Rapport de stage

Refonte de la base de données et de l'application web du protocole suivi de la flore du réseau Flore Sentinelle

Khanh-Chau NGUYEN

Stagiaire au **Conservatoire Botanique National Alpin** et au **Parc National des Ecrins**.

Master 2 Double Compétence : Informatique et Sciences Sociales

Année 2017-2018

Table des matières

Remerciement

Je tiens tout d'abord à remercier Camille Monchicourt – mon tuteur professionnel – et particulièrement à Théo Lechemia – mon encadrant technique – pour les conseils, le suivi, la disponibilité et le temps qu'ils m'ont consacré. Merci de votre gentillesse, de votre patience infinie, de vos encouragements et de votre confiance pour qu'une débutante en informatique comme moi puisse aller de l'avant, progresser chaque jour et donner le meilleur de moi-même.

Je remercie également l'ensemble des collègues du Parc National des Ecrins (PNE), pour votre accueil chaleureux et pour une ambiance de travail sympathique. Merci surtout à l'équipe des stagiaires, qui m'ont aidé à me rendre au parc tous les jours, et avec qui j'ai passé des supers moments pendant et en dehors du temps de travail.

Merci à l'équipe du CBNA pour les retours constructifs aidant le projet à avancer et bien se dérouler.

Enfin, merci à mes tuteurs universitaires Benoit Lemaire et Jean Michel Adam, à l'équipe pédagogique du Master et ainsi à toute la promotion DCISS pour les connaissances transmises, l'attention et le soutien que vous m'avez apporté. Ce fut une année incroyable et merveilleuse que je n'oublierai jamais !

I. Introduction

Dans le cadre du réseau Flore Sentinelle de conservation de la flore, plusieurs structures se sont regroupées pour développer des protocoles communs de suivis des espèces et des habitats patrimoniaux. Un outil de saisie, de stockage et de gestion de ces données, accessibles à tous, est donc indispensable.

Afin de moderniser la chaine de travail et les outils existants pour gérer les données flore (réalisés par le pôle Système d'Information du Parc National des Ecrins), le Conservatoire Botanique National Alpin et le Parc National des Ecrins s'associent pour proposer un stage de développement web et bases de données.

L'objectif principal du stage est de refondre la base de données et l'application web du protocole[1] de suivi de la flore du réseau Flore Sentinelle, piloté par le Conservatoire Botanique National Alpin. Plus concrètement, à la fin du stage, un module[2] de l'application web GeoNature pour la saisie et la gestion des données sera déployé.

Durant 5 mois de stage, avec l'équipe scientifique du Parc National des Ecrins et l'équipe conservation du Conservatoire Botanique National Alpin, j'ai pu réaliser le développement fondamental du module demandé. Actuellement, nous sommes en phase de finalisation de l'application. Une démonstration de cette application est désormais disponible en ligne à l'adresse : http://demo.geonature.fr/geonature/ (Rubrique *Suivi_Flore_Territoire*, Login : *admin*, Password : *admin*).

[1] ***Protocole*** : une description précise d'une procédure, d'un mode opératoire à suivre (à respecter) dans des travaux de terrain (prélèvements) ou de laboratoire (analyses) ou de toute autre activité de réalisation d'essais (*Source* : https://www.aquaportail.com/definition-4376-protocole.html)

[2] ***Module*** : Dans notre contexte, un module correspond à l'application web/mobile du protocole.

II. Contexte

Comme j'ai mentionné un peu plus haut, le stage que j'effectue actuellement, est associé aux deux structures différentes : le Conservatoire Botanique National Alpin et le Parc National des Ecrins.

A. Présentation de la structure d'accueil

1. Le Conservatoire Botanique National Alpin (CBNA)

Le CBNA est un organisme public agréé par le Ministère en charge de l'environnement. Il est dédié à la connaissance et à la préservation de la flore et des végétations des Alpes françaises. Créé en 1988 comme l'association de préfiguration du Conservatoire botanique de Gap-Charance, il est l'un des premiers Conservatoires botaniques nationaux.

Le territoire d'intervention du CBNA est centré sur la zone alpine française. Il couvre 7 départements à travers les régions de Rhône-Alpes et de PACA, résulte de la logique biogéographique alpine.

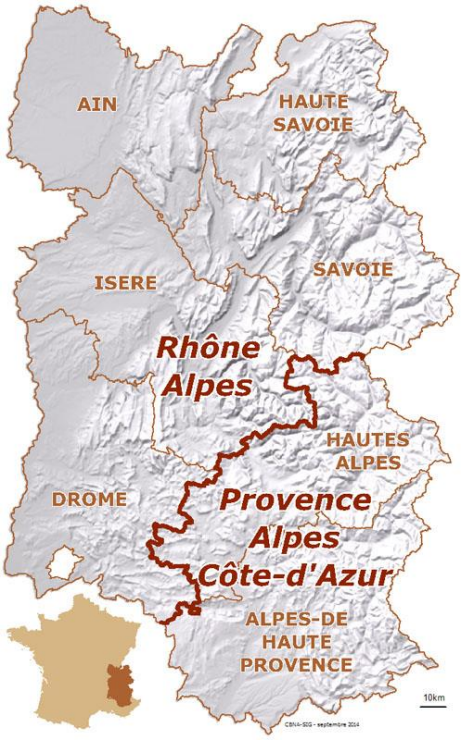


Figure 1: Le territoire d'agrément du CBNA

Depuis 20 ans, le CBNA s'est imposé comme le spécialiste de la flore et des habitats naturels dans les Alpes françaises. Il exerce quatre missions définies par le Code de l'Environnement (articles D416-1 à D416-8) :

- La connaissance :
    - Rassembler la connaissance la plus précise possible sur la répartition et les caractéristiques biologiques de la flore du territoire alpin.
    - Centraliser les données produites par le réseau de partenaires professionnels et amateurs.
    - Connaître et étudier l'état et l'évolution de la flore sauvage et des habitats.

- La conservation :
    - Identifier et conserver les éléments rares et menacés.
    - Partager les expériences et monter des projets de conservation cohérents à l'échelle alpine.

- L'expertise :
    - Expertiser à la demande de l'Etat, des collectivités, des associations et des entreprises pour leurs actions en matière de flore et d'habitats.
    - Accompagner et/ou former les acteurs de la biodiversité à l'échelle régionale, nationale et européenne.

- La sensibilisation :
    - Informer le public à la diversité végétale.
    - Faire connaître les actions du CBNA par la mise à disposition des ressources.

L'équipe pluridisciplinaire du CBNA se divise en quatre services : le service de connaissance, le service de conservation, le service de saisie-internet-géomatique et le service d'administration générale.

Mon stage est rattaché administrativement au Conservatoire Botanique National Alpin, encadré par Paul Ségura (Géomaticien – Webmestre) et Noémie Fort (Chef du service Conservation). Cependant, je suis hébergée dans les locaux du Parc National des Ecrins.

2. Le parc National des Ecrins (PNE)

Selon l'article L.331-1 du code de l'environnement, un parc national est un territoire sur lequel la conservation de la faune, de la flore, du sol, du sous-sol, de l'atmosphère, des eaux et en général d'un milieu naturel présente un intérêt spécial. Il importe de la préserver contre toute dégradation et de le soustraire à toute intervention artificielle susceptible d'en altérer l'aspect, la composition et l'évolution (INSEE).

