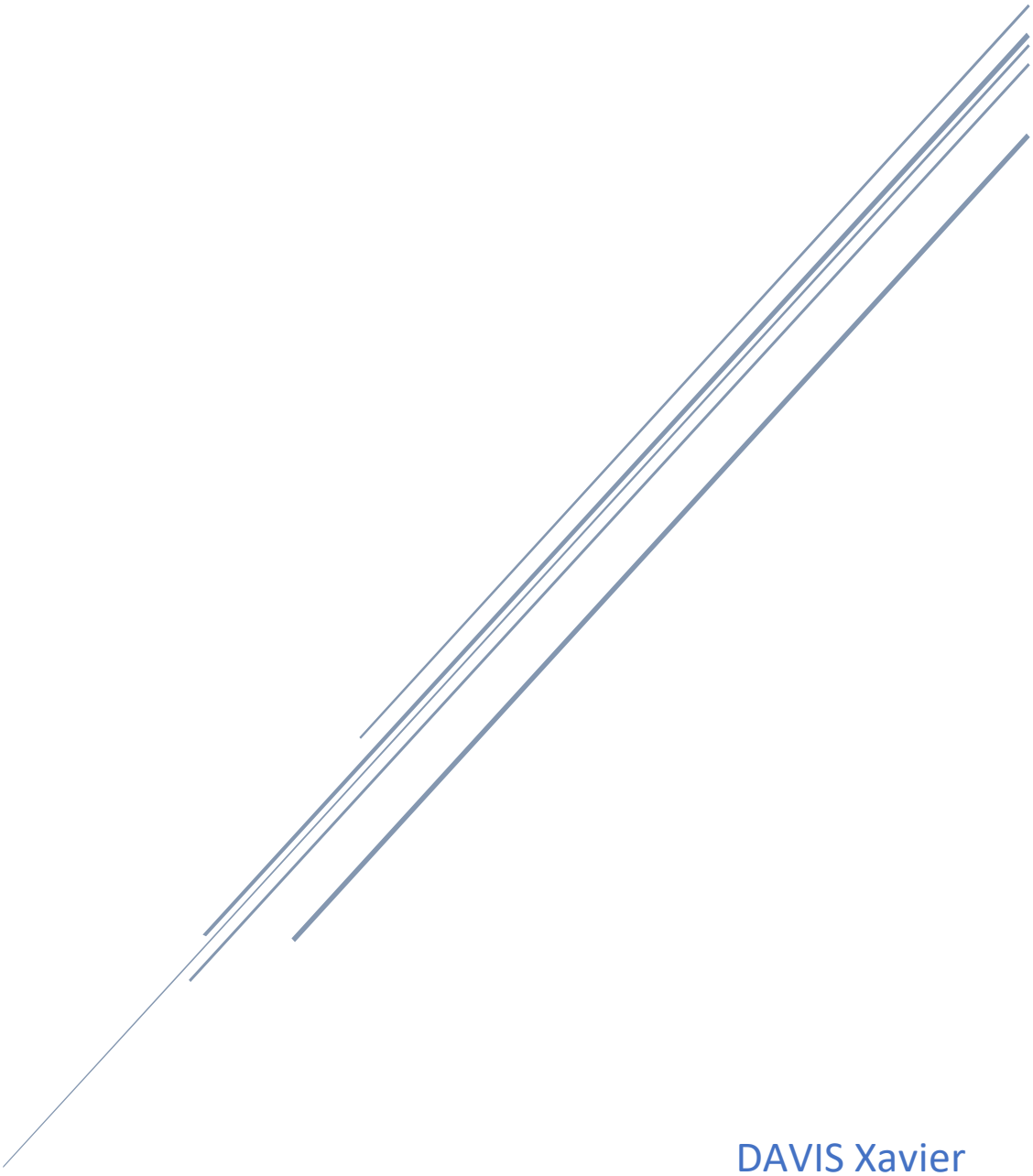
RAPPORT DE STAGE

Amélioration de la chaîne de travail dans le Parc National des Écrins



DAVIS Xavier
Master 2 Double Compétence Informatique et Sciences Sociales
Université Grenoble Alpes

Table des matières

Lexique :

Nomenclature : standardisation de mesure qualitative

Taxon : une espèce dans la classification standard

ORM : Mapping objet-relationnel : interface entre une base de données relationnelle et une application qui simule la programmation-orienté-objet.

1. Contexte du stage

1.1. Le Parc National des Écrins

Situé au cœur des Alpes, à cheval entre le département de l'Isère et celui des Hautes-Alpes, le Parc National des Écrins est une zone de conservation de faune et de flore, qui fête en 2023 ses 50 ans. C'est un des 11 parcs nationaux français, avec ceux de la Vanoise, de Port-Cros, des Pyrénées, des Cévennes, du Mercantour, de la Guadeloupe, de la Réunion, de la Guyane, des Calanques et le parc national de forêts. Le parc possède des sommets culminants jusqu'au 4102 mètres de la Barre des Écrins.

Le parc se divise en multiples sites, avec comme base les Maisons du parc, responsables d'une ou deux zones différentes. présentes dans les zones Il y en a pour le Champsaur-Valgaudemar, pour l'Embrunais, pour la Vallouise-Briançonnais, et pour l'Oisans-Valbonnais. De ces bases travaillent un.e chef.fe de secteur, des assistant.es, des agent.es d'accueil, des technicien.nes et des gardes-moniteur.trices

Le siège du parc se situe au château de Charance, dans le parc de Charance au-dessus de la ville de Gap. C'est là que sont situés la direction, le secrétariat général, le service aménagement, le service d'accueil et de communication, et le service scientifique et son pôle Service Informatique.

Le parc n'est pas seulement la zone géographique, mais également l'organisation et le personnel qui le gère. Les gardes-moniteurs, les scientifiques étudiant la présence de certaines espèces de faune et de flore, et en particulier le Service Informatique. Le monde de l'informatique et celui de la conservation de biodiversité ne sembleraient pas être deux mondes qui vont de pair au premier abord. Le rôle de l'informatique au Parc des Écrins est de fournir aux agents qui vont sur le terrain des outils leur permettant de faciliter la suivie des protocoles scientifiques qu'ils mettent en place.

1.2. Le pôle SI

Le pôle SI aux Écrins est désormais composé de quatre membres :

- Camille Monchicourt, chef de service et géomaticien

C'est lui qui dirige le service, il est impliqué dans les décisions mais aussi dans les grandes releases faites par le pôle SI. Il a été stagiaire au parc avant de l'intégrer en CDI. Camille est également celui qui dirige le SCRUM quotidien.

- Théo Lechémia, développeur

Un autre ancien stagiaire au parc, Théo a effectué un master DCISS avant d'integrér le parc. Il est le développeur le plus expérimenté dans l'équipe actuellement, et est impliqué dans les décisions du pôle, et il s'occupe également en partie du déploiement des outils.

- Vincent Pietri, informaticien :

Il s'occupe de tout le côté hardware au parc, des ordinateurs mais aussi de téléphones portables, et des serveurs. Il peut être en dehors de Gap pour vérifier le bon fonctionnement des systèmes dans les différents secteurs du parc.

- Pierre Narcisi, développeur :

Arrivé au cours du stage, Pierre travaille surtout sur l'outil principal du parc, l'application GeoNature.

1.3. Objectifs du stage

- Développer des formulaires de saisie mobile sur ODK Collect

ODK est un outil qui permet de créer et maintenir des formulaires. Il est utilisé au parc des Écrins pour la digitalisation des protocoles scientifiques. Une application mobile, ODK Collect, existe, et est utilisé pour remplir les formulaires. Cette application permet de fluidifier la chaine de travail.

- Déployer ces formulaires sur les téléphones des agents

Ceci serait simplement le téléchargement des bons formulaires sur les téléphones concernés.

- Assurer la formation des agents (en binôme avec le chargé de mission base de données)

Pour le meilleur fonctionnement possible, les agents doivent savoir exactement comment fonctionne leurs outils.

- Assurer la transmission des données dans l'outil GeoNature

GeoNature et ODK sont deux outils différents gérés par deux organisations différentes. Pour avoir la meilleure chaîne de travail possible, il faut créer un lien entre les deux.

- Intégrer les données historiques des protocoles scientifiques dans GeoNature

Les applications mobiles et web ne sont utilisés pour stocker directement les données des protocoles que depuis peu de temps. Il existe des données datant d'avant la mise en place de ces outils, et il faut trouver un moyen de les avoir dans le même état que les données plus récentes.

