  
    var marker = L.marker([latitude[i], longitude[i]]).addTo(mymap);  
  else  
    var marker = L.marker([latitude[i], longitude[i]], {icon: icon}).addTo(mymap);  
  marker.bindPopup("<b>Ici :</b><br>Les "+"<?php echo getAnimal('name') ?>"+ " peuvent vivre en paix !!!").openPopup();  
}
```

L'API de Leaflet est assez complète et nous permettra d'ajouter de nouvelles fonctionnalités sur la carte interactive à l'avenir.

En ajoutant à tout cela quelques fonctions PHP (complexe, voir le code map.php) nous avons réussi à obtenir un résultat plutôt satisfaisant :

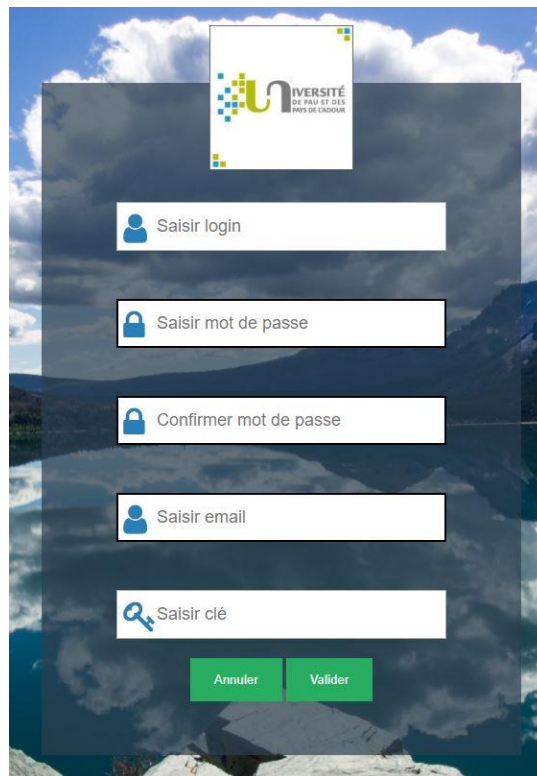


Pour plus d'information à propos de Leaflet vous pouvez consulter la documentation à l'adresse suivante : <https://leafletjs.com/reference-1.6.0.html>

5.3. Les interfaces

Le site du projet propose une interface de connexion et d'inscription. Pour pouvoir ajouter une espèce, il est nécessaire d'être connecté. Nous pourrions à l'avenir ajouter des fonctionnalités réservées aux utilisateurs connectés, ou à ceux qui ont un rôle précis. Un utilisateur non connecté a accès à deux liens : connexion et inscription.

5.3.1. Interfaces d'inscription

The image shows a registration form for the University of Pau. The form is a semi-transparent white box with a dark blue border, centered over a background image of a mountain range under a cloudy sky. At the top of the form is the University of Pau logo, which consists of a stylized 'U' made of green and blue squares, followed by the text 'UNIVERSITÉ DE PAU ET DES PAYS DE L'ADOUR'. Below the logo are five input fields, each with a small icon on the left: a person icon for 'Saisir login', a lock icon for 'Saisir mot de passe', another lock icon for 'Confirmer mot de passe', an email icon for 'Saisir email', and a key icon for 'Saisir clé'. At the bottom of the form are two green buttons: 'Annuler' on the left and 'Valider' on the right.

L'interface d'inscription demande de saisir un **login**, un **mot de passe** et sa **confirmation**, une **adresse email** et une **clé**. Cette clé est nécessaire pour s'inscrire et l'inscription attribue un rôle à l'utilisateur en fonction de cette dernière.

