CATRY Alexandre

Développement d'une application web de traitement de données scientifiques

Rapport de stage de 2ème année
Parcours : IRIS

Enseignant référent: M. Derrien Olivier

Année Universitaire 2016-2017

Engagement de non plagiat.

Je soussigné, ……

N° carte d'étudiant : ……

Déclare avoir pris connaissance de la charte des examens et notamment du paragraphe spécifique au plagiat.

Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document publiés sous quelques formes que ce soit (ouvrages, publications, rapports d'étudiant, internet etc…) constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, je m'engage à citer toutes les sources que j'ai utilisées pour produire et écrire ce document.

Fait le ……

Signature(s)

Ce document doit être inséré en première page de tous les rapports, dossiers et/ou mémoires.

Table des matières

Table des figures

I) Présentation et contexte du stage

a) Présentation du laboratoire

Le laboratoire d'optique atmosphérique (LOA) est un laboratoire CNRS situé sur le campus de l'université Lille 1, dont le domaine d'étude est l'optique atmosphérique. L'optique atmosphérique cherche à modéliser la propagation à travers l'atmosphère de la lumière visible reçue du soleil et de la lumière infrarouge émise par l'ensemble des surfaces et de l'atmosphère terrestres. Les travaux menés au LOA dans ce domaine s'insèrent dans l'étude globale du climat.

Un premier objectif est de quantifier le rôle de ce rayonnement visible et infrarouge dans les échanges énergétiques de la planète, en particulier de préciser le rôle des nuages dans le bilan radiatif de la terre dont ils constituent un facteur essentiel.

Un second axe de recherche porte sur la caractérisation à l'échelle du globe de différents paramètres qui sont en relation directe avec l'évolution climatique (nuages, aérosols, surfaces), en utilisant principalement l'observation satellitaire.

Le laboratoire est composé de différents services, chacun ayant ses missions respectives.

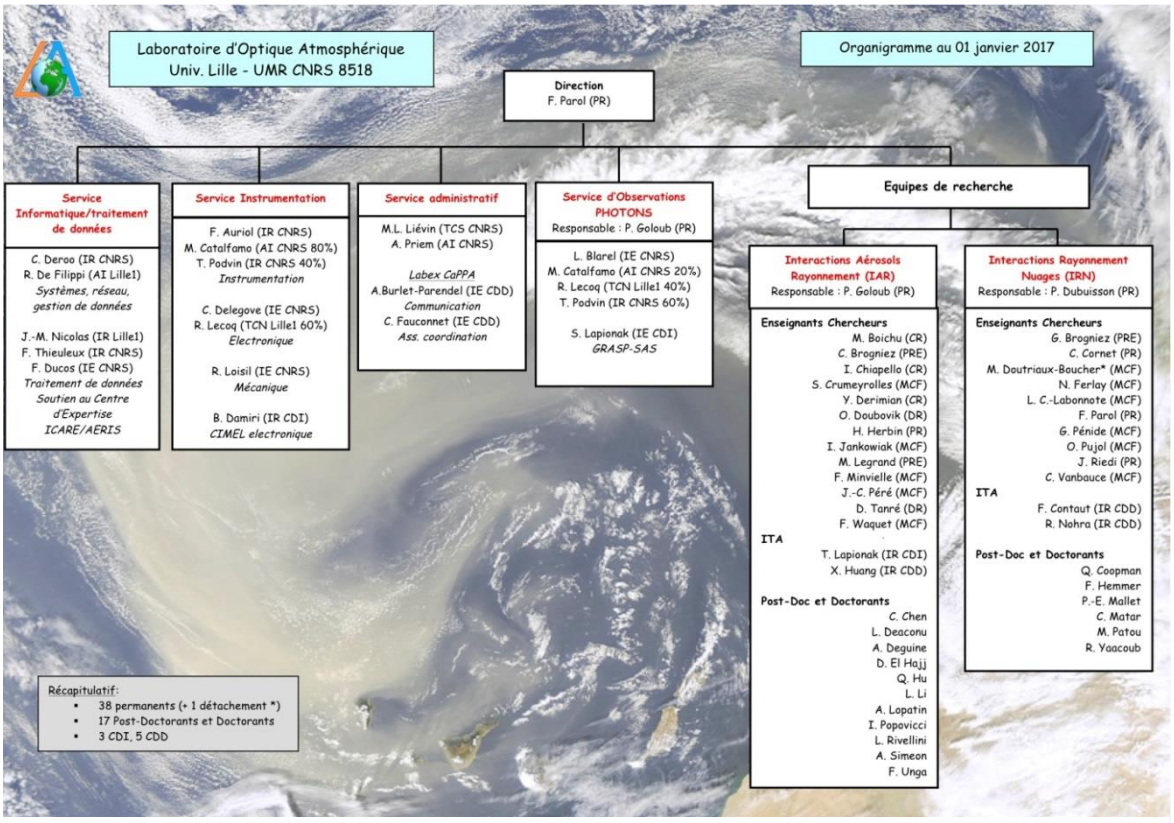

Figure 1 : Organigramme du LOA

Le service PHOTONS, dans lequel j'ai effectué mon stage fait partie des fondateurs, avec la NASA, du réseau mondiale AERONET de photomètres. Ce réseau est constitué de nombreux photomètres, installés sur tous les continents, utilisés pour étudier l'atmosphère en analysant la lumière du soleil. Les intensités captées par les photomètres sont ensuite utilisées pour calculer l'épaisseur optique de l'atmosphère, afin de déterminer le type de particules présentes et les quantifier.