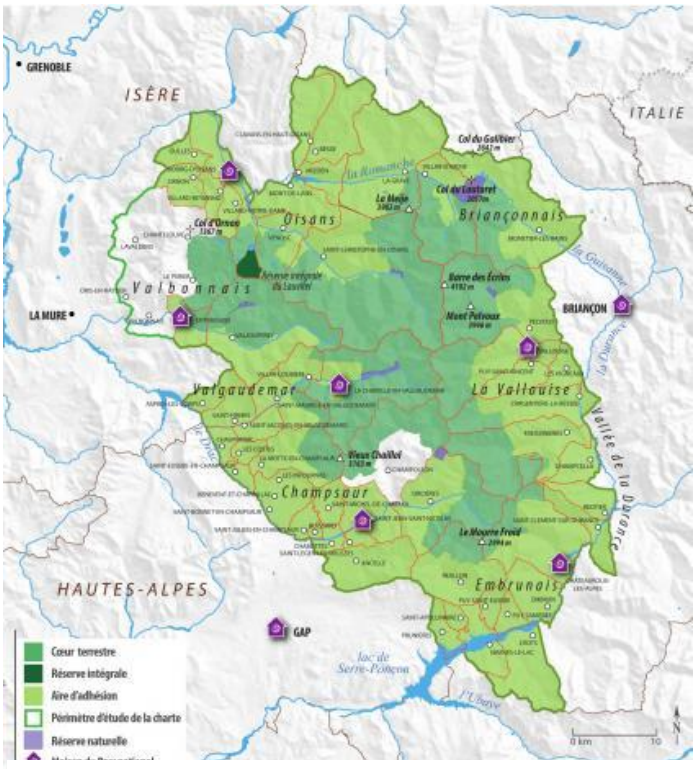


Figure 2: Carte du Parc National des Ecrins

Le PNE est le cinquième parc national, créé officiellement le 27 mars 1973. C'est un établissement public placé sous la tutelle du ministère de l'Écologie, du Développement durable et de l'Énergie, dont ses missions s'organisent autour de deux axes majeurs : la préservation de la biodiversité et la mise en place d'une politique de développement durable sur son territoire. Le PNE est situé dans les Alpes, et s'étend sur une grande partie du massif des Écrins. Il couvre 59 communes (dont 53 d'entre elles ont adhéré à la charte fixant des objectifs de protection des patrimoines), 2 départements : l'Isère (région Auvergne-Rhône-Alpes) et les Hautes-Alpes (région Provence-Alpes-Côte d'Azur) et chevauche la limite entre Alpes du Nord et Alpes du Sud en France.

L'équipe permanente du PNE comprend environ 90 agents, répartis sur 8 sites : le siège administratif est situé à Gap, puis 7 implantations territoriales dont 2 secteurs dans le département de l'Isère et 5 secteurs des Hautes Alpes. Le siège est composé de 4 services : le service général, le service d'aménagement, le service scientifique et le service de communication.

Mon stage se déroule au sein du pôle « Système d'Informations », lui-même rattaché au service scientifique. En effet, le service scientifique du PNE est divisé en deux pôles :

- Le « pôle connaissance » : ses missions consistent à la mise en place de protocole de suivi scientifique (faune, flore et mesures physiques).
- Le pôle « système d'information » : s'occupe de la géomatique et de l'informatique. (Pour en savoir plus : http://geonature.fr/documents/2018-06-Nguyen-Khanh-Chau-Rapport-mi-stage.pdf, partie *Le pôle Système Information*)

Je suis suivie techniquement par Camille Monchicourt – Responsable du Pôle Système d'Information du parc et Théo Lechémia – Développeur web et Base de données. Sous leur tutelle, la mission qui m'a été confiée est de développer un module de GeoNature pour la saisie et la gestion des données du protocole Suivi Flore Territoire du réseau Flore Sentinelle, piloté par le CBNA.

B. Présentation de l'application GeoNature

Figure 3: La page d'accueil de l'application web GeoNature

GeoNature (https://github.com/PnX-SI/GeoNature) dont la première version a été développée entre 2012 et 2015 par le pôle Système d'Information, est un ensemble d'applications web et mobiles pour saisir, gérer, synthétiser et diffuser des données faune et flore. C'est un outil open source développé et maintenu par le collectif de développeurs du réseau des Parcs Nationaux. Cette application web utilise les technologies suivantes : PostgreSQL/ PostGIS comme Système de Gestion de Base de données, Python 3 et ses dépendances, Flask (framework de Python), Apache, Angular4, Angular CLI, librairies JavaScript (Leaflet, ChartJS) et librairies CSS (Bootstrap, Material Design).

Figure 4: Les technologies utilisées

1. Le principe de fonctionnement de GeoNature

Cette application permet de déployer un système d'informations complet pour la gestion des données Faune/ Flore d'une structure, allant de

- La gestion des référentiels (taxonomiques et utilisateurs) à :
- La saisie web et mobile de données dans différents protocoles.
- La gestion de leurs métadonnées.
- L'intégration des données de partenaires.
- L'export des données selon les formats attendus par chaque partenaire.
- La synthétisation des données des différents protocoles sous forme de DEE.
- La diffusion des données sur un portail web grand public.