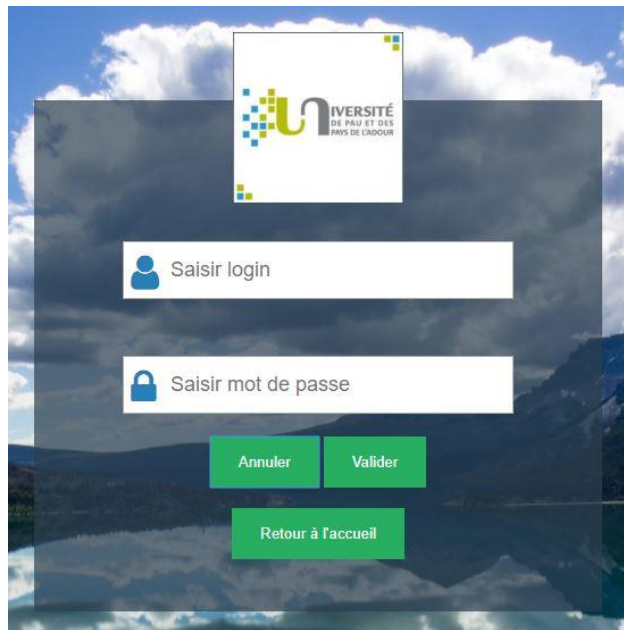
Un message d'erreur est affiché si la clé est invalide ou si les deux champs de mot de passe sont différents.

Une fois que le formulaire est validé, la base de données va réaliser deux choses :

- Vérifier si la clé est valide et si c'est le cas, décider quel rôle va être attribué à l'utilisateur.
- Insérer les informations du nouvel utilisateur muni du rôle en lien avec la clé entrée.

Après la fin et la confirmation de l'inscription, l'utilisateur peut revenir à l'accueil ou se rendre sur la page de connexion à l'aide de boutons.

5.3.2. Interfaces de connexion



L'interface de connexion demande un login et un mot de passe. Lorsque l'utilisateur valide le formulaire, ce dernier envoie les informations à une page « verification.php » où la base de données se charge ensuite de vérifier si les identifiants correspondent puis valide ou non la connexion.

Si les identifiants ne sont pas valides, la page affiche alors une erreur. Si les identifiants sont valides, la page va ensuite

5.3.3. Les sessions PHP

Pour détecter la connexion ou la non-connexion d'un utilisateur sur n'importe quelle page, on utilise les sessions PHP. On doit ajouter sur chaque page PHP qui doit détecter la connexion (ou la session) une ligne de code permettant de démarrer la session : **session_start()**.

```
<?php
session_start();
?>
```

Dans notre projet, les sessions ne sont utilisées que sur les pages « verification.php », « index.php » et « deconnexion.php ».

Pour véritablement savoir si l'utilisateur s'est connecté durant la session, on utilise les variables superglobales de session que nous avons déclarées dans « verification.php » lors de la connexion de l'utilisateur :

```
$response = $db->query(' SELECT COUNT(id) FROM membres WHERE ((identifiant LIKE "'. $ _POST['login']. "') AND (password LIKE "'. $ _POST['password']. "') ' );
$data = $response->fetch();
if($data['COUNT(id)'] == 1) {
    echo "Connexion autorisée";
    session_start();
    $_SESSION['login'] = $_POST['login'];
    $_SESSION['role'] = $_POST['role'];
    header('Location: index.php');
```

Ici on vérifie grâce à la base de données si un utilisateur à ces identifiants existe. La connexion étant autorisée, on démarre la session puis on utilise deux variables superglobales de session `$_SESSION['login']` pour y stocker le login qui a été entré dans le formulaire de connexion puis `$_SESSION['role']` pour y stocker le rôle.

Ces variables sont utilisables sur n'importe quelle page où une session a été démarrée. Nous les avons donc utilisées sur la page « index.php » :

```
<?php
if (isset($_SESSION['login'])) {
    echo '<script type="text/javascript">document.getElementById("envoyer").disabled= false;</script>';
} else {
    echo '<script type="text/javascript">document.getElementById("envoyer").disabled= true;</script>';
}
?>
```

On utilise ici la fonction **isset()** pour savoir si dans la session en cours la variable `$_SESSION['login']` est définie, ce qui est possible seulement par la connexion autorisée de l'utilisateur. Si c'est bien le cas, on exécute une ligne de JavaScript pour activer le bouton permettant d'envoyer le formulaire d'ajout d'espèces. Si la variable n'est pas définie, c'est que l'utilisateur ne s'est pas connecté dans sa session et donc on désactive le bouton « Envoyer ».