- Documenter et illustrer les outils mis en œuvre

La méthode de travail utilisée pour digitaliser les données est en plein changement. Avec des nouveaux outils utilisés, qui ont beaucoup évolués pendant le stage, il est important que la personne qui reprend le mobile sache comment toout fonctionne.

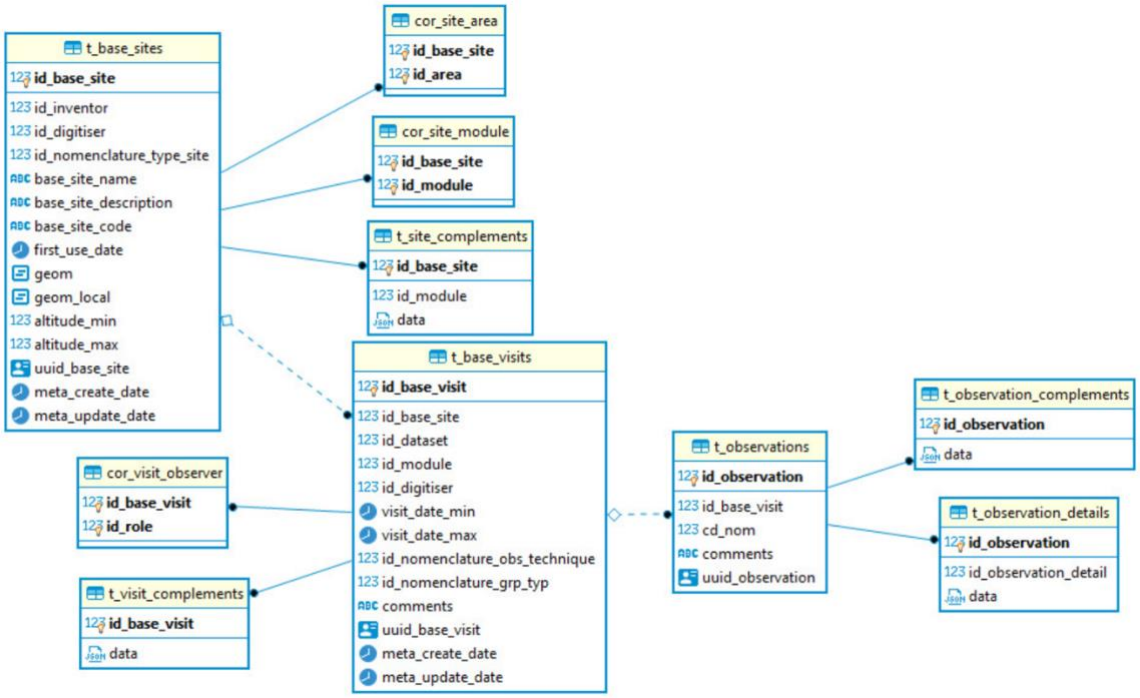
1.4. Technologies utilisés

- GeoNature

L'application GeoNature est celle utilisée par le Parc pour tout ce qui concerne le suivi de la biodiversité. Elle a été développée aux Écrins et mais est utilisée par une centaine d'organisations différentes partout en France. Elle est écrite en Python, utilise le framework Flask, et est open source. Elle est surtout maintenue par le service informatique du parc des Écrins mais d'autres organismes, tel le Parc National des Cévennes et la Ligue de Protection des Oiseaux, y ont également contribués. Cette collaboration permet d'avoir un outil encore plus complet.

GeoNature est composée de modules. Ce sont des petits blocs de l'application qui implémentent certaines fonctionnalités. Le module le plus intéressant pour mon stage s'appelle Monitoring, et concerne certains protocoles scientifiques de suivi d'espèces de faune et de flore. Ces protocoles sont particuliers car ils ont tous la même structure site-visite-observation. Le site est le lieu où l'agent du parc est allé prospecter, une visite est une instance où un agent est allé prospecter sur un site, et une observation est ce qui est observé pour une espèce lors d'une visite. D'autres protocoles existent dans GeoNature, dont celui de flore-prioritaire en particulier, mais ceux-ci n'ont pas là même structure.

Voici le schéma de données du module monitoring.



Nous observons ici les liens entre les différentes tables, qui sont tous reliés, et que lorsque nous ajoutons un élément dans la table des visites et celle des observations, qu'il faut bien que ce soit associé à un site ou à une visite.

- ODK

ODK est un outil externe permettant de créer des formulaires. Il est utilisé partout dans le monde, dans le cas d'études de santé, ou de biodiversité. Il est également utilisé par le parc dans le cadre de l'étude de sa fréquentation. ODK possède une plateforme en ligne de dépôt de données, ODK Central, et une application mobile déjà existante, ODK Collect. Ceci permet au parc d'éviter de devoir embaucher des spécialistes pour développer et maintenir une application interne, ce qui exige plus de dépenses et d'expertise. Ceci signifie également que les formulaires implémentés sont limités par ce qui est fait par les développeurs de l'application.

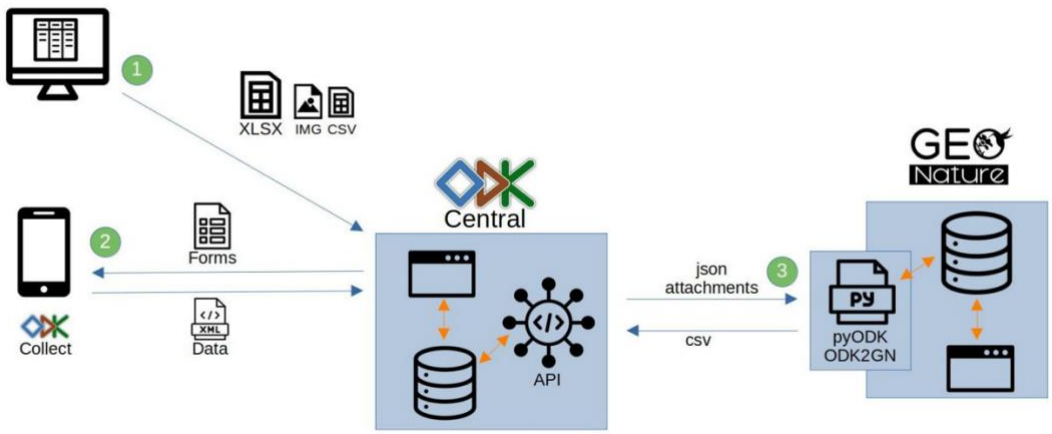
Pour avoir un formulaire prêt à être utilisé, il faut le définir. Ceci peut être fait avec un fichier .xml, mais le format .xlsx est beaucoup plus utilisé car il n'y a pas nécessairement besoin d'être expert en informatique pour le faire. Toute question dans le formulaire a une réponse typée (texte, chiffre, choix dans une liste, champ géographique…). Les listes de choix peuvent être définis soit dans le formulaire, soit dans des fichiers externes. Une fois ce formulaire défini, il est déposé sur ODK Central, dans un projet. Il peut être téléchargé et rempli avec ODK Collect sur un téléphone portable. Une fois rempli avec ODK Collect, les données sont stockées sur le téléphone s'il n'y a pas de connexion Internet, et envoyés à ODK Central au moment où la connexion est rétablie. Il est possible d'utiliser des fonds de carte pour les questions de type géographique, pour une meilleure localisation par un agent.

- ODK2GN

Pour faire le lien entre les deux applications, une librairie Python ODK2GN a été créée fin 2022. Cette librairie permet d'aller accéder à une base ODK Central, et de transposer les données dans GeoNature. Avec cette librairie, l'on peut également mettre à jour les fichiers .csv utilisés pour les choix dans des listes. Ces traitements sont faits par deux commandes lançables dans un terminal.

La librairie qui existait au début du stage était très basique et ne contenait que le code utilisé pour traiter les données, et rien d'autre, pas de tests, pas de tâches de fond. La majeure partie du stage a consisté à étoffer cette librairie pour le rendre plus complet et plus utilisable.

Voici un schéma qui montre comment interagissent les différents éléments.



Le formulaire est d'abord défini en xlsx et envoyé à ODK Central(1), téléchargé, rempli et renvoyé sur ODK Collect (2), et les données sont envoyés en JSON par ODK2GN à GeoNature (3), qui envoie également les fichiers csv dont ont besoin les formulaires à ODK Central.

1.5. Le début du stage

Les missions effectuées pendant la première partie du stage étaient les suivants.

- Montée en compétence sur les technologies utilisées

La plupart des technologies utilisées m'étaient inconnus. Le framework Flask, l'application GeoNature et l'outil ODK étaient nouveaux pour moi, et le langage Python avait été vu en première année de licence, et a été très peu utilisé en master. Le tout début du stage était donc consacré à monter en compétences avec tout ceci.