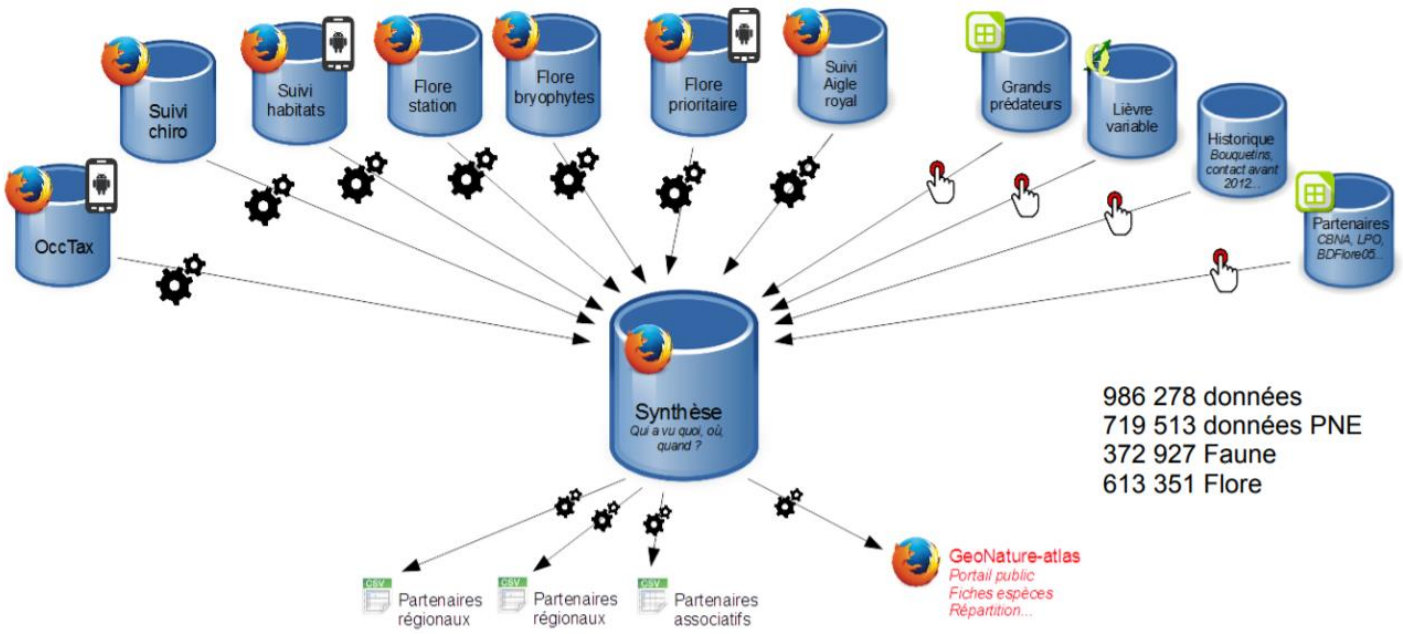


Figure 5: Une base de données de synthèse regroupant les données communes saisies dans les différents protocoles

En d'autres termes, il s'agit d'un ensemble d'application Web et mobile permettant de regrouper l'ensemble des données provenant des protocoles Faune et Flore, de saisir les données pour ces protocoles et de consulter l'ensemble de ces données dans une application de synthèse.

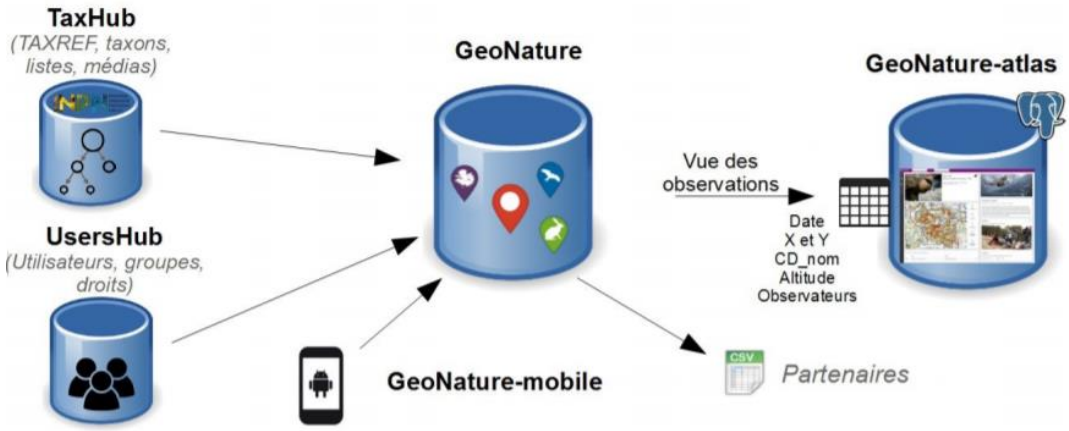
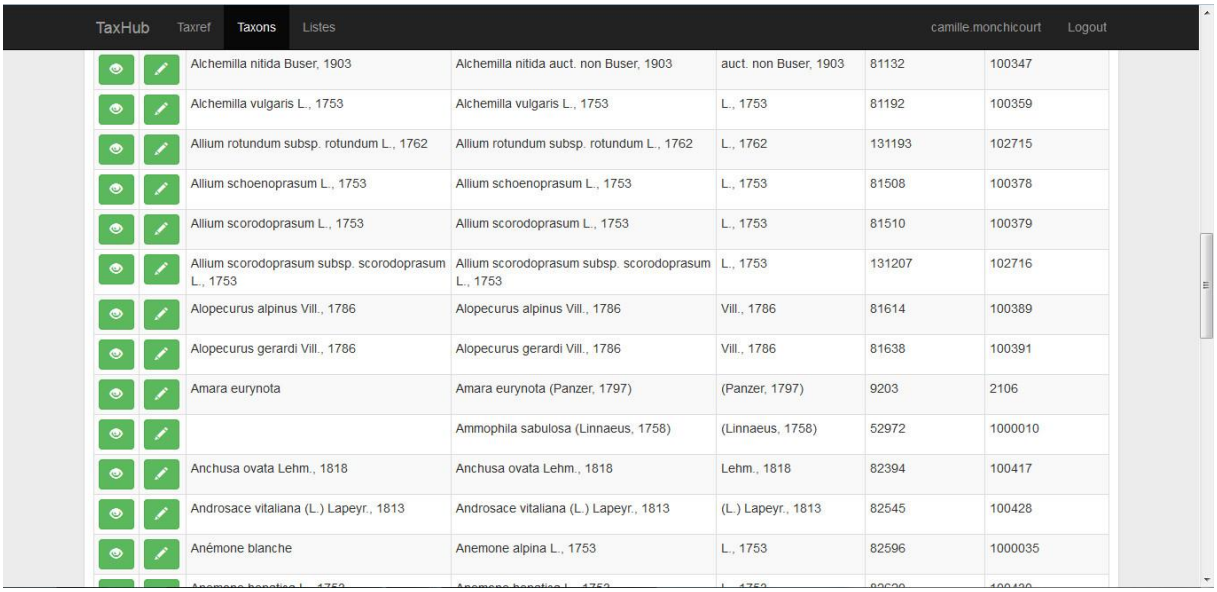


Figure 6: L'ensemble de modules de GeoNature

TaxHub et UsersHub sont des applications web en lien avec GeoNature.

2. La gestion de la taxonomie : TaxHub

Il s'agit d'une application web de gestion centralisée des taxons basée sur une base de données nationale de référence nommée TaxRef (réalisé par le Museum National d'Histoire Naturelle). Elle sert à gérer la liste des taxons présents dans un territoire donné, à y greffer des informations spécifiques (patrimonialité, marqueurs, description, etc.) et à appliquer des filtres en fonction des besoins des différents protocoles. Une fonctionnalité en lien avec l'application GeoNature-atlas permet d'associer à chaque espèce des médias (photo, vidéo, son) ou encore des articles.



Alchemilla nitida Buser, 1903	Alchemilla nitida auct. non Buser, 1903	auct. non Buser, 1903	81132	100347
Alchemilla vulgaris L., 1753	Alchemilla vulgaris L., 1753	L., 1753	81192	100359
Allium rotundum subsp. rotundum L., 1762	Allium rotundum subsp. rotundum L., 1762	L., 1762	131193	102715
Allium schoenoprasum L., 1753	Allium schoenoprasum L., 1753	L., 1753	81508	100378
Allium scorodoprasum L., 1753	Allium scorodoprasum L., 1753	L., 1753	81510	100379
Allium scorodoprasum subsp. scorodoprasum L., 1753	Allium scorodoprasum subsp. scorodoprasum L., 1753	L., 1753	131207	102716
Alopecurus alpinus Vill., 1786	Alopecurus alpinus Vill., 1786	Vill., 1786	81614	100389
Alopecurus gerardi Vill., 1786	Alopecurus gerardi Vill., 1786	Vill., 1786	81638	100391
Amara eurynota	Amara eurynota (Panzer, 1797)	(Panzer, 1797)	9203	2106
	Ammophila sabulosa (Linnaeus, 1758)	(Linnaeus, 1758)	52972	1000010
Anchusa ovata Lehm., 1818	Anchusa ovata Lehm., 1818	Lehm., 1818	82394	100417
Androsace vitaliana (L.) Lapeyr., 1813	Androsace vitaliana (L.) Lapeyr., 1813	(L.) Lapeyr., 1813	82545	100428
Anémone blanche	Anemone alpina L., 1753	L., 1753	82596	1000035

Figure 7: Exemple de l'interface de l'application TaxHub.