Dans le même principe, si l'utilisateur est connecté, les liens « Inscription » et « Connexion » disparaissent et un lien « Déconnexion » apparaît. Lorsqu'on clique sur « Déconnexion », on est redirigé vers la page « deconnexion.php » qui exécute le code suivant :

```
<?php
session_start();
session_destroy();
header("Location: index.php");
?>
```

La page de déconnexion va donc démarrer la session pour ensuite la « détruire » avec la fonction **session_destroy()**. Ainsi, la session disparaît ainsi que toutes les variables superglobales déclarées jusqu'à alors. La page redirige ensuite vers la page d'accueil à l'aide de la fonction **header()**.

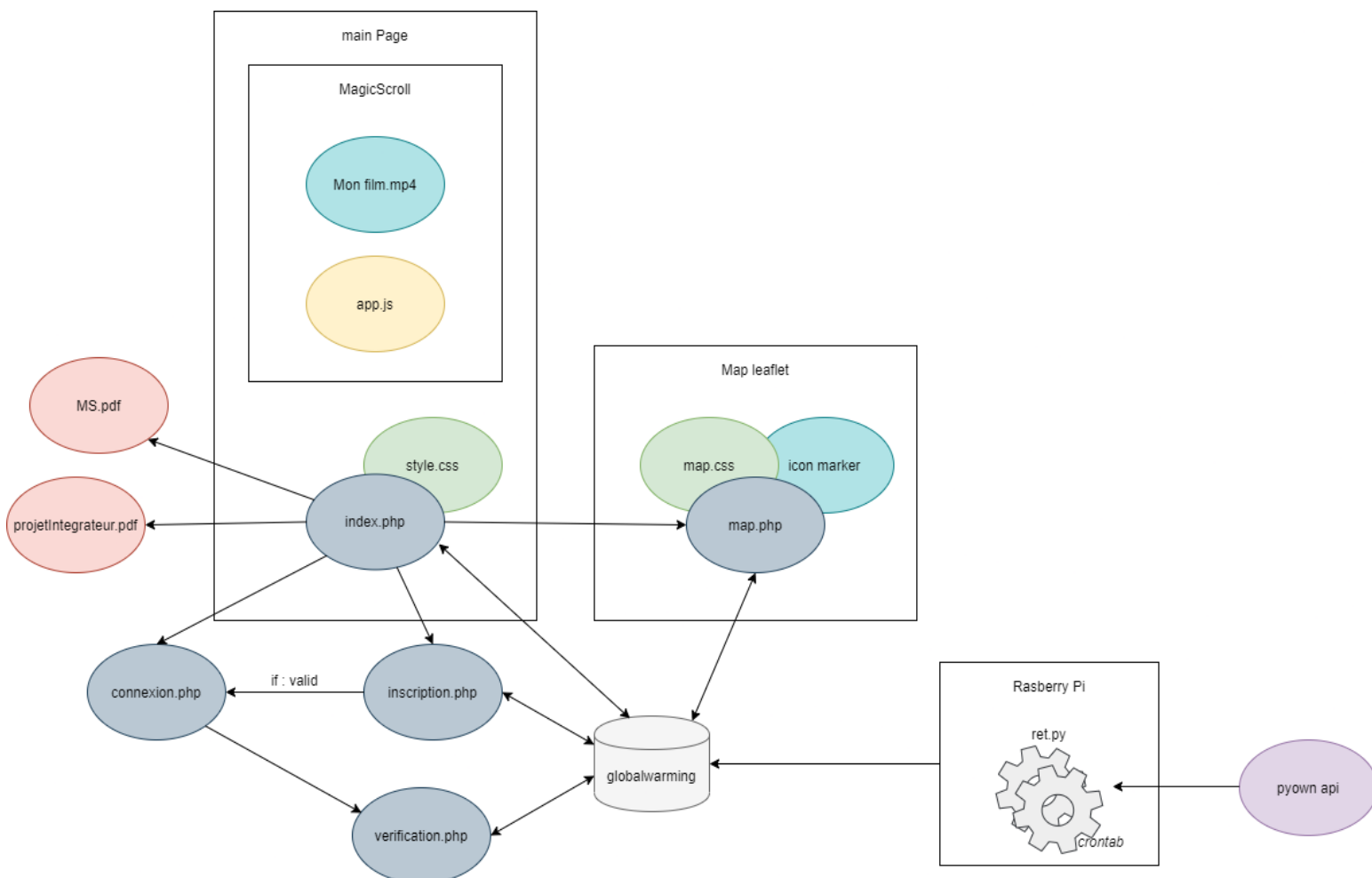
6. Conclusion

Lors de ce projet intégrateur nous avons eu l'occasion de manier beaucoup de langage de programmation, différentes api et outils :

- Langage : HTML, CSS, JavaScript, PHP, SQL, Python, C,
- API : pyowm, Leaflet, OpenStreetMap
- Outils : WAMP server, crontab, PHPmyAdmin, gitHub
- Matériel : Raspberry Pi

Cela était une bonne expérience pour nous de faire communiquer différent langage entre eux notamment la partie web qui a été un casse-tête.

Voici la structure final de notre application web :



Nous avons réussi malgré l'épidémie de covid19 à rendre dans les temps un produit respectant les besoins fonctionnels qu'on s'était fixé. Nous avons aussi dû apprendre à travailler uniquement en télétravail.

Le projet est vraiment intéressant et de nombreuses fonctionnalités peuvent y être ajoutées, ce n'est que le début d'un potentiel infini !

7. Perspective dévolution

Voici une rapide liste de ce qui pourrait être implémenté sur le site dans le futur, de nombreuses fonctionnalités toutes plus intéressantes les unes que les autres :