Ceci a été fait en suivant un tutoriel en ligne, dans le cadre duquel nous créons un site de microblogging. Ce site passe sur de nombreuses fonctionnalités et librairies utilisables dans le cadre d'une application Flask. ODK a également été abordé pour une première fois en créant la possibilité de créer des posts sur ce site avec ODK Collect, et des premiers formulaires très simples.

Ensuite, la prochaine étape du stage était de créer des premiers formulaires pour le module GeoNature Monitoring, en créant un pour le suivi des chauves-souris dans le parc. Ceci a été fait en suivant assez précisément ce qui se fait dans l'application sur l'ordinateur. Ceci a été fait et testé « à la main » avec des vraies données. La même chose a été faite pour le suivi des placettes de nard.

La suite de la première partie du stage a été dédiée à créer un formulaire pour pouvoir implémenter le protocole de flore-prioritaire. Ce protocole n'était pas de type Monitoring, il a donc nécessité un traitement particulier des données reçues par ODK. Ce protocole a ses propres versions des fonctions de traitement des données et de mise à jour des fichiers csv utilisés dans les questions formatées avec un choix dans une liste.

À la toute fin de la partie du stage avant la première soutenance, une première phase de test a été entamée. Quelques premiers tests ont été créer pour tester les fonctions les plus simples du code.

Quasi pleine lune du 30 janvier 2018 depuis la route du col de Gleize : vue sur le sommet de Entre Piniers (3044m) © P. Saulay - PNE

2. La deuxième partie du stage

2.1. La première phase de tests

Le code de la librairie odk2gn n'est pas très facile à tester « à la main ». Il faudrait aller dans ODK, créer une nouvelle soumission, afficher le contenu de la soumission dans le terminal, aller vérifier les clés et les valeurs, et voir qu'il y a une correspondance avec la valeur et ce qui a été entré par l'utilisateur. De plus, cette démarche est longue en termes de temps. Il faudrait un moyen plus efficace pour tester le code.

Un test unitaire est ‘’un processus de vérification d'unique unité de logiciel’’. En effet, il s'agit de tester une fonction en particulier du code, et de le comparer à un résultat attendu défini. Plus encore, tester le code permet de le rendre plus maintenable, mais aussi de gagner du temps de débogage. Il permet également une meilleure compréhension du code existant, car s'il y a des erreurs, il faut voir exactement pourquoi.

Une fixture est un élément de code qui permet de tester plusieurs fonctions dans un environnement consistent et fiable. Il s'agit dans notre cas d'objets de la plupart des classes qui interagissent. Parmi les fixtures d'odk2gn, il y a des plus petits éléments, comme un taxon ou un utilisateur, mais il y a également des fixtures pour des éléments plus compliqués à tester par leur complexité, telle la configuration d'un sous-module monitoring.

Dans notre implémentation, nous avons beaucoup de fixtures. Ils peuvent être des éléments très petits, tel un taxon ou une nomenclature, ou plus gros, comme une soumission comme celles tirées d'ODK Central lors de la synchronisation. Ils sont faits de manière à être le plus simple possible pour un objet de chaque classe, mais assez complexes pour répondre à tous nos besoins lors de la phase de test.

Une fixture peut être utiliser pour mocker une fonction. Mocker une fonction signifie lui fixer une valeur de sortie, et de faire en sorte que si pendant un test une fonction va provoquer une erreur car elle n'est pas testable, nous pouvons faire en sorte que non seulement elle ne va pas poser de problèmes lors de la phase de test, mais également qu'elle va renvoyer un résultat stable.

Dans nos tests unitaires, nous avons commencé par tester les plus petites fonctions de récupération et de données depuis la base. Ces fonctions sont utilisées pour créer les fichiers csv qui sont téléversés avec les formulaires. Pour tester ces fonctions, nous vérifions qu'ils renvoient bien des listes non vides, et que dans la liste nous avons bien les noms de clés corrects pour chaque élément. Nous avons également vérifié que les valeurs de retour étaient bien celles des fixtures créées pour tester notre code.

Le test de la fonction pour récupérer les nomenclatures a été la plus compliquée à effectuer. En effet, ceci nous exige à aller regarder la configuration des sous-modules. Le problème est qu'un sous-module fixture n'a pas de configuration existante, ce qui serait presque impossible à récupérer sans mocker. Toutes les autres fonctions de récupération de valeurs, cependant, ont pu être testés lors de la première phase de tests.

La phase de test a également exigé de tester les plus grandes fonctions existantes dans la librairie. Celle pour tester la synchronisation des données est la plus délicate. Les tests de cette fonction ont exposé la plus grosse limite de ces tests : il est impossible de tester pour de vrai la connexion avec ODK. Pour au moins tester la partie relative au traitement des données qui sont reçues par les requêtes sur cette plateforme, il a été nécessaire de « mocker » les fonctions qui traitaient de ceci, ainsi que celles qui cherchaient la configuration d'un sous-module Monitoring qui est seulement existante en version fixture. Cependant, une première version de test unitaire de cette fonction a été faite.

Bien qu'elle soit une fonction plus petite, la fonction de mise à jour des fichiers joints aux formulaires n'est que très peu testable. En effet, cette fonction ne fait que mettre en mode brouillon le formulaire en ligne, puis il construit les nouveaux fichiers csv, et enfin les joint au formulaire, et le republie en version finie. Il a été expliqué plus haut pourquoi les fonctions de récupération et création du fichier des nomenclatures est plus compliqué, et avec les deux autres fonctions étant liées à ODK, cette fonction n'a été traitée que pendant la deuxième phase de test.

2.2. Implémentation de la création de sites dans ODK2GN

Dans GeoNature, nous avons un module nommé Monitoring. Ce module regroupe un ensemble de protocoles scientifiques représentés par des sous-modules. Ces protocoles ont la particularité de tous avoir la même structure site-visite-observation. Le site est la zone où l'on va prospecter lors du protocole. Ils peuvent appartenir à des groupes ou non. La visite représente une instance où l'on est allé prospecter sur un site. Une observation représente ce qui est observé pour une espèce

Dans l'implémentation existante, lors de l'utilisation d'ODK Collect dans le cadre d'un de ces protocoles, le protocole était de choisir le site où l'on prospectait parmi une liste prédéfinie, et de créer dans l'application mobile une visite et éventuellement une ou plusieurs observations. Pour le Parc, les sites sont prédéfinis afin de donner aux agents les coordonnées spécifiques à prospecter, qui sont des lieux plus précis, et facilement visitables à plusieurs reprises. Cependant, GeoNature est bien une application développée au Parc des Écrins, mais ce n'est pas la seule organisation qui l'utilise.

Une des organisations qui utilise GeoNature est la Ligue de Protection des Oiseaux (LPO). Ils se sont décidés en plus à utiliser la librairie ODK2GN pour la mise en place de certains protocoles en mobile, dont au moins un où figure dans le protocole la création de nouveaux sites. Une réunion entre les services informatiques de la LPO et du Parc des Écrins a eu lieu, où il a été convenu que la mise en place de cette possibilité serait faite par le Parc, qui possédait plus de familiarité avec la librairie où serait implémentée cette fonctionnalité.

Nous avons décidé de faire des versions adaptées de deux formulaires déjà existants afin d'avoir quelque chose de testable par une personne. Les formulaires en question sont celui pour les oiseaux de montagne et celui pour les placettes de nard. Ces protocoles utilisent deux sites avec des types géographiques différentes, un utilise des points, alors que pour le suivi du nard, les géométries sont des polygones, et donc à traiter de manière différente.

Pour l'implémentation de la création de sites, il a été convenu de ne pas réinventer la roue, et de créer une fonction qui parse la soumission ODK, en tire les entrées utilisées pour créer un site, et crée une nouvelle entrée dans la base de données GeoNature, exactement de la même manière que pour créer et mettre en base de données les visites et les observations. Les noms des clés de dictionnaire à utiliser sont fournis dans le fichier de configuration de la librairie. Une particularité des sites vient du fait qu'ils peuvent être en groupes. Une fonction pour lister les groupes de sites a été faite et est utilisée dans la création de fichiers csv contenant les valeurs à utiliser pour les formulaires, comme pour les autres listes.