Chaque espèce est identifiée par un identifiant national unique : cd_nom (clé primaire), ce qui facilite l'échange des données entre les structures.

TaxHub permet d'administrer un schéma « taxonomie » dans une base de données PostgreSQL. Ce schéma comprend les tables de TaxRef et leur contenu complet.

TaxHub utilise déjà les technologies cibles de la refonte de GeoNature (Python Flask, Angular, Bootstrap). Il ne sera donc pas refondu et n'est concerné que par quelques évolutions.

3. La gestion des utilisateurs : UsersHub

C'est une application de gestion centralisée des utilisateurs. Elle regroupe l'ensemble des utilisateurs d'applications web afin de gérer de manière différenciée et centralisée les droits d'accès à ces applications, ainsi que le contenu des listes déroulantes d'observateurs. Concrètement, elle permet de gérer les utilisateurs et de les placer dans des groupes, de créer différents niveaux de droits et de les affecter aux utilisateurs (ou groupe d'utilisateur pour chacune des applications).

Figure 8: L'application UsersHub

UsersHub administre un schéma « utilisateurs » dans une base de données PostgreSQL. Celui-ci est répliqué dans les différentes base de données de l'ensemble des applications GeoNature et sera automatiquement mis à jour à chaque modification dans UsersHub.

UsersHub utilise des technologies datées (ExtJS, PHP Symfony). Cependant, il est fonctionnel est indépendant. Il ne sera donc pas refondu et n'est concerné que par quelques évolutions.

4. Exemples des applications de GeoNature

4.1 GeoNature-Mobile

L'application GeoNature-mobile permet de saisir le contact Faune et les protocoles de suivi de la flore prioritaire sur des appareils mobiles Android (https://github.com/PnEcrins/GeoNature-mobile). Elle est conçue pour un usage hors ligne. Les agents peuvent l'utiliser pour saisir les observations directement sur le terrain et donc alimenter la base de données de GeoNature.

Figure 9: L'interface de l'application GeoNature-mobile

GeoNature-mobile a été maintenue régulièrement depuis sa première version pour être compatible avec les nouvelles versions d'Android et être de plus en plus générique.

4.2 GeoNature-Atlas

L'application GeoNature-atlas permet de publier un atlas en ligne pour mettre à disposition toutes les observations présentes dans GeoNature (https://github.com/PnEcrins/GeoNature-atlas).

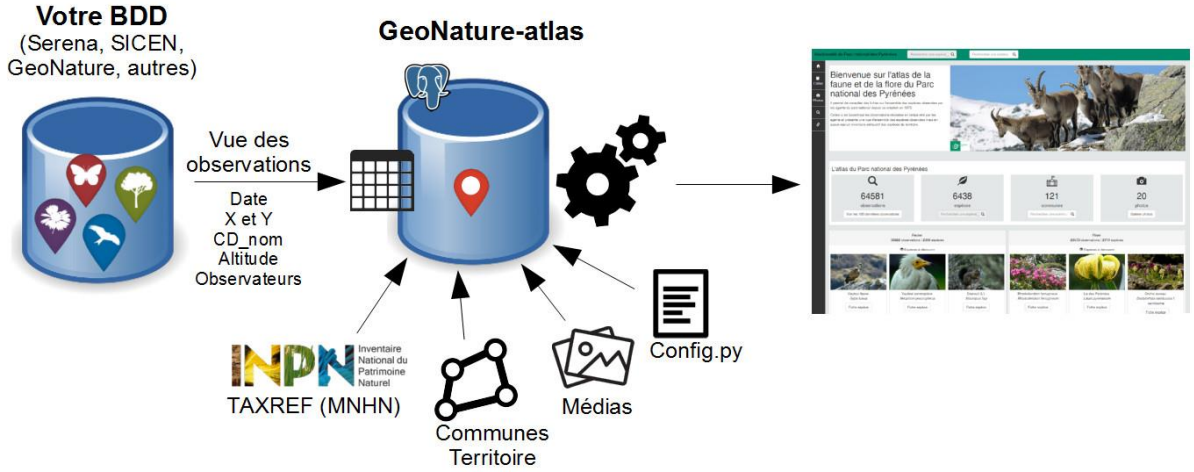


Figure 10:Principe de GeoNature-Atlas (construire une vue à partie des observations et de la table TaxRef et publier l'atlas en ligne dynamiquement)

Cette application est alimentée par les données présentes dans la synthèse de GeoNature, et peut être installée indépendamment de GeoNature, en se branchant sur n'importe quelle base de données contenant des observations Faune/ Flore. Elle est entièrement générique et paramétrable.

4.3 GeoNature-Suivi

GeoNature-Suivi est une application web développée par le Parc National des Cévennes. Elle a été développée pour le protocole Suivi Chiroptères mais a été conçue de manière générique pour pouvoir être adaptée à d'autres protocoles de suivi, sans être entièrement redéveloppé.