- Mettre en corrélation les données actuelles avec des données de températures historiques pour réaliser une **analyse prédictive** et définir, pour une zone, un **potentiel de pérennité** pour la survie de l'espèce. Cette corrélation permettra également d'étudier l'évolution du réchauffement climatique au fil du temps et de créer des **modèles statistiques** (Processus de data mining : voir annexe) afin de créer des **connaissances**. Ces interprétations sont une des préoccupations les plus importantes du siècle.
- Nous avons pu nous apercevoir que l'obtention des bases de données de température à la surface du globe est extrêmement coûteuse. Comme notre **application génère** cette base de données au fil du temps, après plusieurs années de mise en service, cette base deviendrait très riche et est intéressante. Pour le point de vue financier de notre application, nous pourrions très certainement la rentabiliser très largement avec une certaine marge en **commercialisant** notre base de données.
- **Ajout de fonctionnalité :**
 - Prérequis pour une espèce en plus des températures : eau (salée/douce), forêt, glace, désert, jungle, humidité, luminosité... Tâches en partenariat avec des biologistes.
 - Dessiner sur la carte des zones correspondant aux territoires des espèces.
 - Ajouter une vue avec les localisations actuelles des espèces dans le monde.
 - Possibilité de localiser les pays où la situation climatique devient catastrophique : cela permettrait d'agir pour sauver des zones avant qu'elles ne deviennent hostiles pour toute forme de vie. Il faudrait définir des protocoles pour réhabiliter une zone en proie à une catastrophe climatique.
 - Possibilité de localiser des catastrophes naturelles : séisme, tsunami, sécheresse, inondation, cyclone/tempête... Cela pourra également nous permettre de connaître l'impact du réchauffement climatique sur la fréquence et l'intensité des catastrophes naturelles.