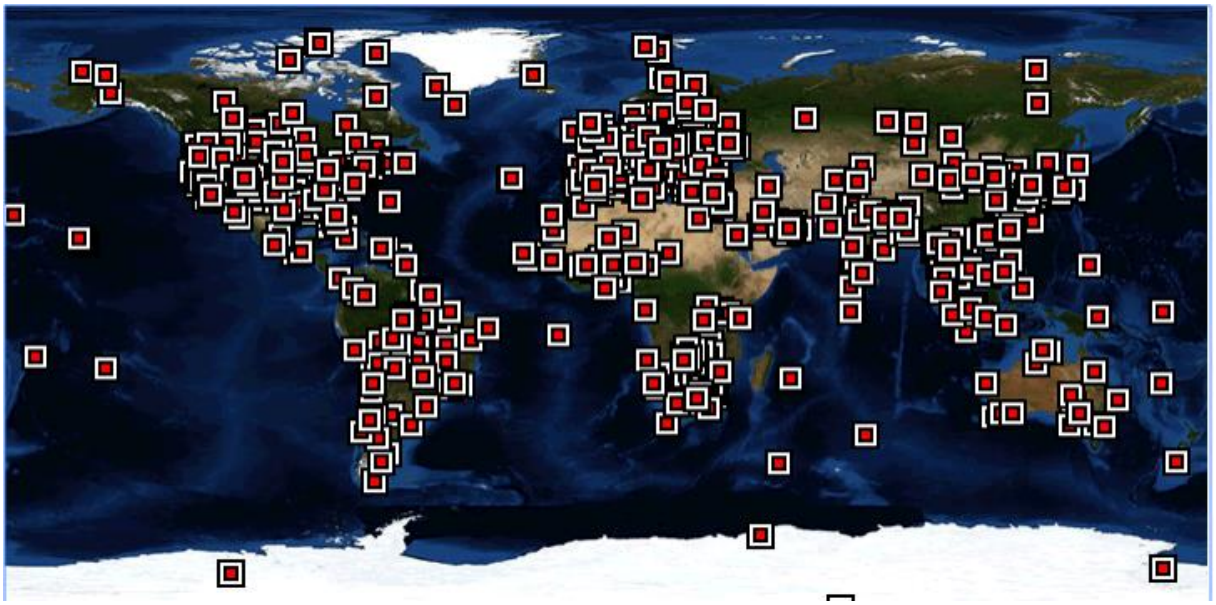

Figure 2 : Carte du réseau AERONET

b) Contexte du stage

Depuis 2008, le laboratoire d'optique atmosphérique travaille sur un photomètre mobile : le Photomètre Léger Aéroporté pour la Surveillance des Masses d'Air (PLASMA). Le PLASMA a été conçu dans le cadre de l'étude des aérosols, pour valider des mesures satellitaires. Il sert à réaliser des cartes en trois dimensions de l'épaisseur optique de régions de l'atmosphère accessibles par avion. Deux PLASMA existent à

Figure 3 : Un des PLASMA existants

l'heure actuelle et un troisième est en construction. Les PLASMA sont équipés de 9 filtres dans le visible et de 6 filtres dans l'infrarouge.

Avant mon stage, pour exploiter les données du PLASMA, la démarche était la suivante :

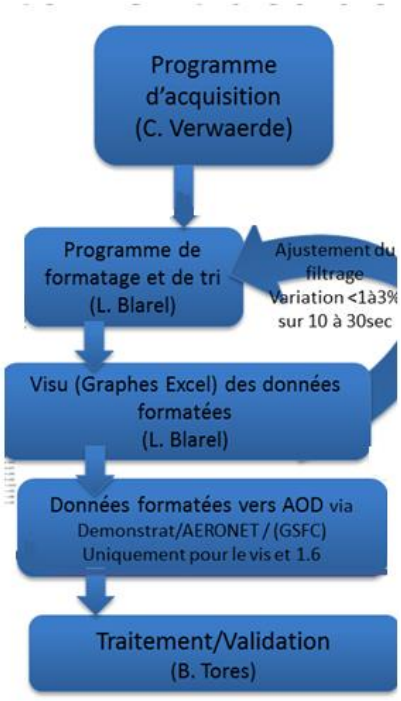

Figure 4 : Procédure de traitement des données du PLASMA

- Un programme d'acquisition écrit les données brutes au fur et à mesure que le PLASMA les obtient.
- Un programme de traitement convertit les données brutes en ligne de mesures, puis il les filtre pour éliminer les mesures aberrantes.
- Une visualisation a lieu sur Excel, pour détecter un besoin de modification des paramètres de filtrage.
- Un programme, sur un serveur de la NASA, calcule les épaisseurs optiques à partir des lignes de mesures.
- Un programme valide les épaisseurs optiques.

Les étapes de filtrage et de visualisation entrainaient une perte de temps considérable. En effet, l'opérateur devait basculer entre le programme de formatage et Microsoft Excel de manière répétée pour obtenir le résultat idéal. De plus, les fichiers de données étaient stockés directement au format txt sur un ordinateur du laboratoire. La gestion des fichiers était donc également très chronophage.