2.3. Taches de fond

2.3.1. Menu de configuration

L'idée de faire ces fonctions de mise à jour des formulaires est de pouvoir les mettre en tâche de fond. Nous pouvons mettre à jour ces fichiers manuellement, mais ici on ne peut pas nécessairement savoir quand est-ce que des nouvelles données ont été envoyés à ODK Central, ou bien quand est-ce qu'un nouvel élément de liste a été créé et si il doit être intégré à un des fichiers csv envoyés joint au formulaire. Ce serait le cas par exemple quand une nouvelle personne intègre l'équipe du parc, ou quand une nouvelle espèce est à prospecter dans un protocole scientifique. Ainsi, lancer ces fonctions en tâche de fond de manière régulière éliminerait ces problèmes. La librairie initiale ne contenait rien qui permettait de faire ceci au moment de prise en main, à l'exception d'une première version des commandes à lancer. Cette mise en place à donc nécessité une assez grande revue de tout le code existant.

Afin de faire ceci, nous avons décidé de créer des tables dans la base de données représentant les formulaires ODK dans la base de données. En effet, sans faire ceci, c'est très compliqué de garder trace et de configurer les paramètres à fournir dans les fonctions faisant le lien entre notre base de données et ODK Central. Nous avons donc créé dans la base de données un schéma ODK2GN et une table t_odk_forms pour stocker ces informations, ainsi qu'un fichier de migration de base de données contenant le code pour son installation ou sa desinstallation. Cette table contient des enregistrements de type TOdkForms. Ces enregistrements possèdent comme attributs un id pour la base de données, la chaîne de caractères d'id de formulaire qui définit l'URL pour ce formulaire, l'id du projet ODK, qui contient le formulaire (et qui lui aussi est dans l'URL du formulaire), l'id de base de données du module (ou sous-module) auxquels les formulaires sont associés en clé étrangère, et le nom des fonctions de synchronisation et de mise à jour des formulaires. En effet, ces noms de fonctions sont celles qui indiquent si on fait le traitement pour un sous-module de monitoring ou bien si c'est un sous-module à part. Nous avons ainsi une manière de stocker les paramètres à utiliser lors du lancement des tâches de fond.

Cependant, nous ne voulons pas que ces paramètres soient configurés depuis le SGBD utilisé dans le cadre de notre installation. Il faut donc un moyen de pouvoir avoir un menu pour pouvoir gérer nos enregistrements de formulaires. ODK2GN étant désormais un sous-module GeoNature à part entière, nous avons décidé d'utiliser la même méthode qui existe dans l'application GeoNature pour configurer notre sous-module. Ceci est fait avec l'aide de la libraire Flask-Admin.

Flask-Admin est une librairie de fonctionnalités permettant d'avoir un modèle CRUD qui s'associe à une base de données pour créer des interfaces d'administration d'objets de manière simple, en partant d'un modèle de données existant. Le fonctionnement en effet est de transformer les vues en classes Python-Flask. Cette librairie contient une classe ModelView qui est utilisé comme superclasse pour toute vue que nous voulons créer. Dans chaque classe de vue, nous pouvons configurer quelles méthodes de CRUD sont implémentables. Nous pouvons associer ces vues à des classes d'objets existants, et à la base de données utilisée.

L'implémentation de Flask-Admin est très simple. En effet, notre vue d'administration va intégrer le menu d'administration du back office de GeoNature qui existe déjà, donc tout ce qui est relatif à la sécurité est déjà existant, et tout ce qui est relatif au fait que nous travaillons dans une application Flask aussi. Dans un fichier *admin.py*, Nous avons donc créé une classe OdkFormModelView, qui hérite de la superclasse ModelView de Flask-Admin. Notre classe contient seulement deux éléments qui indiquent qui a aura accès à cette vue.Il n'existe rien d'autre dans la classe, pour notre librairie ceci suffit. Les méthodes de la classe mère feront le reste du travail pour nous.

Nous ajoutons au menu d'administration de GeoNature une vue de la classe que nous venons de créer, qui utilise le modèle TOdkForm, qui est associé à la base de données de GeoNature, qu'on a décidé d'appeler « Formulaires ODK », et que nous avons décidé d'associer à une catégorie de vues « odk2gn ». Ce nom de catégorie apparait dans le menu d'administration du back office. Notre vue est désormais utilisable pour créer nos objets. Lors de l'utilisation de notre nouvelle vue, l'id du formulaire comme donné à ODK, l'id du projet ODK, et les noms de commandes

sont à fournir dans des champs textes, alors que le module associé est choisi parmi la liste de tous les modules installés dans notre application GeoNature.

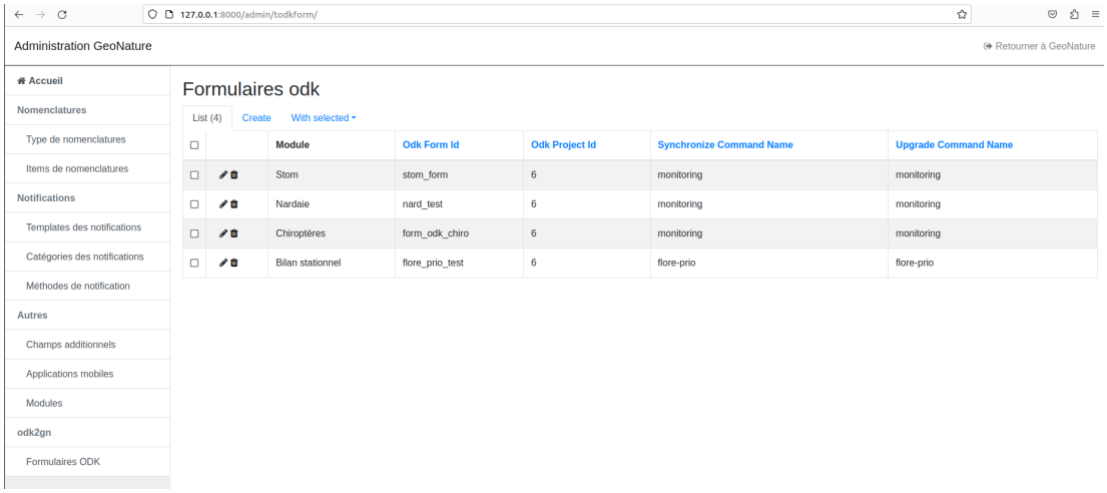
Voici à quoi ressemble le fichier admin.py

```python
from odk2gn.models import TOdkForm
from odk2gn.gn2_utils import get_monitoring_modules


class OdkFormModelView(CruvedProtectedMixin, ModelView):
    module_code = "ADMIN"
    object_code = "ODK2GN"


flask_admin.add_view(
    OdkFormModelView(model=TOdkForm, session=DB.session, name="Formulaires ODK", category="odk2gn")
)
```

Ceci est tout le code nécessaire à pour créer la vue.
Voici la vue qui résulte de ce code.



Le tableau contient tous les éléments nécessaires à faire tourner les fonctions dans les tâches de fond.

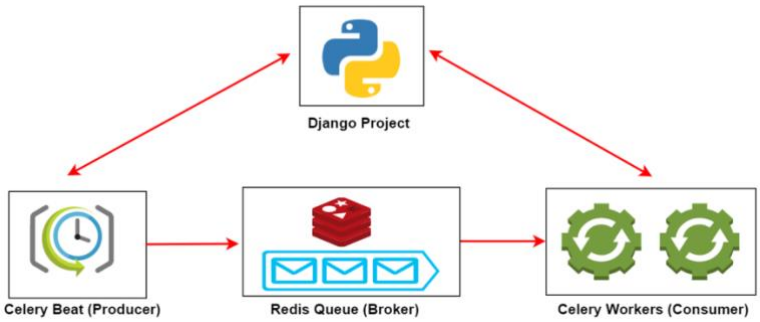
2.3.2. Celery et les tâches de fond

Une file d'attente de tâches est une série de travaux qui peuvent être effectués sur un ou plusieurs ordinateurs, de manière asynchrone. Ils peuvent être séparés sur des threads différents de manière à être exécutés de manière synchrone ou de manière asynchrone. Ces tâches sont exécutées par des workers. Un worker est un processus dédié pour traiter les tâches. Les messages contenant les tâches à exécuter sont envoyés par une application client. Un broker reçoit les messages et créé la file d'attente, en vérifiant que deux tâches qui ne peuvent pas être lancés en même temps ne le sont pas.

La librairie Celery est une libraire de gestion de tâches. Elle contient des workers, mais aussi la possibilité de planifier le moment d'exécution des taches, avec Celery-beat. Celery et Celery-beat proposent la possibilité de travailler avec plusieurs brokers différents, dont Redis, qui est utilisé par GeoNature. Nous utilisons Celery-

beat pour signaler quand est-ce que les tâches doivent être lancés, le broker Redis pour envoyer les tâches à Celery, qui est utilisé comme worker ici.