- **Améliorer la sécurité du site et son aspect** : définir des protocoles.

Nuage de mot :



Structure base de données :

Table Animal

id	name	temp_min	temp_max	image
1	Ours polaire	-60	5	Ours.jpg
5	Phoque	-15	5	Phoque.jpg
6	Esteban	15	28	
7	Axek	30	60	Axek.jpg
8	Barnou	5	20	Barnou.jpg

Table membres

id	identifiant	password	email	rôle
1	Nicolas	Barnou	dddddd	admin
2	Axel	Axek	ddddddd	admin
3	Mouad	abcd	dddddd	admin
4	Esteban	1234	dddddddddd	admin

Table role_keys

rôle	role_key
admin	36d3ee5a
biologist	fd297722
ecologist	0614c1b5

Table data

city	max_temp	min_temp	temp	ret_time	longitude	latitude	humidity	sunrise_time	sunset_time	atmos_pressure	weather	last_update
Abuja	34.7	34.7	34.7	2020-04-16 19:00:04	7.49	9.06	12	2020-04-16 05:19:29	2020-04-16 17:39:35	1004	few clouds	2020-04-16 19:00:03
Accra	30	30	30	2020-04-16 19:00:05	-0.2	5.56	84	2020-04-16 05:52:51	2020-04-16 18:07:44	1007	broken clouds	2020-04-16 19:00:02
Adamstown	22.78	22.78	22.78	2020-04-16 19:00:06	-130.1	-25.07	77	2020-04-16 14:55:48	2020-04-17 02:23:49	1015	light rain	2020-04-16 19:00:06
Algiers	18	18	18	2020-04-16 19:00:09	3.04	36.75	88	2020-04-16 05:11:56	2020-04-16 18:22:44	1015	scattered clouds	2020-04-16 19:00:08
Alofi	24	24	24	2020-04-16 19:00:11	-169.92	-19.06	94	2020-04-16 17:30:06	2020-04-17 05:08:01	1014	overcast clouds	2020-04-16 19:00:10
Amman	16	16	16	2020-04-16 19:00:12	35.95	31.96	55	2020-04-16 03:05:48	2020-04-16 16:05:38	1011	few clouds	2020-04-16 18:57:59
Amsterdam	17.78	10.56	14.04	2020-04-16 19:00:13	4.89	52.37	54	2020-04-16 04:39:48	2020-04-16 18:40:05	1016	broken clouds	2020-04-16 18:59:11
Ankara	12	8	10.02	2020-04-16 19:00:14	32.85	39.92	54	2020-04-16 03:08:50	2020-04-16 16:27:24	1018	scattered clouds	2020-04-16 18:59:01
Antananarivo	17	17	17	2020-04-16 19:00:15	47.54	-18.91	82	2020-04-16 02:59:59	2020-04-16 14:38:45	1023	few clouds	2020-04-16 19:00:15
Apia	28.22	28.22	28.22	2020-04-16 19:00:16	-171.77	-13.83	61	2020-04-17 17:33:32	2020-04-18 05:10:57	1012	moderate rain	2020-04-16 19:00:16
Ashgabat	17	17	17	2020-04-16 19:00:17	58.38	37.95	63	2020-04-17 01:27:56	2020-04-17 14:43:38	1012	clear sky	2020-04-16 18:58:25
Asmara	14.12	14.12	14.12	2020-04-16 19:00:19	38.93	15.33	28	2020-04-16 03:08:56	2020-04-16 15:38:40	1010	clear sky	2020-04-16 19:00:18
Asunción	26	26	26	2020-04-16 19:00:20	-57.64	-25.3	31	2020-04-16 10:06:04	2020-04-16 21:33:58	1023	clear sky	2020-04-16 19:00:02