Le but de mon stage était de réaliser une application web rassemblant le traitement des données brutes, le filtrage, la visualisation, et le calcul des épaisseurs optiques. En plus des fonctionnalités déjà existantes dans les différents programmes, il m'était également demandé de créer un système de base de données pour stocker les campagnes de mesures réalisées par chaque PLASMA. Enfin, je devais proposer un

système d'automatisation du traitement des données qui permettrait de voir l'évolution des mesures pendant leurs réalisations.

Le stage s'est déroulé sous la supervision de M. Luc Blarel, ingénieur CNRS et futur utilisateur de l'application, et M. Fabrice Ducos, ingénieur CNRS, qui me conseillait sur l'aspect informatique.

Le choix du langage de programmation avait déjà été défini par M. Ducos : Python avec le framework Django sous Ubuntu. J'ai donc commencé par prendre en main ce framework. Le type de base de données était libre et j'ai commencé le développement avec la base de données par défaut du framework : SQLite. Il s'est par la suite avéré que PostGreSQL offrait de nombreuses possibilités supplémentaires, c'est donc sur ce type de base de données que l'application a été terminée.

Le projet est nommé « Système de Traitement des AOD de PLASMA » ou STrAP.

II) Description du travail

a) Structure d'une application Django

La première étape de mon stage a été d'apprendre à utiliser le framework Django du langage Python. Ce framework permet de créer des applications très facilement, tout en offrant de nombreuses possibilités. J'ai appris à l'utiliser grâce à deux livres, disponible sur place, ainsi qu'au tutoriel en ligne du site OpenClassroom.

Une application développée sous Django est organisée en différentes apps. Ces apps sont des sous-programmes Django pouvant fonctionner séparément, et créées de manière arbitraire par le développeur pour diviser son travail. Une app portant le nom du projet est créée par défaut. Chacune de ces apps possède la même structure :

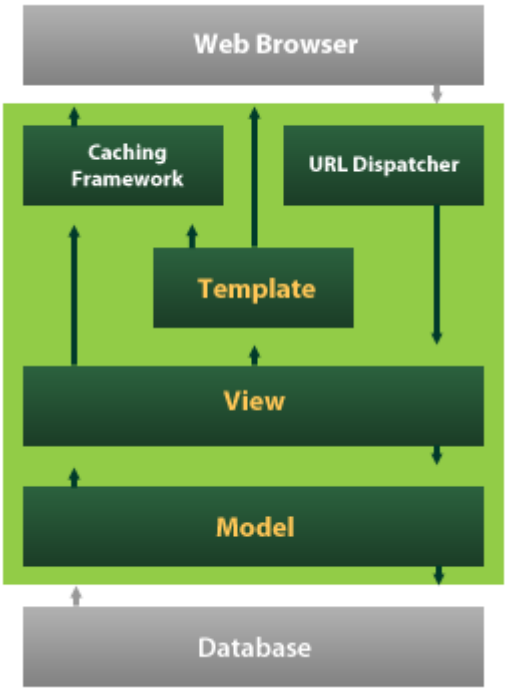

Figure 5 : Architecture du framework Django

Les Models sont les tables de la base de données et sont créés dans un fichier models.py.

```
class GPS(models.Model):
        """
Ce modèle correspond aux trames GPS (commençant par G) du fichier texte d'entrée.
        """
        type_trame=models.CharField(max_length=1)
        secondes=models.IntegerField()
        latitude=models.FloatField()
        longitude=models.FloatField()
        altitude=models.FloatField()
        cap=models.CharField(max_length=10)
        tangage=models.CharField(max_length=10)
        roulis=models.CharField(max_length=10)
        vitesse=models.CharField(max_length=10)
        trace_au_sol=models.CharField(max_length=10)
        top_GPS=models.CharField(max_length=10)
        tempv=models.IntegerField()
        tempir=models.IntegerField()
        numero_trame=models.IntegerField(db_index=True) #numero de la trame dans le fichier txt dont elle provient
        jour=models.ForeignKey('campagne.Jour',db_index=True)

        def __str__(self):
                return(self.type_trame+' '+str(self.secondes))
```

Figure 6 : Exemple de model

Les Views sont les fonctions appelées lorsque l'utilisateur entre une adresse URL de l'application. Elles sont créées dans un fichier views.py.

```
@login_required
def accueil_jour(request,campagne_nom,jour_url,nom_plasma):
        """
Cette view correspond à la page d'accueil d'un jour.
        """
        campagne=get_object_or_404(Campagne,nom=campagne_nom)
        jours=Jour.objects.filter(campagne=campagne).order_by('compte_jour')
        jour=jours.get(date_url=jour_url,plasma__nom=nom_plasma)
        return render(request,'accueil_jour.html',{'campagne':campagne,'jours':jours,'jour':jour,'jours':jours})
```