Voici un schéma du fonctionnement de Celery-beat.



À l'heure indiqué, Celery-Beat envoie les tâches, définis dans le projet Python, au broker (Redis), qui en fait une file d'attente, et les envoie au workers dans l'ordre de la file. Les workers les exécutent dans le cadre du projet.

Deux fonctions ont été créés dans un fichier *tasks.py* dédié. La première fonction synchronise tous les modules possédant un formulaire ODK associé, et le deuxième met à jour les fichiers joint aux formulaires ODK correspondants. Une particularité ici est que les fonctions à lancer sont utilisées dans des commandes CLI. Après que la base soit synchronisée pour un module, ou que les fichiers sont mis à jour, nous avons choisi de faire attendre deux secondes avant de faire les mêmes opérations afin de limiter le nombre de processus qui s'exécutent en même temps.

La limite de ceci vient du fait que le server ODK Central contient un historique des versions pour chaque formulaire, donc il faut être vigilant à ne pas le surcharger. Si on met à jour les fichiers csv pour quatre sous-modules tous les jours, il y aura au moins 1400 versions de formulaires à stocker par an, ce qui n'est pas optimal en termes de stockage de données.

Le fichier crontab dans un système Linux définit l'heure ou la fréquence à laquelle une tâche est lancée. Dans une ligne de ce fichier figure la commande à lancer, mais également quand est-ce qu'il est lancé. Le minutage est défini par une chaine de caractères composée de cinq éléments séparés par des espaces. Celery propose une fonction qui a le même rôle, que nous utilisons pour nos tâches de fond. Bien que ces éléments soient définis comme des paramètres explicites de fonctions, ce qui est utilisé comme paramètre est stocké dans le fichier de configuration, dans la même forme qu'une chaine crontab. Nous avons profité de ce fait pour revoir ce fichier de configuration.

Voici à quoi ressemblent ces chaînes de caractères. Celui-ci est dans le fichier de configuration exemple a copier lors de l'installation.

```
[tasks]
    synchronize_schedule = "0 0 * * *"
    upgrade_schedule = "0 0 * * *"
```

Ces deux chaînes indiquent qu'il faudrait lancer les fonctions de synchronisation des données de la base et de mise à jour des fichiers csv tous les jours à minuit.

Notre fichier de configuration est en format TOML. Ce format est utilisé particulièrement pour les fichiers de configuration. C'est un format fait à la fois pour être facilement lisible pour les humains, mais également pour être facilement passable en élément hashable, comme le JSON ou comme un dictionnaire Python. Le fichier de base possède des éléments liés à ODK Central, une adresse électronique où sont envoyés les notifications d'erreurs lors de la synchronisation, et quelques éléments indiquant le nom des éléments de formulaire qui peuvent être modifiés. Tout ce qui est des adresses électroniques n'a pas changé lors du passage au nouveau fichier de configuration. Un élément contenant les chaines crontab pour les tâches est ajouté. Les éléments pour la synchronisation des divers modules monitoring sont dans un tableau, et le code de chaque module est ajouté. Nous avons ainsi un fichier de configuration qui couvre tout le code qu'il faut.

2.4. Deuxième phase de tests

2.4.1. Tests unitaires

La première phase de test a testé la première version de la fonction de synchronisation, ainsi que la plupart des fonctions de récupération de données dans la base de GeoNature. Cependant, avec beaucoup d'évolutions assez considérables pour ODK2GN, une deuxième phase de tests a été nécessaire pour tester les nouveautés, ainsi que pour compléter la première phase, et pour aller beaucoup plus loin que les tests unitaires.

Tout d'abord, les nouveautés par rapport à la création de sites ont été testées. La nouvelle fonction ressemble à celle existante pour créer les visites et les observations. La chose la plus importante à vérifier est que le site est bien créé, peu importe la structure de la géométrie qui le définit. Ceci n'a pas été très facile, car nous avons butté sur un problème avec notre traitement des données géographiques.

Les données géographiques sont fournies dans la soumission depuis ODK en format GeoJSON. Ce format attribue à une géométrie un type, défini dans un dictionnaire avec la clé 'type', et des coordonnées dans un tableau. Ces coordonnées, comme fournies par ODK ont des valeurs x et y pour la longitude et la latitude, une valeur z pour l'altitude, et une valeur de précision. Cependant, la valeur de précision des points ne nous intéresse absolument pas, et l'altitude est calculée dans la base GeoNature. De plus, dans la base GeoNature, les données sont en format WKT, un format différent de géométrie. Il faut donc trouver un moyen de reformatter les données.

Nous avions eu une version initiale de la fonction de formatage des géométries. Il enlevait les données en trop dans les coordonnées, et ensuite le faisait passer par une librairie nommée Shapely, pour le transformer en Shape, classe depuis laquelle elle peut être transformée en WKT. Cependant, notre première version ne fonctionnait pas systématiquement et le code n'était pas du plus propre. Nous avons donc modifié la fonction pour qu'elle soit meilleure et en renvoyant un format WKB (WKT en binaire, utilisé dans d'autres modules de GeoNature. De plus, les fixtures étaient trop basiques pour les tester. Nous avons donc créé un grand nombre de fixtures supplémentaires, dont points et polygones avec deux, trois et quatre coordonnées.

En testant avec ces fixtures, nous avons pu observer qu'avec un point, la nouvelle méthode de formatage fonctionnait, mais que ce n'était pas le cas avec un polygone avec 4 valeurs par coordonnée. Cependant, avec trois valeurs par coordonnée, il n'y avait pas de problèmes. La fonction de formatage a encore été adaptée , enlevant la précision pour les géométries où ceci est nécessaire. Ce problème a donc été résolu.

Voici la nouvelle fixture pour le polygone avec quatre valeurs par coordonnées

```python
@pytest.fixture(scope="function")
def polygon_4():
    polygon = {
        "geometry": {
            "type": "Polygon",
            "coordinates": [
                [
                    [6.0535113, 44.5754145, 0, 0],
                    [6.0535109, 44.5754135, 0, 0],
                    [6.0535120, 44.5754150, 0, 0],
                    [6.0535113, 44.5754145, 0, 0],
                ],
            ],
        }
    }
    return polygon
```

Dans la première phase de test, il n'y avait pas énormément de tests sur les exceptions lancées lors de la synchronisation des données entre ODK et GeoNature. Nous avons donc créé plus de fixtures avec des soumissions semblables à ceux d'ODK. Ces fixtures sont en effet des grands dictionnaires, contenant plusieurs autres fixtures, ou des valeurs quelque peu bidon. Une première version a été faite avec des valeurs choisis précisément pour que la synchronisation fonctionne normalement. Une deuxième à été créée avec un mauvais formatage, et un troisième avec une mauvaise donnée qui provoque une erreur.

La première version de ce test finit avec une assertion que le code de sortie est 0 (ce qui signifie que la synchronisation s'est bien produite). Pour les autres, nous vérifions qu'une exception de la bonne classe a été lancée, et qu'il y ait bien une erreur qui renvoie le code 1.

Afin de bien pouvoir tester la récupération des nomenclatures, nous avons dû créer une fixture représentant la configuration du sous-module monitoring. Les fichiers de config étant en JSON, nous avons donc crée un énorme dictionnaire qui ressemble à ceux pour la config, mais en plus simple. Nous avons créé une question avec un

type nomenclature, qui demande qu'il y ait une liste de nomenclatures d'un certain type. Le type en question est celui de la nomenclature fixture. Nous avons ainsi les éléments nécessaires pour que le test de récupération des nomenclatures dans la base de données fonctionne encore.

Voici une capture d'écran avec un extrait de la fixture représentant la configuration du sous-module fixture.

Ce qui est à noter ici est ce qui est dans la nomenclature, avec ces lignes de code nous définissons que les nomenclatures que nous voulons avoir dans un éventuel fichier csv associé seraient ceux avec le type donné dans la fixture, et ayant le code de la nomenclature fixture.

Nous sommes également revenus sur les tests pour le sous-module flore-prioritaire. Ce sous-module est désormais séparé du reste du code, de manière à qu'il soit installable seulement s'il y a besoin. Les mêmes modifications faites au code pour les données géographiques à été reproduite ici. Les tests n'étaient pas tous fonctionnels à la fin de la première phase de tests, avec les mêmes problèmes pour les nomenclatures et pour la mise à jour des fichiers csv. Avec des modifications très semblables à celles effectuées pour le reste des tests, nous avons le bon fonctionnement des tests ici.