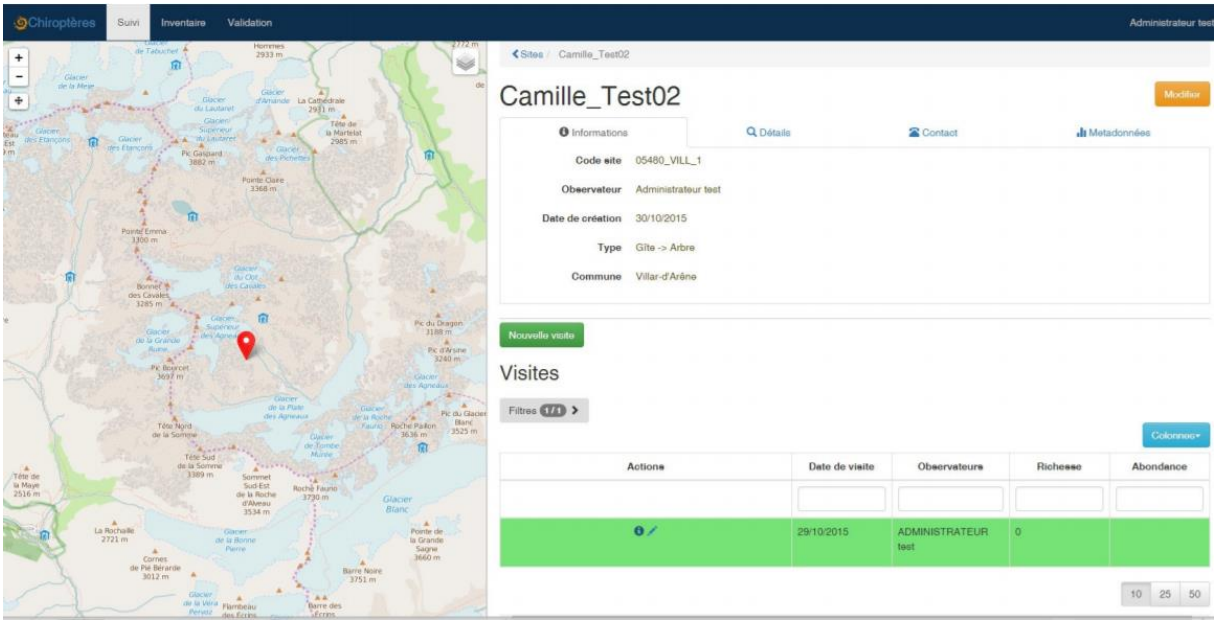


Figure 11: Exemple de l'application Suivi Chiropètes

Autrement dit, cette application s'appuie sur même le principe que tous les protocoles de suivi qui sont basés sur des sites dans lesquels des visites sont réalisées (plus ou moins régulièrement). Par conséquent, la partie Sites et Visites est générique et commune à tous les protocoles. A chaque visite, des observations de taxons sont faites avec les spécificités de chaque protocole, il faut donc les adapter en y associant des champs liés au protocole.

Le module Suivi Flore Territoire – celui que je dois développer durant mes 5 mois de stage – emprunte ce même principe.

III. Problématiques – objectifs

A. Intérêts de GeoNature version 2 (GeoNature2)

L'objectif principal du développement de GeoNature2 est de mettre en place un système d'information de faune et flore commun, partagé et modulaire pour les parcs nationaux. En effet, GeoNature v1 n'est pas assez modulaire pour permettre à chacun de l'adapter à ses besoins et à ses protocoles. L'objectif est de proposer un système modulaire pour ne pas imposer des outils ou des technologies, mais plutôt partager une organisation des données et un ensemble d'outils et de compétences dans lequel, chacun peut intégrer ses propres protocoles.

Par ailleurs, GeoNature1 devient obsolète techniquement : les technologies vieillissantes et non maintenues, la présence des défauts de compatibilité, d'ergonomie et de performances (pour en savoir plus : http://geonature.fr/documents/2018-06-Nguyen-Khanh-Chau-Rapport-mi-stage.pdf, partie *L'ancienne version de GeoNature : les inconvénients*).

En résumé, la version 2 de GeoNature est une refonte complète de l'application :

- Refonte de l'organisation des données et le socle des outils Faune et Flore pour :
    - Mieux pouvoir répondre aux questions de chaque protocole.
    - Mieux intégrer les données des partenaires, voire des citoyens.
    - Valoriser ces données publiquement.
    - Uniformiser la manière d'organisation des données et les méthodes de travail pour faciliter les échanges entre les partenaires, l'assistance mutuelle et l'interopérabilité[3].
- Refonte technologique en migrant GeoNature vers les technologies adoptées par les parcs nationaux (Python, Flask, Angular, Bootstrap).
- Refonte de l'architecture du code pour rendre GeoNature plus générique et modulaire[4].
- Refonte ergonomique pour moderniser l'application.

---

[3] ***Interopérabilité*** : ou interfonctionnement en informatique, est la capacité que possède un système informatique à fonctionner avec d'autres produits ou systèmes informatiques, existants ou futurs, sans restriction d'accès ou de mise en œuvre. (***Source*** : Wikipédia).

[4] ***Modulaire*** : se dit d'un système, matériel ou logiciel, conçu en séparant les fonctions élémentaires pour qu'elles puissent être étudiées et réalisées séparément (Source : Larousse).

B. Pourquoi le module Suivi Flore Territoire ?

Ce module vise à étudier la présence d'une espèce définie sur des territoires donnés (au niveau du territoire alpin), et par conséquent, permet de suivre son évolution dans le temps. Un territoire donné est associé à une Zone de Prospection[5] (ZP). Sur chaque ZP, un taxon[6] spécifique est prédéfini pour le suivi. Chaque ZP est découpée en mailles 25m*25m. Les agents devront ensuite visiter régulièrement chaque maille pour connaître la présence ou l'absence de ce taxon. Ces observations seront saisies en grille de case à cocher. Il est souhaité que la coloration de chaque maille change en fonction de la présence ou non de l'espèce suivie, et aussi, du statut de la maille (si elle est déjà visitée ou pas). Lors du relevé, l'observateur note aussi la date, son nom et les types de perturbation si elles existent.

Figure 12: Illustration de la réalisation du protocole Suivi Flore Territoire

[5] *Zone de prospection* : Une surface qui comprend l'aire de présence de l'espèce et l'aire d'absence de l'espèce (là où on a cherché l'espèce mais où on ne l'a pas trouvée).

[6] *Taxon* : Un taxon est une unité quelconque (genre, famille espèce, sous-espèce, etc.) des classifications hiérarchiques des êtres vivants. Généralement, le terme est employé aux rangs spécifiques (l'espèce) et sub-spécifique (la sous-espèce).

Un des plus importants intérêts du développement de Suivi Flore Territoire est la possibilité de suivre les changements/ l'évolution de l'espèce suivie dans le temps. Actuellement, même si ce protocole est commun à toutes les structures, la manière dont les données sont récoltées diverge. Le risque majeur qui se pose est que les utilisateurs peuvent se perdre dans le temps s'ils utilisent les données d'un autre utilisateur ou d'un autre organisme : les observations saisies ne pourront pas, ou difficilement être réutilisées. Par conséquent, le résultat serait biaisé et les stratégies données ne seront pas totalement adaptée au contexte, ou ne seront pas à la hauteur de l'attente.

De plus, il est également souhaité que le module soit développé de manière générique pour pouvoir être réutilisé dans d'autres protocoles liés aux habitats. C'est le cas du module Suivi Station et Suivi Individu (http://geonature.fr/documents/2018-06-Nguyen-Khanh-Chau-Rapport-mi-stage.pdf, partie *Les modules à développer*) dont leur développement sera mis en oeuvre prochainement, tout en s'appuyant sur le module Suivi Flore Territoire et en ajoutant des spécificités propres à chacun de ces protocoles. Cela rejoint l'objectif principal de la migration vers GeoNature2.

C. Objectifs du stage.

1. Les objectifs principaux

- Refondre la base de données et de l'application web du protocole en tant que module de GeoNature2.
- Modéliser le protocole de suivi flore du réseau Flore sentinelle pour traduire les évolutions à y apporter.
- Concevoir la nouvelle base de données du protocole.
- Développer le module de saisie et de gestion des données du protocole.

2. Les objectifs spécifiques

- Analyser la base de données et l'application web existantes.
- Participer à la rédaction d'un cahier des charges pour présenter les évolutions à réaliser.
- Migrer les données existantes dans la nouvelle base de données.
- Documenter et illustrer les développements réalisés.

3. Les objectifs opérationnels.

- La construction du Modèle Conceptuel des Données.
- Le test pour démontrer la faisabilité du protocole.
- Le développement fondamental du protocole Suivi Flore Territoire.
- Le maquettage et la conception d'interface graphique.
- Les tests : les tests unitaires, les tests d'intégration, les tests système et les tests d'acceptation.
- La correction des bugs et l'amélioration selon des feedbacks.
- La finalisation du projet.