Athens	15.56	10	13.04	2020-04-16 19:00:21	23.72	37.98	35	2020-04-16 03:47:47	2020-04-16 17:01:28	1019	clear sky	2020-04-16 18:57:31
Avarua	24	24	24	2020-04-16 19:00:22	-159.78	-21.21	83	2020-04-16 16:51:17	2020-04-17 04:25:44	1014	overcast clouds	2020-04-16 19:00:22
Baghdad	25	25	25	2020-04-16 19:00:23	44.4	33.34	27	2020-04-16 02:30:32	2020-04-16 15:33:18	1010	scattered clouds	2020-04-16 18:59:33
Baku	9	9	9	2020-04-16 19:00:24	49.89	40.38	76	2020-04-16 02:00:08	2020-04-16 15:19:48	1018	overcast clouds	2020-04-16 18:58:44
Bamako	37	37	37	2020-04-16 19:00:26	-8	12.65	29	2020-04-16 06:18:41	2020-04-16 18:44:18	1011	few clouds	2020-04-16 19:00:25
Bangkok	31.11	27.78	29.5	2020-04-16 19:00:27	100.52	13.75	79	2020-04-16 23:03:20	2020-04-17 11:31:10	1009	scattered clouds	2020-04-16 18:55:38
Bangui	27	27	27	2020-04-16 19:00:28	18.55	4.36	78	2020-04-16 04:38:45	2020-04-16 16:51:51	1005	thunderstorm	2020-04-16 18:58:20
Banjul	28	28	28	2020-04-16 19:00:29	-16.58	13.45	54	2020-04-16 06:52:22	2020-04-16 19:19:15	1010	clear sky	2020-04-16 18:59:50
Basseterre	30	28	28.73	2020-04-16 19:00:30	-62.73	17.29	69	2020-04-16 09:53:48	2020-04-16 22:26:57	1017	shower rain	2020-04-16 19:00:29
Beijing	13.89	10	12.44	2020-04-16 19:00:31	116.4	39.91	81	2020-04-16 21:33:31	2020-04-17 10:53:58	1010	scattered clouds	2020-04-16 18:56:39
Beirut	22	8.33	16.5	2020-04-16 19:00:32	35.49	33.89	46	2020-04-16 03:05:32	2020-04-16 16:09:34	1009	scattered clouds	2020-04-16 18:56:50
Belgrade	15	12	13.62	2020-04-16 19:00:33	20.47	44.8	41	2020-04-16 03:51:20	2020-04-16 17:23:55	1017	scattered clouds	2020-04-16 18:55:57

Tutogit :

Allez sur <https://git-scm.com/>

Télécharger la dernière version et suivez les étapes d'installation :



Une fois installé, ouvrir votre terminal (cmd pour windows), puis il faut vérifier que git a correctement été installé.

Taper la commande : `git --version`

```
Last login: Sat Sep 30 11:10:00 on ttys000
$ git --version
git version 2.14.1
$
```

Si vous voyez git avec sa version, c'est que tout s'est bien passé.

Il faut ensuite définir l'identité de l'utilisateur :

`git config --global user.name "Nicolas Evain"`

`git config --global user.email "Barnou64@gmail.com"`

Pour créer un projet :

`mkdir nom`

`cd nom`

Il faut créer un répertoire puis aller dedans pour initialiser git.