Cependant, tout le code n'est pas testable. Tout ce qui concerne l'automatisation de la synchronisation de la base et la mise à jour des fichiers en tâches de fond est difficile à tester, les fonctions liées aux tâches de fond n'ayant pas de paramètre. Mais dans les tests, le fait que nous devons simuler la liaison avec ODK est peut-être le plus imparfait, car une majeure partie du code y est liée.

2.4.2. Tests d'intégration

Enfin, pour voir que tout s'installe comme il faut à chaque modification, et que le code fonctionne, nous avons mis en place des tests d'intégration avec les GitHub Actions. Les GitHub Actions sont un moyen de lancer des routines lors de la mise à jour du code, ou d'une autre action. Dans notre cas, il s'agirait de lancer les tests lors d'un commit sur la plateforme.

Le fichier contenant ce qui est à lancer est un fichier de format yml. Ce fichier est un autre moyen d'avoir quelque chose avec une structure clé : valeur, mais lisible pour un humain. Un fichier contient une série d'actions qui porte un nom et qui se lance lors de certains évènements sur son repository GitHub (push ou pull request par exemple). Les actions sont dans un bloc « jobs ». Ce bloc définit d'abord l'environnement virtuel dans lequel il est lancé, ou le « build ». Cette partie contient les versions du software nécessaire dans l'environnement virtuel pour le bon fonctionnement de ce qui suit. Il définit ensuite dans un blocs « services » d'autres extensions nécessaires, par exemple pour les données géographiques dans la base de données.

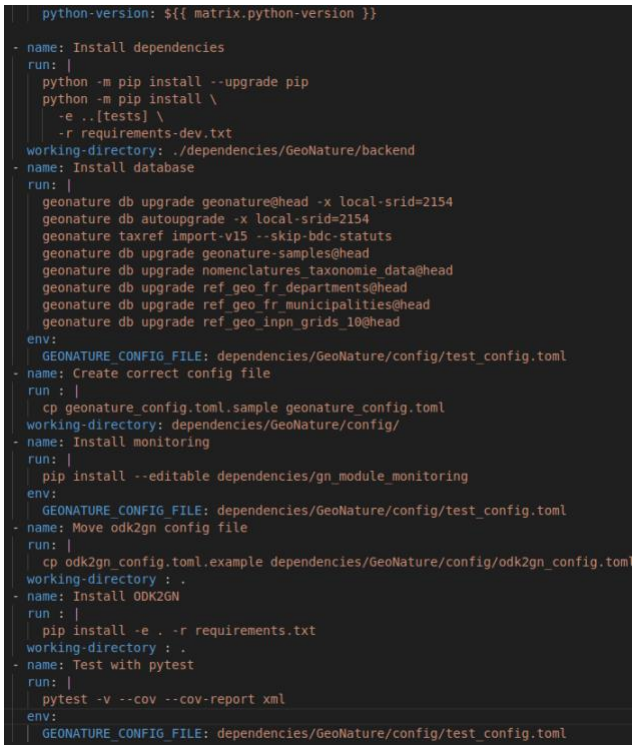
Les tâches à lancer sont dans un bloc « steps » à l'intérieur du bloc « build ». Chaque tâche porte un nom, comporte un bloc « run » dans lequel sont écrits les commandes à lancer et se produit dans un environnement spécifique, et peut également être spécifié à être lancé dans un directory en particulier, si ceci est nécessaire. L'environnement dans lequel tout est effectué est exactement comme si un environnement semblable était installé sur un ordinateur, il faut donc consacrer certaines étapes aux installations nécessaires.

Si notre librairie dépend d'autres librairies , ceux-ci s'appellent des sous-modules, et doivent être installés dans l'environnement également. La commande « git submodule add <URL> » permet d'ajouter un sous-module à installer. Nous avons plusieurs sous-modules à lancer choisi de regrouper tous les sous-modules dans un seul dossier.

Seulement une fois que tout est installé vient le moment ou nous pouvons définir les tâches à lancer. Pour nous, ceci est simplement la commande qui lance les tests. Si les tests passent, les modifications sont bonnes et n'ont pas affecté le bon fonctionnement du code.

Pour nous, une fois que nous avons défini le système sur lequel notre environnement virtuel sera, nous installons toutes les dépendances, puis la base de données et ses extensions, puis ODK2GN, et enfin nous lançons les tests.

Voici un extrait du code pour les tâches lancées dans les tests d'intégration.



```
    python-version: ${{ matrix.python-version }}

- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    python -m pip install \
      -e ..[tests] \
      -r requirements-dev.txt
  working-directory: ./dependencies/GeoNature/backend
- name: Install database
  run: |
    geonature db upgrade geonature@head -x local-srid=2154
    geonature db autoupgrade -x local-srid=2154
    geonature taxref import-v15 --skip-bdc-statuts
    geonature db upgrade geonature-samples@head
    geonature db upgrade nomenclatures_taxonomie_data@head
    geonature db upgrade ref_geo_fr_departments@head
    geonature db upgrade ref_geo_fr_municipalities@head
    geonature db upgrade ref_geo_inpn_grids_10@head
  env:
    GEONATURE_CONFIG_FILE: dependencies/GeoNature/config/test_config.toml
- name: Create correct config file
  run : |
    cp geonature_config.toml.sample geonature_config.toml
  working-directory: dependencies/GeoNature/config/
- name: Install monitoring
  run: |
    pip install --editable dependencies/gn_module_monitoring
  env:
    GEONATURE_CONFIG_FILE: dependencies/GeoNature/config/test_config.toml
- name: Move odk2gn config file
  run: |
    cp odk2gn_config.toml.example dependencies/GeoNature/config/odk2gn_config.toml
  working-directory : .
- name: Install ODK2GN
  run : |
    pip install -e . -r requirements.txt
  working-directory : .
- name: Test with pytest
  run: |
    pytest -v --cov --cov-report xml
  env:
    GEONATURE_CONFIG_FILE: dependencies/GeoNature/config/test_config.toml
```

Ici nous avons les installations des dépendances, de la librairie à tester, et le lancement de la commande des tests.

2.5. Intégration de de données historiques

La dernière tâche du stage était d'intégrer des anciennes données relatives au protocole STOM, celui de suivi des oiseaux de montagne, dans GeoNature. Ces données sont stockées dans des fichiers en format .ods, qui ne sont pas le plus facile à traiter. Pour les données antérieures à 2021, tout est dans un seul grand fichier, avec les données 2021 étant dans des plus petits fichiers, avec un fichier par secteur.

La première étape est d'interpréter ce que représente chaque ligne dans le fichier. Chaque ligne contient beaucoup d'information, dont le secteur du parc, le nom du site, la date et l'heure de la visite, le nom de l'agent qui a effectué la visite, les conditions lors de la visite, des éléments descriptifs du site pendant la visite, l'espèce observée, et le comptage de chaque espèce de la manière exigée par le protocole.
Nous avons déterminé que chaque ligne dans un document représente une observation effectuée pendant une visite. Une visite regroupe toutes les observations effectuées pendant une période un jour particulier par un ou deux agents.

Voici un extrait d'un des fichiers de données :