IV. Déroulement du stage

Le travail de l'équipe est organisé selon les méthodes Agile. Le projet est divisé en plusieurs sous-projets avec des objectifs à court-terme. Puisqu'il est difficile de tout prévoir et tout anticiper, cette approche laisse la place aux imprévus et aux changements. De plus, elle facilite le dialogue entre les membres de l'équipe, ainsi qu'entre l'équipe et les autres acteurs (partenaires, clients, etc.). Pour chaque étape du projet, l'évolution de ses besoins est prise en compte et les ajustements sont effectués. L'équipe doit donc être capable de se remettre en cause et de chercher en permanence à évoluer.

A. La préparation du travail de développement

1. La découverte de l'environnement de travail.

Avant de commencer mon stage, mes connaissances dans le domaine de Système d'Information Géographique, ainsi que les outils et les logiciels utilisés, étaient assez limitées. A mon arrivée, l'équipe de travail a pris le temps de bien m'expliquer le fonctionnement de chaque application employée, son architecture, son utilité dans le contexte du travail mais également le fonctionnement des structures d'accueil, leur histoire et leurs objectifs.

2. L'installation de l'environnement de travail.

Avec l'aide de mes tuteurs techniques, j'ai pu apprendre comment installer l'environnement de travail, à savoir, l'application GeoNature. J'ai pu donc mieux appréhender l'architecture de GeoNature, ses composants génériques[7], la manipulation des données et également les erreurs survenues au cours de l'installation.

3. La montée en compétence technologiques.

Comprenant bien les difficultés que posait le projet pour une débutante en programmation/ développement web, l'équipe de travail du pôle Système d'Information m'avait accordé 3 semaines de montée en compétences. C'était pour moi une grande occasion de pouvoir m'imprégner dans l'environnement des langages utilisés, de découvrir les nouvelles notions et également de consolider mes connaissances de base. Ceci étant dit, ce n'était que des éléments les plus basiques et j'ai énormément appris des choses au cours du développement du module Suivi Flore Territoire.

4. La réunion de définition des objectifs.

Cette réunion a eu lieu de 4 juin 2018 et avait pour but de mettre au clair la pertinence des modules à développer, et le choix de priorisation des projets.

Nous avons finalement décidé de mettre l'accent sur le module Suivi Flore Territoire.

B. Le développement

1. La réalisation du Modèle Conceptuel des Données

Un Modèle Conceptuel des Données (MCD) est la représentation la plus abstraite des données d'un système d'information. Les données sont représentées sous forme d'entités et d'associations entre entité (*Source* : http://www.smartmodel.ch/home/questce/mcd).

Une entité est un ensemble homogène d'éléments qui correspondent au même « objet » à informatiser. Cette entité a un nom unique.

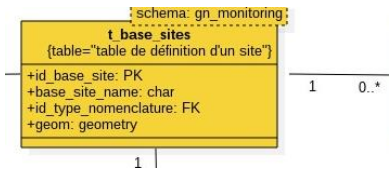


Figure 13: Une entité du MCD

[7] ***Composants génériques*** : intégrés dans le module GN2CommonModule et réutilisables dans n'importe quel module.

Une association est un ensemble de liens de même nature entre élément d'ensembles.

Après la réunion de l'équipe, je me suis mise à la construction du MCD pour le module Suivi Flore Territoire. Il se base principalement sur ce qui a été défini dans le protocole Suivi Habitat Territoire (http://geonature.fr/documents/2018-06-Nguyen-Khanh-Chau-Rapport-mi-stage.pdf, partie *Les modules à développer*), qui est lui-même s'appuie sur le schéma **gn_monitoring** (SI/GeoNature/blob/develop/data/core/monitoring.sql). Il s'agit d'un schéma permettant de gérer de manière centralisée les visites et les sites, de les étendre dans le schéma des protocoles de type Suivi et d'y associer des observations.

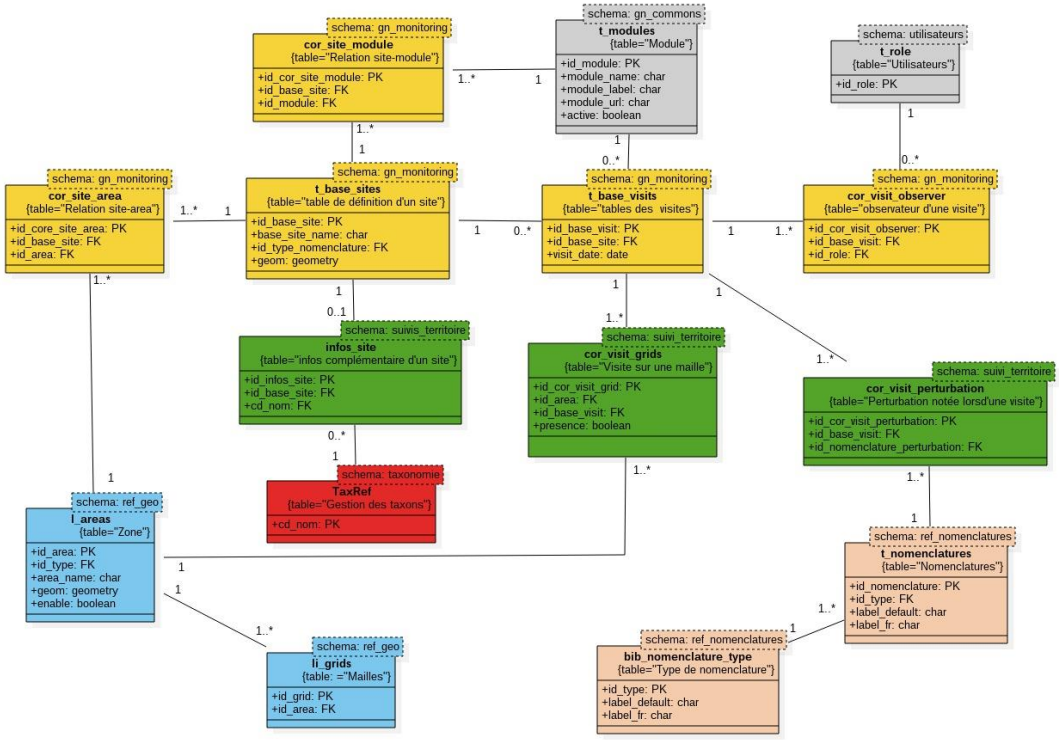


Figure 14: MCD du module Suivi Flore Territoire

- Les tables en jaune sont des tables génériques de GeoNature 2, elles se retrouvent dans le schéma gn_monitoring.
- Les tables en vert sont des tables « cœur » de notre module Suivi Flore Territoire. Ce sont des extensions des tables génériques.
- Les tables restantes sont des tables communes et sont utilisées dans plusieurs modules de GeoNature.