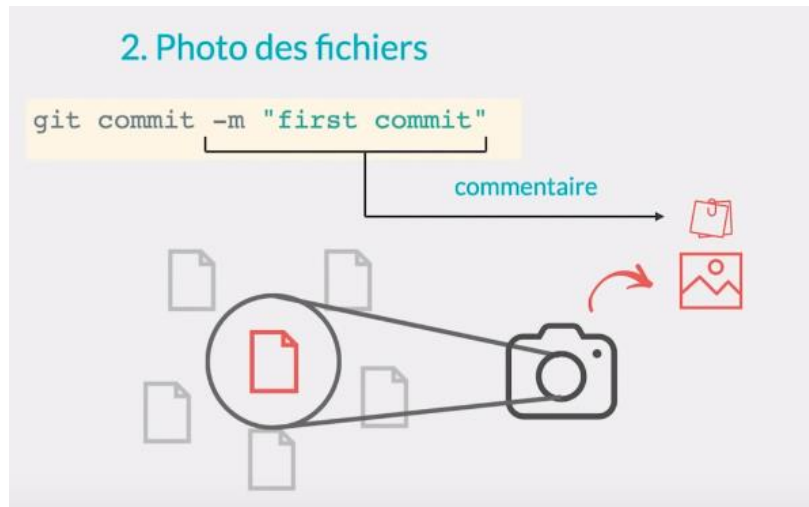
Taper la commande : `git init`

// pour demander un statut global des fichiers de notre répertoire taper : `git status`

Le commit : (sert à enregistrer l'état actuel du projet), elle se fait en deux étapes

1. La sélection des fichiers : `git add filename` → `git add index.html` (par exemple)

2. Le commit :



Pour avoir la time line des commit : `git log` (qui a fait quoi)

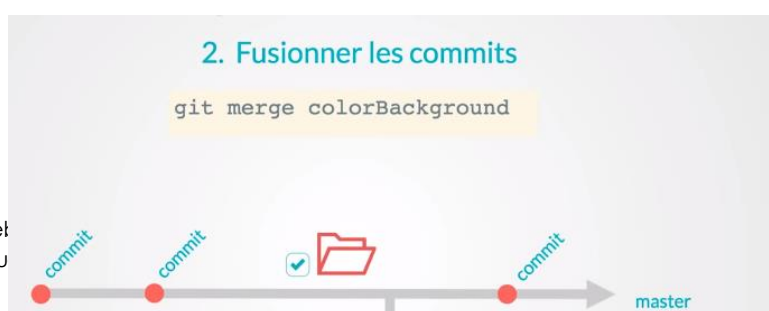
Les branches (copie de ton projet initial dans lequel tu vas pouvoir faire de la merde : version, si les modifications ne te plaisent pas tu vas pouvoir les supprimer sinon tu vas pouvoir les rapatrier) :



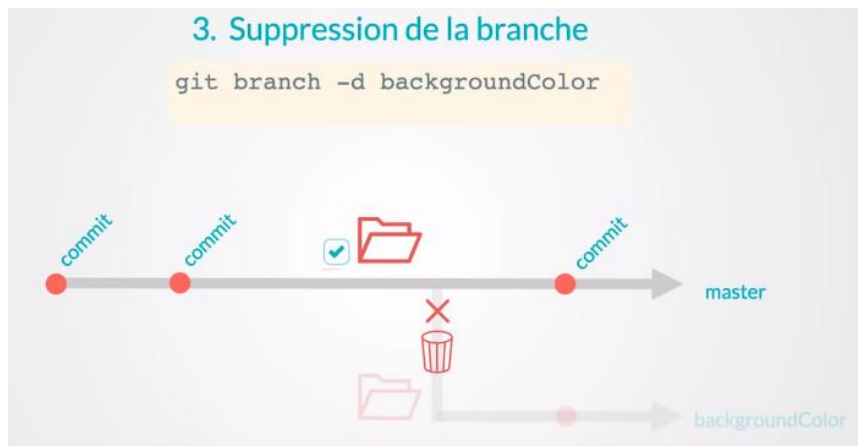
Pour lister les branches : `git branch`

Pour changer de branch : `git checkout nonBranch`

Pour fusionner les commit :

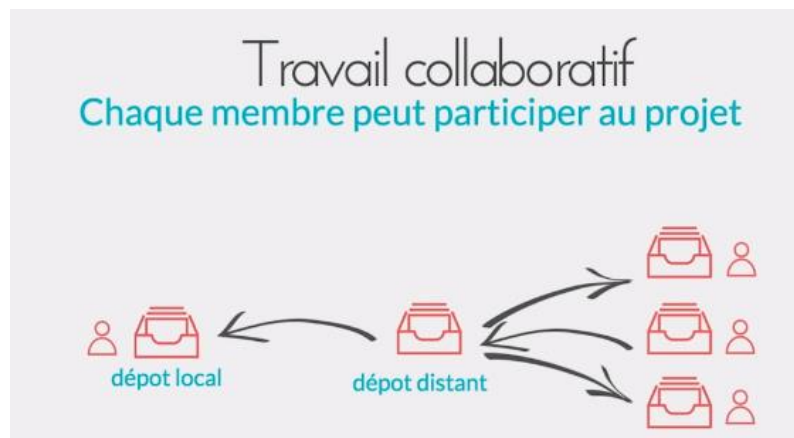


Puis on supprime la branche :



Pour collaborer avec d'autres personnes :

Notion de dépôt distant & Github :



Pour créer un dépôt distant il faut aller sur github et se créer un compte.