Figure 7 : Exemple de view

L'URL Dispatcher est une table associant les adresses URL aux views. Chaque view peut répondre à une ou plusieurs adresses URL. Le dispatcher est créé dans un fichier urls.py.

```
urlpatterns = [
        url(r'^(?P<campagne_nom>[\w\-]+)/$', views.home_page, name='home'),
    url(r'^(?P<campagne_nom>[\w\-]+)/(?P<jour_url>[\w\-]+)/(?P<nom_plasma>[\w\-]+)/$', views.accueil_jour,name='accueil_jour'),
        url(r'^(?P<campagne_nom>[\w\-]+)/(?P<jour_url>[\w\-]+)/(?P<nom_plasma>[\w\-]+)/filtrage/$',views.filtrage,name='filtrage'),
        url(r'^(?P<campagne_nom>[\w\-]+)/(?P<jour_url>[\w\-]+)/(?P<nom_plasma>[\w\-]+)/filtrage/graph/$',views.graph,name='graph'),
        url(r'^(?P<campagne_nom>[\w\-]+)/(?P<jour_url>[\w\-]+)/(?P<nom_plasma>[\w\-]+)/filtrage/graph/delete/$',views.delete_filtre,name='delete_filtre'),
        url(r'^(?P<campagne_nom>[\w\-]+)/(?P<jour_url>[\w\-]+)/(?P<nom_plasma>[\w\-]+)/export/$',views.export,name='export'),
        url(r'^(?P<campagne_nom>[\w\-]+)/(?P<jour_url>[\w\-]+)/(?P<nom_plasma>[\w\-]+)/graph_glob_seul/$',views.graph_glob_seul,name='graph_glob_seul'),
]
```

Figure 8 : Exemple de dispatcher

Les Templates sont les gabarits de page web, dans lesquelles seront inscrites les données envoyées depuis la base de données entre {{ }}. Elles sont stockées dans un dossier Templates et sont codées en HTML, CSS et JavaScript selon les besoins du développeur.

```
{% extends 'strapapp/base.html' %}

{% block title %}
    STrAP application
{% endblock %}

{% block nav %}
    <ul class="nav nav-tabs">
        <li role="presentation" class="active"><a href="/strapapp/{{campagne.nom}}/{{jour.date_url}}/{{jour.plasma.nom}}/">Accueil</a></li>
        <li role="presentation"><a href="/strapapp/{{campagne.nom}}/{{jour.date_url}}/{{jour.plasma.nom}}/filtrage">Filtrage des données</a></li>
        <li role="presentation"><a href="/strapapp/{{campagne.nom}}/{{jour.date_url}}/{{jour.plasma.nom}}/export">Exporter les résultat</a></li>
        <li role="presentation"><a href="/strapapp/{{campagne.nom}}/">Retour aux résumés des jours</a></li>
        <li role="presentation"><a href="/index/">Changer de campagne</a></li>
    </ul>
{% endblock %}


{% block content %}
<h3>Récapitulatif de la journée</h3>
<p>Date: {{jour}}</p>
<p>Nom du PLASMA: {{jour.plasma.nom}}</p>
<p>Lignes de mesures générées: {{jour.nb_glob}}</p>
<p>Lignes de mesures filtrées: {{jour.nb_glob_filtre}}</p>

<iframe id="frameTest" src="/strapapp/{{campagne.nom}}/{{jour.date_url}}/{{jour.plasma.nom}}/graph_glob_seul/" alt='erreur' frameborder="0" scrolling="no"

{% endblock %}
```

Figure 9 : Exemple de template

Comme dit précédemment, chaque app possède chacun de ces fichiers. Dans le cas des URL Dispatchers, le fichier urls.py de l'application créée par défaut permet de rediriger les appels de view vers le dispatcher de l'app à utiliser.

Le développement a commencé avec une base de données SQLite. Par la suite, j'ai découvert que Django propose des fonctionnalités supplémentaires s'il est utilisé avec une base de données PostGreSQL. Parmi ces fonctionnalités se trouvent en particulier les ArrayFields. Ces champs de models permettent d'ajouter un tableau dans le model. Leur utilisation était particulièrement pertinente pour ce projet car les données sont répétées pour chaque filtre dans les lignes de mesures. J'ai donc décidé de continuer le développement sous PostGreSQL.

b) Elaboration de la base de données

Avant de me lancer dans le développement, il m'a fallu déterminer une structure pour la base de données, et donc prendre connaissance des détails du sujet.

Les données brutes créées par le programme d'acquisition sont sous la forme d'un fichier txt composé de trames de 3 types : Instrument, GPS et Mesure. Chaque fichier d'acquisition commence par une trame Instrument, commençant par la lettre S, contenant différentes données sur le PLASMA utilisé ainsi que la date. Viennent ensuite des trames GPS et Mesures. Une fois par seconde, une trame GPS, commençant par la lettre G, est écrite avec l'heure et les données GPS du PLASMA. Les trames Mesure correspondent à une mesure sur une unique longueur d'onde. Elles contiennent la valeur de la mesure, le numéro du filtre, le gain et d'autres informations relatives à la mesure. Elles sont inscrites dès qu'une mesure est terminée. Elles commencent par un V si la mesure vient d'un filtre dans le visible et un I si c'est un filtre dans l'infrarouge. Voici un exemple de trames du fichier d'entrée :

```
S0 25 11 4 13 3219 500 0 200 -20 3578 432 3715
   406 1488 992 16

V28443 2 4 1228 1830 -9 1 7645 0 209

I1803404 4 1 1230 1830 -6 -9 7647 0 23

G25576 440833516 50591583 1128 0 0 0 1 14181 67
   150 137 2 3 8 -166100 -66227
```

Les lignes de mesures à générer reprennent une grande partie des données des trames. J'ai décidé de créer une table pour chaque type de trame ainsi que pour les lignes de mesures.