2. La preuve de concept ou la démonstration de faisabilité du module Suivi Flore Territoire

Une preuve de concept (POC : Proof of Concept) ou démonstration de faisabilité, est une réalisation expérimentale concrète ou préliminaire, courte ou incomplète, illustrant une certaine méthode ou idée pour en démontrer la faisabilité (Wikipédia). L'objectif est de valider la conformité du projet avec les objectifs fixés, et plus globalement, avec le cahier des chargés élaborés au début du projet. Une POC permet de tirer les première conclusions d'un projet et de prendre les bonnes décisions pour en assurer le succès.

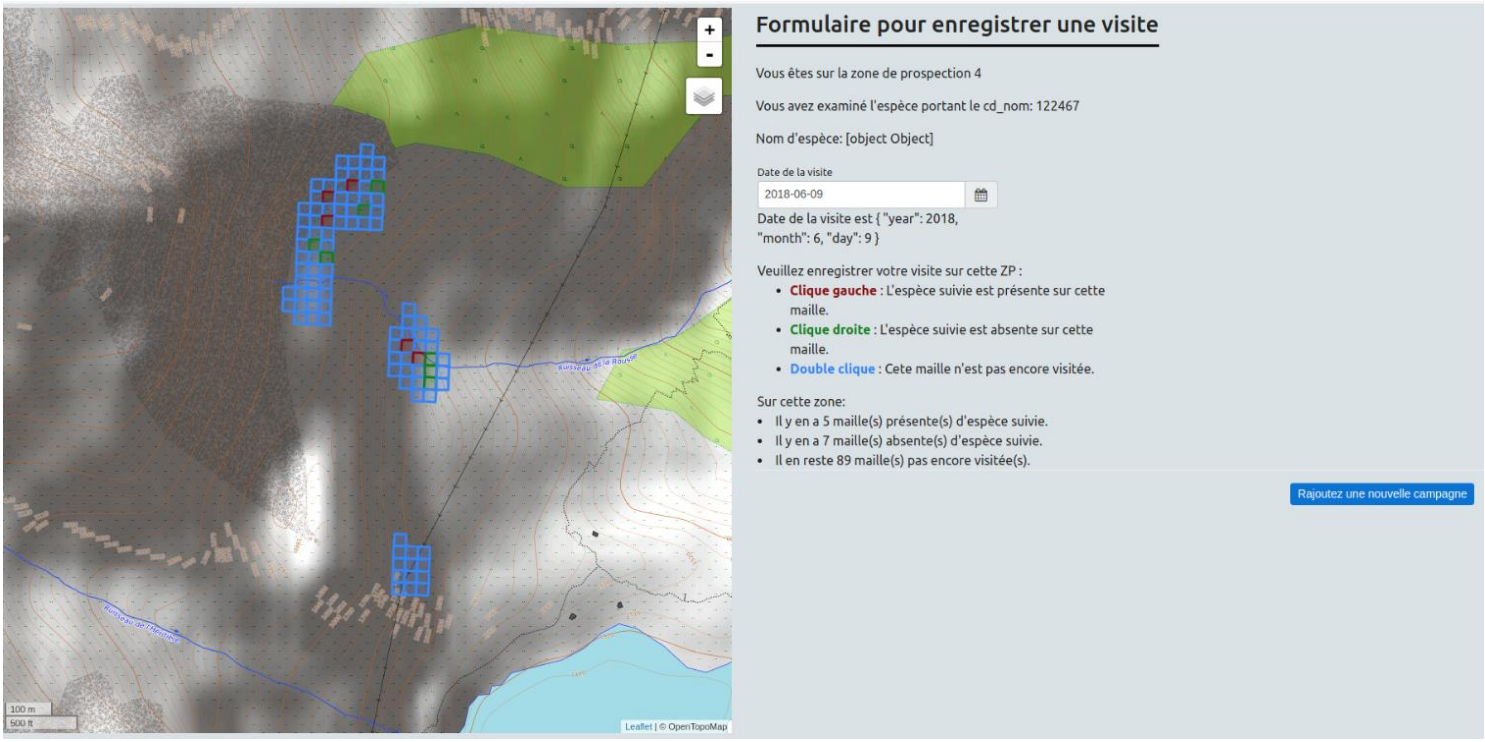


Figure 15: Démonstration de faisabilité du formulaire des visites du module Suivi Flore Territoire

J'ai réalisé une POC pour le formulaire des visites du module Suivi Flore Territoire.

Bien que ce formulaire soit incomplet et nécessite de nombreuses modifications, nous avons pu en tirer profit : l'ajout des nouveaux champs sur le formulaire des visites, l'organisation générale des composants du module, la confirmation du MCD et la création des tables propres à ce module, les premières idées de la conception de l'interface graphique, etc.

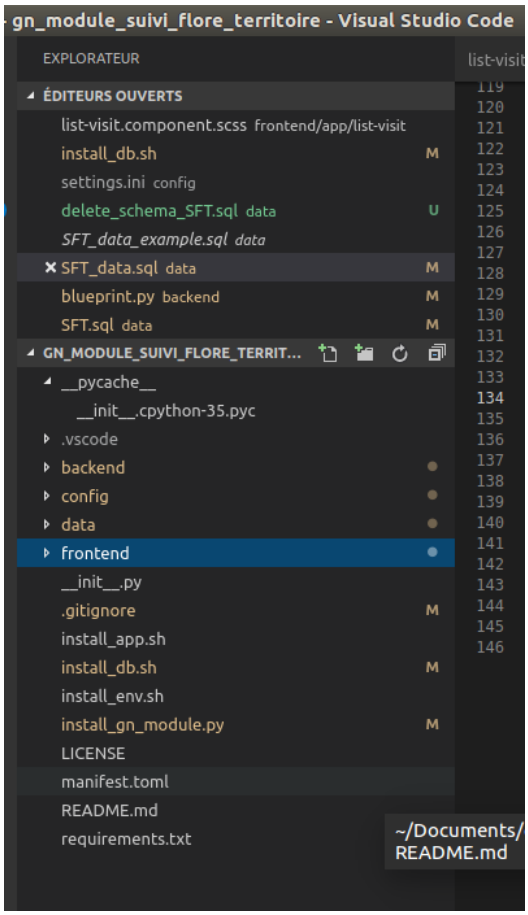
Grâce à la POC réalisée, les conclusions nécessaires sont données pour que la mise en place du développement du module Suivi Flore Territoire puisse débuter dans les meilleures conditions.

3. La programmation

Vous trouverez ci-dessous le lien vers le dépôt GitHub de notre module Suivi Flore Territoire :

https://github.com/PnX-SI/gn_module_suivi_flore_territoire

3.1 Structure du code du module Suivi Flore Territoire



L'illustration nous montre comment est organisé le code de notre application module Suivi Flore Territoire.

D'une manière générale, le corps de notre module se divise en 4 grandes parties :

- La partie Config : contient les fichiers pour la configuration globale du module.
- La partie Data : contient les fichiers permettant de créer la base de données propre à ce module.
- La partie Back-end : interagit avec la base de données, écrit une Web API Rest[8] indiquant ce qu'il faut afficher/ ajouter/ modifier dans la base de données. Ecrit en Python et Flask.
- La partie Front-end : permet d'afficher le contenu/ l'interface graphique de notre module. Il s'agit d'une combinaison de HTML, SCSS et TypeScript. Utilise Angular4.