	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF	AG	AH
1	COUV	NUAGE	PLUIE	VENT	VISI	DENEIGEMT	ESPECE	NB05	NB510	NB100	MAX10														
2		1	1	1	1	100	Pipit spioncelle	3	3	0				Liste espèces		ELEMENT PAYSAGE 1	ELEMENT PAYSAGE 2	ELEMENT PAYSAGE 3	% ROCH	% SOL NU	% HERB	%ARBRISSEAU	%ARBRISSEAU_1M	%ARBUSTE	%ARBRES	RQ
3		1	1	1	1	100	Traquet motteux	1	2	0				Pipit spioncelle		2			25	5	70	0	0	0		0Gros bloc et cairn plaqué
4		1	1	1	1	80	Traquet motteux	1	1	0				Traquet motteux		2			25	5	70	0	0	0		0Gros bloc et cairn plaqué
5		1	1	1	1	80	Rouge-queue noir	1	1	0				Rougequeue noir					10	0	90	0	0	0		0caillou dans pelouse, cairn
6		1	1	1	1	80	Monticole de roche	0	0	1				Accenteur mouchet					10	0	90	0	0	0		0caillou dans pelouse, cairn
7		1	1	1	1	80	Pipit spioncelle	1	0	0				Alouette des champs					10	0	90	0	0	0		0caillou dans pelouse, cairn
8		1	1	1	1	80	Accenteur alpin	0	0	1				Chocard a bec jaune					10	0	90	0	0	0		0caillou dans pelouse, cairn
9		1	1	1	1	90	Accenteur alpin	0	0	1				Linotte mélodieuse					10	0	90	0	0	0		0caillou dans pelouse, cairn
10		1	1	1	1	90	Traquet motteux	1	1	0				Tarier des prés					5	0	95	0	0	0		0source et piquet bois
11		1	1	1	1	90	Pipit spioncelle	1	1	0				Crave a bec rouge					5	0	95	0	0	0		0source et piquet bois
12		1	1	1	1	50	Rouge-queue noir	1	1	0				Accenteur alpin					5	0	95	0	0	0		0source et piquet bois
13		1	1	1	1	50	Accenteur alpin	1	2	0				Pipit des arbres					70	0	30	0	0	0		0bord de casse, cairn
14		1	1	1	1	95	Alouette des champs	1	1	1				Merle a plastron					70	0	30	0	0	0		0bord de casse, cairn
15		1	1	1	1	95	Pipit spioncelle	2	2	0				Pinson des arbres					10	0	90	0	0	0		0tertier de marmotte, piquet boi
16		1	1	1	1	100	Pipit spioncelle	2	2	0				Verdurion montagnard					10	0	90	0	0	0		0tertier de marmotte, piquet boi
17		1	1	1	1	100	Traquet motteux	2	2	0				Monticole de roche					10	0	90	0	0	0		0cairn et piquet bois
18		1	1	1	1	100	Traquet motteux	1	1	0				Bruant ortolan					10	0	90	0	0	0		0cairn et piquet bois
19		1	1	1	1	100	Pipit spioncelle	2	2	0				Troglodyte mignon		8			5	0	95	0	0	0		0tertier de marmotte et cairn, gr
20		1	1	1	1	100	Alouette des champs	0	0	1				Fauvette babillarde		8			5	0	95	0	0	0		0tertier de marmotte et cairn, gr
21		1	1	1	1	100	cincle plongeur	0	0	1				Bruant jaune		8			5	0	95	0	0	0		0tertier de marmotte et cairn, gr
22		1	1	1	1	80	Pipit spioncelle	2	2	0				Niverolle alpine		8			5	0	95	0	0	0		0tertier de marmotte et cairn, gr
23		1	1	1	1	80	Traquet motteux	1	1	0				Sizerin flamme					5	0	95	0	0	0		0herbe verte, piquet bois et lam
24		1	1	1	1	80	Rouge-queue noir	1	1	0				Corneille noire					5	0	95	0	0	0		0herbe verte, piquet bois et lam
25		1	1	1	1	100	Rouge-queue noir	0	0	1				Grive draine					5	0	95	0	0	0		0herbe verte, piquet bois et lam
26		1	1	1	1	100	Traquet motteux	2	2	0				Faucon crecerelle					5	0	20	75	0	0		0cairn près d'un bloc plat
27		1	1	1	1	100	Pipit spioncelle	2	2	0				Perdrix bartavelle					5	0	20	75	0	0		0cairn près d'un bloc plat
28		1	1	1	1	100	Pipit spioncelle	2	2	0				Mesange noire					5	0	20	75	0	0		0cairn près d'un bloc plat
29		1	1	1	1	100	Traquet motteux	1	1	0				Bruant fou		8	2		2	1	97	0	0	0		0cairn sur caillou plat et piquet,
30		1	1	1	1	100	Tarier des prés	1	1	0						8	2		2	1	97	0	0	0		0cairn sur caillou plat et piquet,
31		1	1	1	1	100	Chocard a bec jaune	0	0	2						8	2		2	1	97	0	0	0		0cairn sur caillou plat et piquet,
32		1	1	1	1	100	Linotte mélodieuse	0	0	1						8	2		2	1	97	0	0	0		0cairn sur caillou plat et piquet,
33		1	1	1	1	100	Traquet motteux	1	1	0				Hirondelle de rochers		8	2		2	1	97	0	0	0		0cairn sur caillou plat et piquet,
34														Coucou gris											
35														Pipit rousseline											
36		1	1	1	1	100	Pipit spioncelle	1	1	0				Pouillot de bonelli		4			2	1	95	0	0	1	1	
37		1	1	1	1	100	Pie bavarde	0	0	1				Caille des blés		4			2	1	95	0	0	1	1	
38		1	1	1	1	100	Pinson des arbres	0	0	1				Pouillot véloce		4			2	1	95	0	0	1	1	
39		1	1	1	1	100	Alouette des champs	0	0	1				Merle noir		4			2	1	95	0	0	1	1	
40		1	1	1	1	100	Traquet motteux	0	0	1				Serin cini		2			0	5	95	0	0	0	0	
41		1	1	1	1	100	Pipit spioncelle	2	2	0				Milan royal		2			0	5	95	0	0	0	0	
42		1	1	1	1	100	Alouette des champs	0	0	1				Cincle plongeur		2			0	5	95	0	0	0	0	
43		1	1	1	1	100	Pipit spioncelle	1	1	0				Hirondelle de fenetre		2			0	6	94	0	0	0	0	
44		1	1	1	1	100	Alouette des champs	2	2	0				Bergeronnette grise		2			0	6	94	0	0	0	0	
45		1	1	1	1	100	Pipit spioncelle	2	2	0				Chardonneret elegant		2			1	5	94	0	0	0	0	
46		1	1	1	1	100	Alouette des champs	0	0	1				Martinet noir		2			1	5	94	0	0	0	0	
47		1	1	1	1	90	Pipit spioncelle	2	2	0				Mesange bonsai					3	6	91	0	0	0	0	
48		1	1	1	1	90	Alouette des champs	2	2	1				Martinet alpin					3	6	91	0	0	0	0	
49		1	1	1	1	90	Pipit spioncelle	2	2	0				Tetras lyre					80	16	4	0	0	0	0	
50		1	1	1	1	90	Alouette des champs	1	1	0				Mesange charbonniere					80	16	4	0	0	0	0	
51		1	1	1	1	95	Pipit spioncelle	2	2	0									75	20	5	0	0	0	0	
52		1	1	1	1	95	Alouette des champs	0	0	1									75	20	5	0	0	0	0	
53		1	1	1	1	95	Accenteur alpin	0	0	1									75	20	5	0	0	0	0	
54		1	1	1	1	95	Traquet motteux	1	1	0									75	20	5	0	0	0	0	
55		1	1	1	1	90	Alouette des champs	2	2	0									5	20	75	0	0	0	0	
56		1	1	1	1	90	Pipit spioncelle	2	2	0									5	20	75	0	0	0	0	
57		1	1	1	1	90	Niverolle alpine	0	3	0									5	20	75	0	0	0	0	
58		1	1	1	1	90	Accenteur alpin	1	0	0									5	20	75	0	0	0	0	
59		1	1	1	1	95	Alouette des champs	2	2	0						2			1	7	92	0	0	0	0	
60		1	1	1	1	95	Pipit spioncelle	1	2	0						2			1	7	92	0	0	0	0	
61		1	1	1	1	100	Pipit spioncelle	2	2	0						2			5	30	65	0	0	0	0	
62		1	1	1	1	100	Crave a bec rouge	0	0	2						2			5	30	65	0	0	0	0	
63																										
64								28	31	11																

Le grand nombre de données (ceci n'est pas le fichier contenant le plus de données), et le fait que ces données sont mélangés font que les données ne sont pas très évidentes à trier.

Ces fichiers de données ne sont pas très solides. Les données figurant dans ces fichiers ont été d'abord notés par les agents sur le site, puis rentrés dans le document plus tard lorsqu'ils sont revenus à la base. Certaines données ont pu ne pas être relevés, comme beaucoup de données décrivant les habitats. Ces données inconnues font que les données sont moins riches, mais les visites sans ceci sont pourtant intégrables dans la base de données. Certaines données manquantes sont plus problématiques. Il y avait une visite où l'heure de la visite n'a pas été relevée. Sans heure, ces visites sont impossibles à rentrer. Nous avons décidé de prendre l'heure pile au milieu de celle de début de la visite précédente et de l'heure de début de visite sur le site suivant. Plus encore, les fichiers ods était basées sur deux modèles différents, avec des noms de colonne différents selon le modèle. Il a donc fallu faire un choix entre les deux afin d'avoir un standard pour les noms de colonne.