Les mesures sont réalisées dans le cadre de campagnes de mesures qui s'étendent sur plusieurs jours. J'ai créé les tables Campagne et Jour. Chaque trame/ligne de

mesures est identifié par sa clé étrangère vers un Jour et chaque jour par sa clé étrangère vers une Campagne.

Enfin, j'ai créé une table Paramètre PLASMA pour l'ensemble des paramètres nécessaire à l'exploitation des données, tels que les longueurs d'ondes et les coefficients de conversion. Chaque jour possède une clé étrangère vers un jeu de paramètres. La structure finale de la base de données est la suivante :

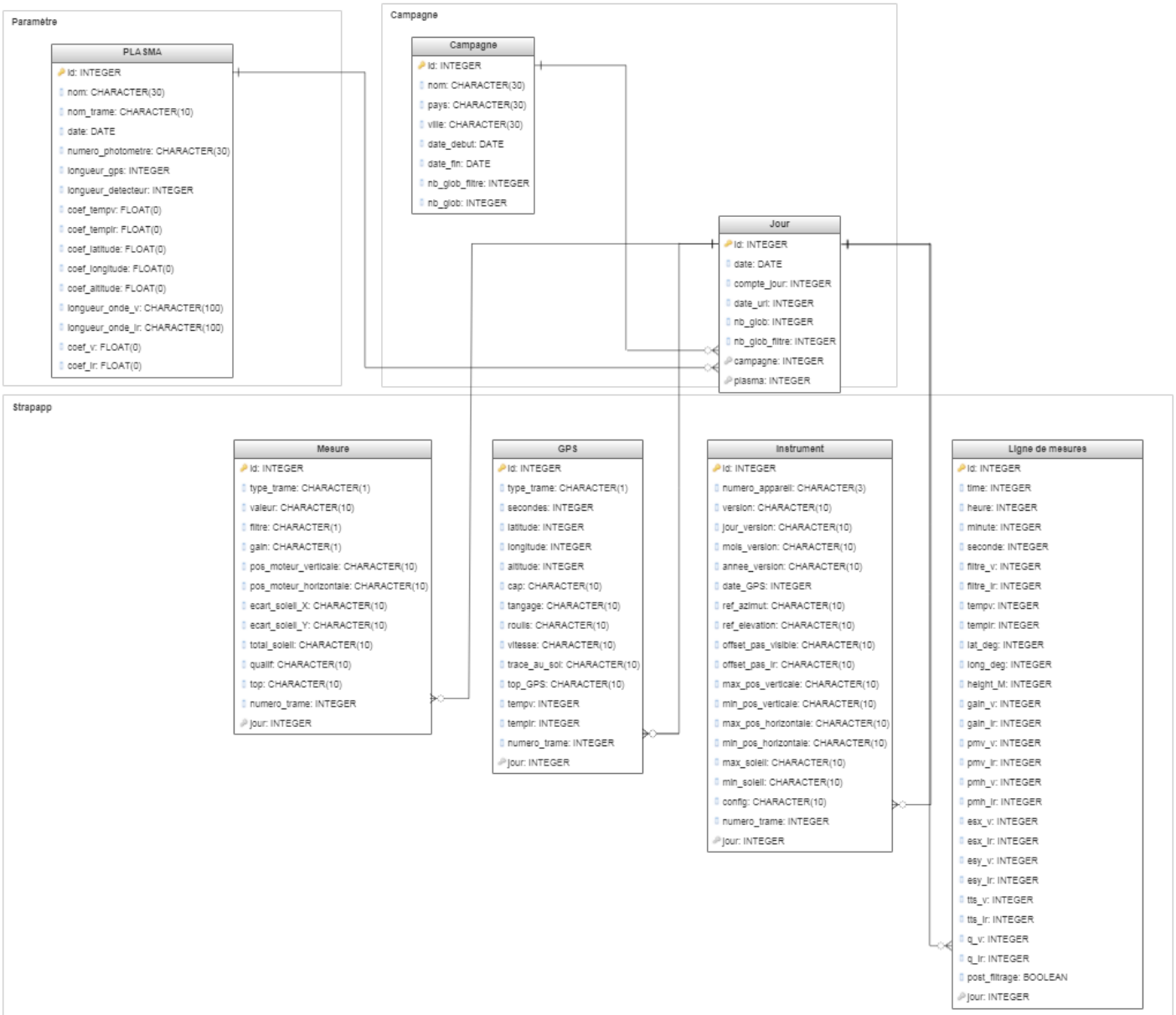

Figure 10 : Structure de la base de données

J'en ai déduis une structure pour mon application, composée de 4 apps :

- Strapapp : Apps « principale » qui se charge du traitement des données. C'est dans cette application que sont définis les 4 modèles stockant les données mesurées.
- Parameter : Apps gérant les paramètres PLASMA.
- Campagne : Apps gérant les Campagnes et les Jours.
- Index : Apps gérant l'index de STrAP.