Puis aller sur : <https://github.com/new>

Et créer le répertoire : nom, description, public.

Il faut maintenant créer un pont entre notre dépôt local et distant :

...or create a new repository on the command line

```
echo "# globalWarming" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/Pitolord/globalWarming.git
git push -u origin master
```

...or push an existing repository from the command line

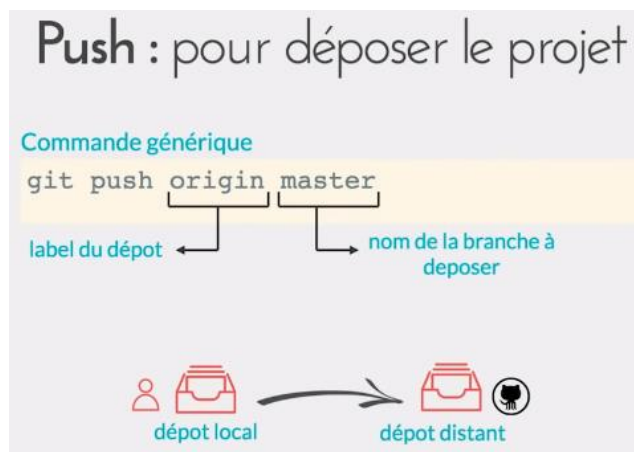
```
git remote add origin https://github.com/Pitolord/globalWarming.git
git push -u origin master
```

Dans notre cas utiliser : `git remote add origin`

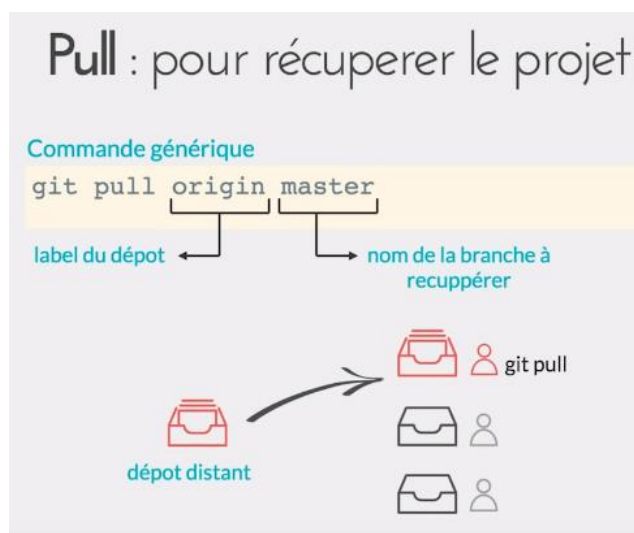
<https://github.com/Pitolord/globalWarming.git>

Ça va créer le pont qui va s'appeler origin : pour voir s'il existe un pont : `git remote`

Il reste à déposer le projet avec :



Pour récupérer le projet :



Processus de data mining :

Nous avons abordé notre sujet en effectuant un processus de data mining.