Pour améliorer la solidité des données, il a fallu modifier quelques fichiers. Pour les fichiers où les éléments descriptifs des sites figurent sur une autre feuille que les données d'observation, nous avons rigoureusement transposé ces éléments sur la première feuille afin de ne pas les perdre. Nous avons également rempli les lignes vides contenant l'heure et les conditions des visites, afin de bien pouvoir associer la bonne observation avec la bonne visite. Ainsi, toutes les lignes sont complètes.

Pour intégrer les données dans la base, il a fallu choisir une stratégie. Nous aurions pu le faire directement en SQL. Cependant, nous avons décidé de le faire avec des scripts Python. En effet, si nous reformattons les fichiers en .csv, et que nous utilisons la librairie CSV de Python, et notamment sa classe DictWriter, nous pouvions transformer les lignes des fichiers en dictionnaire Python, et ainsi, nous avons une méthode de traitement qui ressemble à celle utilisée dans ODK2GN.

La première chose à traiter est tout ce qui est relatif au site. Chaque visite doit être associé à un site pour pouvoir être intégré dans la base, avec comme clé étrangère l'identifiant clé primaire des sites. Dans les fichiers, nous avons un nom de groupe de sites, et un numéro de site, et éventuellement un code de site. Si le code du site est présent, il est unique, et nous pouvons directement faire une requête dessus. Sinon, il faut concaténer le nom du groupe de sites avec le numéro du site pour obtenir son nom dans la base de données. Nous pouvons faire une requête dans la base à partir de cette chaîne. Aucun nouveau site ne sera créé car il a été décidé par les personnes en charge du protocole d'aller seulement sur des sites bien définis. La requête sur la base avec l'ORM SQLAlchemy devrait renvoyer la bonne clé.

Pour traiter les visites, nous faisons d'abord une requête dans la base de données pour récupérer certaines données qui seront les mêmes pour chaque ligne du fichier. Nous formattons ensuite la date pour qu'elle soit intégrée dans la base de manière convenable. Toutes les données descriptives du site et des conditions de la visite sont insérées dans une colonne JSON. Cette colonne existe pour rapporter toutes les informations spécifiques à chaque protocole de type Monitoring. Nous traitons ensuite les observateurs en faisant une requête sur le nom de famille de l'agent (et du prénom s'il est fourni), nous récupérons les bons éléments depuis la base, et nous les associons à la visite. La visite est ainsi prête à être intégrée dans la base.

Dans le traitement d'une ligne de fichier, nous vérifions d'abord si la visite existe déjà dans la base. Une visite préexistante serait une visite sur le même site le même jour avec les mêmes observateurs. Dans le cas où elle existe déjà, la visite retournée par la requête est celle qui sera associé à l'observation. Sinon, et seulement dans le cas où la visite en question n'existe pas déjà dans la base de données, elle est créée. L'observation est créée ensuite. Pour avoir le bon code d'espèce, nous faisons la requête sur son nom.

Pour pouvoir tester convenablement le code, nous pouvons faire un dump de la base de production afin d'avoir les bons sites, et de pouvoir vérifier le bon fonctionnement du code. De cette manière, les données en production ne seront pas modifiées jusqu'à ce qu'on sache qu'il n'y aura pas de problèmes.

C'est ici qu'est le plus visible l'utilité du travail effectué pendant le stage. Rentrer les données même avec un script pour nous faciliter la vie est un processus très long et très sensible à l'erreur. Donner aux agents un outil qui permet aux agents de limiter les erreurs et d'envoyer automatiquement les données dans la base est beaucoup plus puissant.

Roselin cramoisi © P. Saulay - PNE

3. Bilan du stage

3.1. ODK2GN : d'une librairie à un module GeoNature à part entière

La plus grande partie du stage étant consacré à travailler sur la librairie ODK2GN, nous pourrions voir exactement ce qui a changé au cours du stage avec cette librairie. Tout d'abord, ce n'est plus seulement une librairie, mais un module à part entière dans GeoNature, ce qui signifie que les commandes ont changé. Les commandes se lancent désormais de cette manière :

Ensuite, nous avons deux nouveaux formulaires prêts à être utilisés par les agents du parc, ou par ceux d'autres organisations (le Conservatoire Botanique National Alpin en particulier est concerné par le protocole de flore-prioritaire). Il n'y en avait qu'un de prêt précédemment, celui pour les oiseaux de montagne.

Nous avons également un outil beaucoup plus riche en fonctionnalités. Le fait que cette librairie soit initialement développée par le Parc National des Écrins et celui des Cévennes fait qu'il était limité en utilités à ceux spécifiques à ces deux organisations, en particulier avec l'impossibilité de créer des sites avec un mobile. Le fait que ceci soit possible rend l'application plus polyvalent. La Ligue de Protection des Oiseaux peut ainsi l'utiliser pour certains de leurs protocoles.

Avant le début de ce stage, le seul moyen de tester le code était d'utiliser ODK pour créer une soumission, afficher son contenu à l'écran pour voir si les bonnes valeurs étaient avec les bonnes clés, qu'aucune exception n'était levée, et que la donnée a été transférée à la base de données convenablement. Faire tout ceci est long et n'est pas nécessairement facile. En ajoutant des tests, le code est plus maintenable, et il est plus facile de voir ce qui est à modifier dans le cas d'une mise à jour de GeoNature ou de l'implémentation d'une nouvelle fonctionnalité.

Enfin, le fait que les fonctions de mise à jour des fichiers csv associés aux formulaires et celle de la synchronisation des données d'ODK puissent être lancés de manière automatique améliore encore plus la chaîne de travail de manière que les données sont mises en base rapidement, et que le risque de ne pas avoir la bonne donnée dans un protocole peut être limité.

3.2. Difficultés pendant le stage et tâches non aboutis

Pendant le stage, certaines tâches que nous avons initialement voulu faire n'ont pas pu être effectués. Nous avions d'abord voulu tester les formulaires dans le parc pour voir s'il collent bien aux attentes et aux demandes des gardes-moniteurs et aux autres personnes sensibles de les utiliser. Cependant, pour des raisons de temps, et des moments où certaines personnes étaient en vacances, ceci n'a pas pu se passer.

Dans le cadre de l'implémentation de la création des sites avec ODK, nous avons voulu tester une nouvelle fonctionnalité de ce software qui aurait peut-être été intéressant pour notre usage. Cette fonctionnalité nécessite une version d'ODK Central plus récente que celle sur le serveur de test, et le serveur de test est partagé par le Parc National des Écrins et celui des Cévennes, qui le gère. Nous avons essayé de l'installer en local, mais sans réussite.

Enfin, en termes de difficultés, le fait que j'ai été confronté à autant de nouveaux outils, de nouvelles méthodes de coder, et dans un langage avec lequel je n'étais pas très à l'aise m'ont posé quelques difficultés. De plus, pour certaines librairies, ou outils, la documentation pouvait être difficile à comprendre, et il n'était pas toujours très facile de voir le lien entre ce qui y figurait et ce qui était dans mon code. Nous avons également le problème que la technologie venue de l'extérieur comme ODK peut avoir des problèmes que nous ne pouvons pas corriger.

Pendant le stage, j'ai pu me refamiliariser avec Python, voir un nouveau framework, plusieurs nouvelles méthodes de créer des vues, l'utilisation des ORM, les tâches de fond et les queues, et les tests unitaires pour ne citer que quelques connaissances. D'autres nouvelles connaissances sont plus d'ordre culturels, tels quelques petites connaissances sur la taxonomie, mais également sur le monde de la géomatique.

Circumpolaire du plateau des Rouies : panorama pointe du vallon des Étages, cime de Clot Châtel, barre des Écrins © P. Saulay - PNE

Annexes et webographie.

Lien vers le projet git : https://github.com/Xav18/odk2gn/tree/to1.0

Lien vers l'article contenant le schéma de Celery-Beat : https://scripting4ever.wordpress.com/2020/07/25/scheduling-jobs-in-python-django-in-windows-based-environment

Documentation Celery-Beat : https://django-celery-beat.readthedocs.io/en/latest/

Git submodules, prise en main : https://git-scm.com/book/en/v2/Git-Tools-Submodules

Documentation Flask-admin : https://flask-admin.readthedocs.io/en/latest/

Documentation ODK : https://docs.getodk.org/

Documentation GeoNature : https://docs.geonature.fr/

Les photos figurant dans ce rapport ont été prises par Pascal Saulay, photographe du Parc National des Écrins, qui est malheureusement décédé pendant le stage.

Lien vers les articles où figurent ces photos :
https://www.ecrins-parcnational.fr/breve/roselin-gondouins
https://www.ecrins-parcnational.fr/actualite/deces-pascal-saulay-agent-photographe-passionne-parti-tot