Cette structure est la version finale de la base de données, mais elle a évolué avec l'avancement du projet.

c) Lecture des données mesurées

La lecture des données s'effectue sur le page d'accueil d'une Campagne dans l'application.

Figure 11 : Page d'accueil d'une Campagne

Cette lecture se déroule en deux temps : une première phase qui consiste à lire le fichier que l'utilisateur fournit en entrée, et une seconde où l'on génère les lignes de mesures.

La première phase est une simple lecture ligne par ligne du fichier d'entrée. La première lettre d'une trame indiquant son type, il est très simple de l'enregistrer dans le bon model. La première trame (Instrument) précisant la date et le numéro du PLASMA utilisé, elle est utilisée pour créer le Jour auquel appartiendront les trames suivantes. Les trames sont corrigées dans le cas où certains caractères indésirables se sont insérés. Une entrée est écrite dans un fichier log.txt pour chaque erreur corrigée, afin de pouvoir vérifier les corrections automatiques. Une vérification a également lieu sur la longueur des trames Mesure et GPS pour s'assurer qu'aucune information n'est manquante. Enfin, l'application supprime les trames GPS où une erreur de temps se produit en vérifiant les trames dans l'ordre d'écriture. Sans erreur, l'écart entre deux trames GPS successives est d'une seconde, autrement, la trame est supprimée.

La phase de création des lignes de mesures consiste à sélectionner une trame GPS et une trame de Mesure pour chaque filtre visible et infrarouge. Il est important que ces trames soient proches en temps, autrement les lignes de mesures n'ont aucun sens. La proximité temporelle est assurée grâce à l'attribut « numero_trame » de chaque trame. Cet attribut correspond au numéro de ligne de la trame dans le fichier originel.

d) Filtrage des données

Le filtrage s'effectue sur la page « Filtrage des données » de l'application.

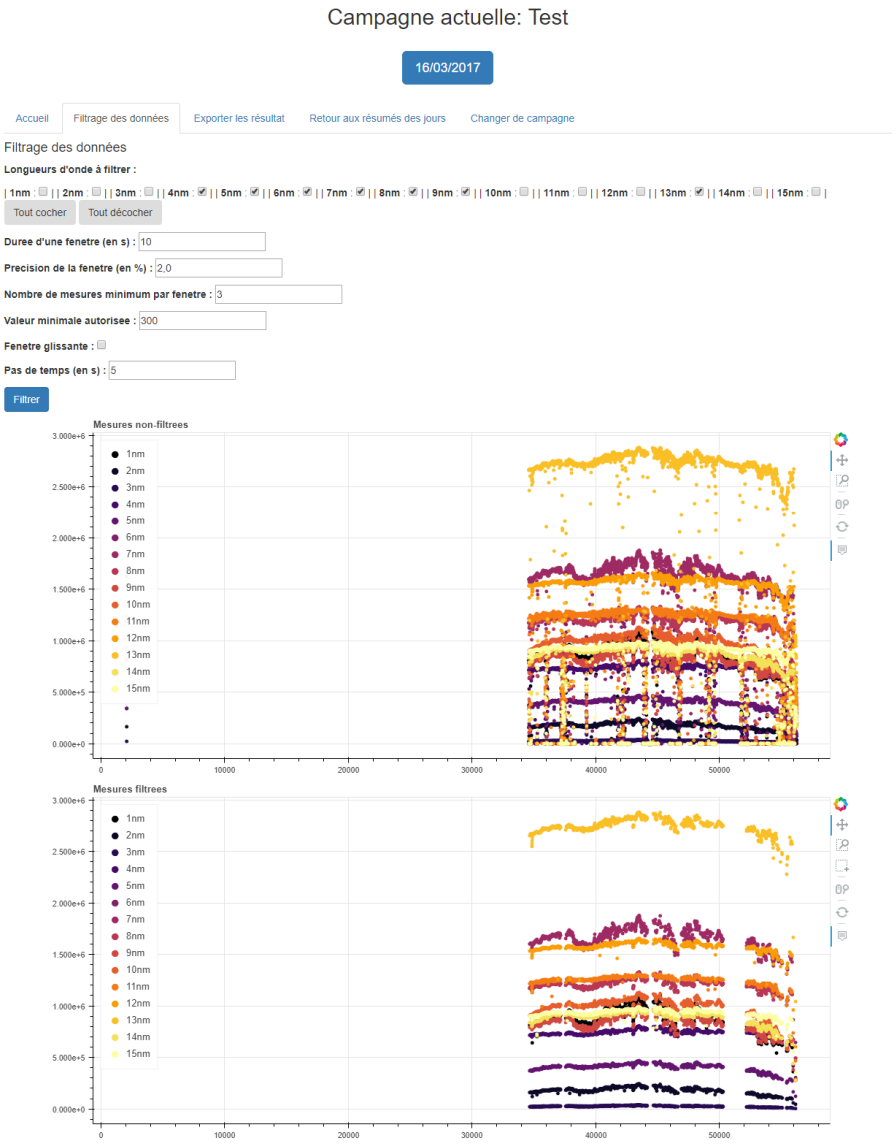

Figure 12 : Page de filtrage

Le filtrage des données a pour but de retirer les données perturbant le calcul des épaisseurs optiques, telles que les mesures à travers un nuage ou résultantes d'un problème technique. Le filtrage est réalisé en divisant la durée de mesure en fenêtre de durée fixe (par défaut 10 secondes). On commence par récupérer toutes les mesures réalisées dans cet intervalle de temps. Si toutes les mesures d'une fenêtre vérifient pour chaque longueur d'onde l'équation suivante :

$$\frac{\max(mesures) - \min(mesures)}{(\frac{\max(mesures) + \min(mesures)}{2})} < précision$$

avec « précision » un paramètre choisi par l'utilisateur (par défaut 2%), la fenêtre est validée. Une ligne de mesures filtrée est alors créée en faisant la moyenne des différentes mesures de la fenêtre. Si la fenêtre n'est pas valide, on passe à la fenêtre suivante.

Cette méthode fonctionne relativement bien, cependant elle a l'inconvénient de supprimer beaucoup de données au moindre nuage. M. Blarel m'a donc demandé de développer un mode de filtrage « fenêtre glissante », qui n'existait pas sur sa version du programme de traitement. Ce mode de filtrage reprend le principe exposé précédemment, à la différence que lorsqu'une fenêtre n'est pas valide, la fenêtre se décale d'une seconde au lieu de passer à la fenêtre suivante, ce qui permet de conserver un maximum de données.

Une fois le filtrage terminé, les résultats s'affichent dans les deux graphiques de la page. Le premier correspond aux données avant filtrage, tandis que le second représente les données après le filtrage. Les axes de ses deux graphiques sont liés, de telle sorte qu'un zoom sur l'un entraine le même zoom sur l'autre.

e) Représentation graphique

Pour éviter les allers-retours entre mon application et Microsoft Excel, comme c'était le cas sur l'ancien programme, M. Blarel m'avait demandé de rendre possible le tracé des données directement dans l'application. L'affichage des données est particulièrement utile pour le filtrage, car il permet de voir directement si un changement de paramètres de filtrage est nécessaire ou permettrait de meilleurs résultats.

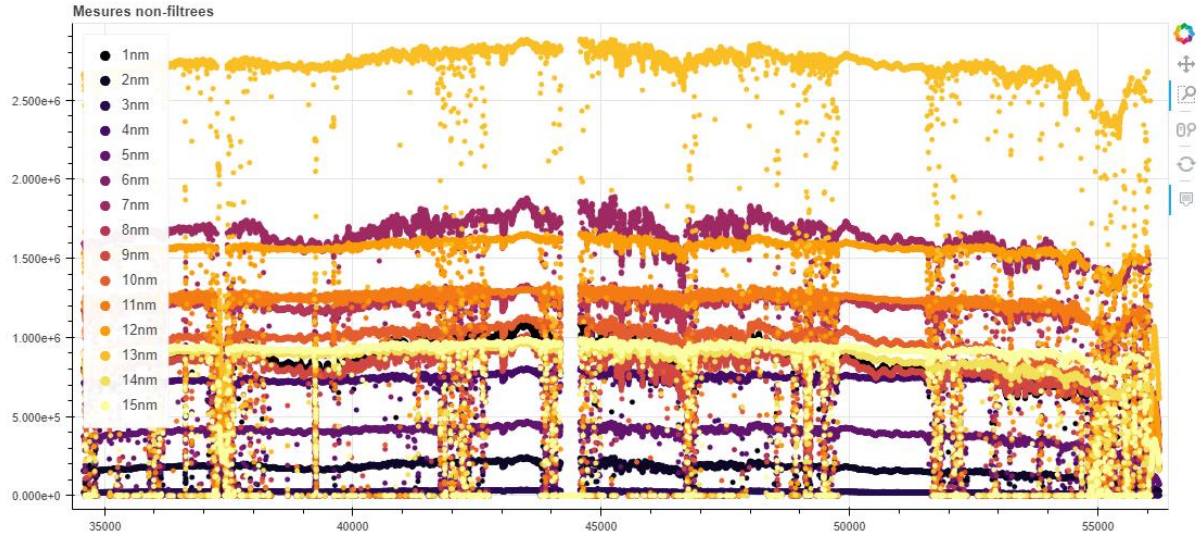

Figure 13 : Exemple de graphique sous Bokeh

La représentation graphique nécessitait l'utilisation d'un module Python. J'ai d'abord utilisé Matplotlib, qui permet un tracé très simple. Je souhaitais en outre un graphique interactif, qui permettrait à l'utilisateur de zoomer et se déplacer dans le graphique, ce qui est impossible avec Matplotlib. J'ai donc tenté d'utiliser le module mpld3, qui correspondait à mes attentes. Je me suis cependant vite rendu compte que mpld3 n'était pas du tout adapté au nombre important de points à tracer.

Finalement, le module Bokeh offrait des performances bien supérieures, en proposant d'intéressantes fonctionnalités. M. Blarel m'avait demandé la possibilité de supprimer des lignes de mesures filtrées individuellement, j'ai donc développé un outil pour Bokeh, qui permet de sélectionner un ou plusieurs points directement sur le graphique et de les supprimer.

f) Exportation des résultats

La récupération des résultats se fait sur la page « Exportation des résultats ».

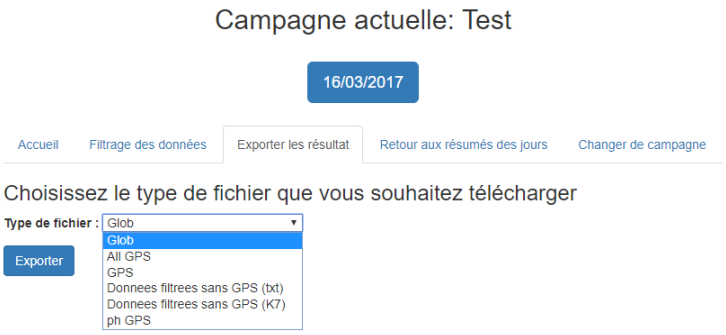

Figure 14 : Page d'exportation

Les résultats sont récupérés dans différents fichiers au format .txt. Ces fichiers reprennent certaines données des lignes de mesures et étaient générés à l'origine par l'ancien programme de traitement. Les fichiers sont sélectionnables par un menu déroulant et sont téléchargeables directement sur l'application.

g) Gestion des paramètres

La gestion des paramètres se fait sur la page « Gestion des paramètres ».

Figure 15 : Page de gestion des paramètres

Ces paramètres sont nécessaires à l'exploitation des données mesurées et doivent être définis avant tout traitement. En plus de leur création, M. Blarel m'a également demandé de réaliser un système de sauvegarde externe, afin d'éviter de devoir les recréer régulièrement. Pour répondre à ces besoins, j'ai réalisé des fonctions d'exportation et d'importation de paramètres. Les paramètres sont exportés dans un fichier .txt au format JSON. L'importation est une simple lecture de JSON suivie d'une inscription dans la base de données.

h) Mode automatique

Le mode automatique permet de réaliser les étapes de lecture de données automatiquement, dès la réalisation des mesures. Par manque de temps, le mode automatique n'a pas été déployé sur le serveur avant la fin de mon stage, cependant il est opérationnel et sera déployé sous peu par M. Ducos.

Ce mode est réalisable grâce au module Celery de Python, qui permet d'instaurer des tâches périodiques dans un fichier tasks.py. La tâche périodique qui réalise le mode automatique scanne régulièrement un dossier de dépôt, qui possède un sous-dossier au nom de chaque Campagne créée. Lorsqu'il trouve un fichier .txt, il l'intègre à la campagne correspondante et effectue les étapes décrites plus haut. Enfin, il déplace le fichier .txt dans un dossier d'archive.

i) Documentation

Ce projet ayant était réalisé en très grande autonomie, j'ai dû à réaliser deux documentations : une notice d'utilisation à destination de M. Blarel, afin de lui expliquer le fonctionnement de l'application, et une documentation technique, à destination de M. Ducos, qui s'occupera de la maintenance et du développement de l'application après mon départ.

III) Optimisations réalisables

Par manque de temps, plusieurs fonctionnalités n'ont pas pu être réalisées.

Le calcul des épaisseurs optiques n'a pas pu être intégré à l'application.

La partie d'exportation des données présente deux problèmes :

- L'un des formats demandé par M. Blarel impliquait de créer un fichier au format .K7. Ce type de fichier est spécifique aux photomètres, il n'y a donc pas de documentation en ligne. Je n'ai donc pas pu la développer.
- Les entêtes de certains fichiers exportés ne s'accordent pas avec les longueurs d'onde définies dans les paramètres.

L'initialisation du mode automatique nécessite que la base de données ait déjà été initialisée, autrement l'application ne se lancera pas.

M. Blarel m'avait également demandé plusieurs fonctionnalités non prioritaires que je n'ai pas eu le temps de réaliser. Entre autre, pouvoir naviguer entre les campagnes de mesures grâce à un calendrier et une carte du monde aurait été apprécié.

La fonction de filtrage semblant encore un peu longue à exécuter, je pense qu'elle peut être optimisée pour permettre à meilleur confort d'utilisation.

Enfin, l'apparence de l'application est très basique. J'aurai aimé avoir plus de temps à lui accorder.

Conclusion

Ce stage m'a permis de réaliser le développement d'une application web dans son intégralité. L'application est fonctionnelle et sera utilisée par M. Blarel dans sa campagne de mesures en Namibie fin août. Elle lui permettra un gain de temps et d'efficacité. Les documentations sont claires et ont été approuvée MM. Blarel et Ducos.

A travers le développement, j'ai pu mettre en application certaines connaissances acquises durant ma seconde année à Seatech. L'utilisation et l'adaptation à un framework, vu lors de l'apprentissage du Visual C++, m'ont permis de très vite prendre en main le framework Django. La programmation orientée objet était également au cœur du projet.

Ce stage m'a aussi permis de développer mon aptitude à travailler en autonomie.

Lors de mon stage, j'ai également découvert l'aspect social, indispensable en entreprise. En effet, bien que j'aie développé l'application entièrement seul, en grande autonomie, j'ai eu des contacts réguliers avec M. Blarel pour lui présenter mon travail et pour qu'il me précise ce qu'il souhaitait changer, m'obligeant parfois à repenser complétement la manière de fonctionner d'une partie de mon application. J'ai également eu de nombreux échanges avec M. Ducos, qui m'a conseillé et aidé à de nombreuses reprises dans le développement.

Le développement de l'application aura été très passionnant, confirmant mon choix de m'orienter dans cette voie.