Figure 16: L'organisation du code du module Suivi Flore Territoire

[8] ***Web API Rest*** : Une API compatible REST, ou « RESTful », est une interface de programmation d'application qui fait appel à des requêtes HTTP pour obtenir (GET), placer (PUT), publier (POST) et supprimer (DELETE) des données (***Source*** : https://www.lemagit.fr/definition/API-RESTful)

L'interaction entre le Back-end et la base de données sont des interactions entre Flask et PostgreSQL-PostGis[9] via l'ORM[10] SQLAlchemy. L'interaction entre le Back-end et le Front-end est l'interaction entre Flask et Angular via API Rest. Enfin, le framework Bootstrap gère le CSS.

Par ailleurs, il existe d'autres fichiers : install_gn_module.py (l'installation du module), README.md (le document de présentation générale du module).

3.2 Contenu du module

3.2.1 Le répertoire Data : SQL Base de données

L'ensemble des données des protocoles de GeoNature sont centralisées dans une base de données PostgreSQL (avec l'extension PostGIS qui active la manipulation d'informations géographiques sous forme de géométrie). Notre module Suivi Flore Territoire n'est pas une exception. Ce choix du système de base de donnée est porté sur sa capacité de gestion de gros volumes, la facilité de d'intégration des données et des fonctionnalités destinées aux développeurs d'applications.

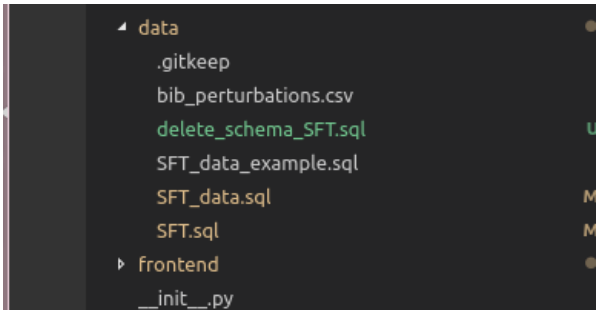


Figure 17: Le contenu du répertoire Data

Ce répertoire « data » contient 2 fichiers principaux :

- SFT.sql : fichier des scripts SQL pour créer le schéma pr_monitoring_flora_territory (schéma du module Suivi Flore Territoire), ainsi que les tables (celles d'extension du schéma gn_monitoring), ses contraintes et une vue dédiée à l'exportation des données. Ce fichier est indispensable pour le fonctionnement du module.

---

[9] ***PostGis*** : une extension (plugin) du SGBD PostgreSQL, qui active la manipulation d'informations géographiques (spatiales) sous forme de géométries (points, lignes, polygones). (***Source*** : Wikipédia).

[10] ORM : **Object-Relational Mapping.** Il s'agit d'une technique de programmation informatique qui permet de simplifier l'accès à une base de données en proposant à l'informaticien des « objets » plutôt que d'accéder directement à des données relationnelles

- SFT_data.sql : fichiers des scripts SQL pour insérer les données dans les tables correspondantes (ex : les données spatiales pour afficher une zone de prospection[11] (ZP) et ses mailles, le nom de commune, etc.). L'utilisateur peut utiliser les données pré-fournies ou insérer ses propres données selon notre modèle de script.

A la racine du module, nous trouvons le fichier install_db.sh. L'exécution de ce fichier permet la création de notre base de données pour Suivi Flore Territoire. Il s'agit d'un script bash Linux qui :

- Fait appel au fichier settings.ini se trouvant dans le répertoire config .
- Utilise la commande *shp2pgsql* de PostGIS afin de transformer les fichiers shapefiles[12] en table dans la base de données.
- Fait appel au fichier SFT.sql et SFT_data.sql, qui sont copiés dans le répertoire tmp, et puis lancent la commande *psql* permettant de « saisir des requêtes de façon interactive, de les exécuter dans PostgreSQL et de voir les résultats de ces requêtes » (https://docs.postgresql.fr/8.2/apppsql.html).
- Enregistre le résultat dans le fichier install_sft.log (la sortie normale ET la sortie d'erreurs).

3.2.2 Le répertoire Config

Ce répertoire comprend les fichiers dédiés à la configuration de l'installation de la base de donnée, et la configuration front-end propre du module Suivi Flore Territoire.

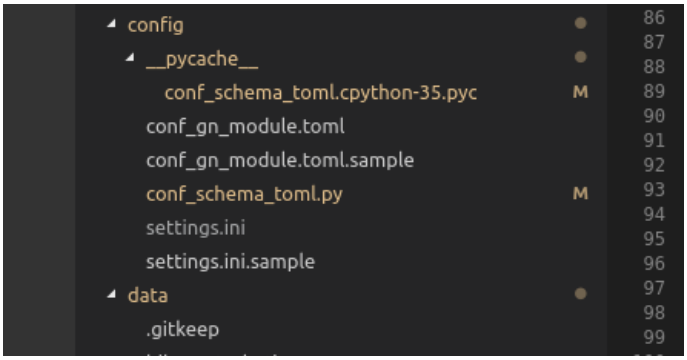


Figure 18: Le contenu du répertoire Config

11

12 Shapefiles : format de fichier pour les systèmes d'informations géographiques (Wikipédia).

a. La configuration de l'installation de la BD : settings.ini

Tous les paramètres nécessaires à l'installation de la BD sont stockés ici.

- db_host : l'adresse du serveur de la base de données connectée.
- db_name : le nom de la base de données.
- user_pg : le nom d'utilisateur.
- user_pg_pass : son mot de passe.

Ce fichier est mis dans .gitignore. Vu qu'il contient des données personnelles, nous ne voulons pas le pousser sur le dépôt git. Néanmoins, nous avons créé un fichier settings.ini.sample pour fournir un exemple aux utilisateurs.

b. La configuration front-end du module

Comme je l'ai mentionné un peu plus haut, un des gros intérêts de GeoNature2 est de rendre l'application modulaire et facilement partagé entre les partenaires. Il ne faut pas que les options/ les variables soient écrites « en dur », car elles se différencient d'une base de donnée à l'autre (exemple : il ne faut pas mettre l'id_application_suivi_flore_territoire=18 dans les codes, car dans une autre base de donnée, ce numéro 18 peut être occupé par une autre application. Cela va créer des erreurs fatales !).

Pour résoudre ce problème, un maximum de variables sont passées en paramètre pour que l'application soit configurable. L'utilisateur n'a qu'à rentrer dans le fichier conf_gn_module.toml tout ce qui est nécessaires pour la configuration du module (et pas la base de données !) :

- L'identifiant de l'application.
- Le format d'export des fichiers.
- Les messages d'erreurs.
- Etc.

Ce fichier définit les paramètres disponibles et les valeurs par défauts de certains paramètres du module. Il est écrit en python, en utilisant la librairie marshmallow[13].

---

[13] *Marshmallow* : Il s'agit d'une bibliothèque d'ORM/ d'un framework agnostic (Qualifie un code dont l'architecture permet de l'exécuter dans plusieurs frameworks possibles) permettant de convertir des types de données complexes, tels que des objets, vers et à partir de données Pythons. Pour en savoir plus : https://marshmallow.readthedocs.io/en/3.0/

De même, conf_gn_module.toml est également mis dans .gitignore. Un fichier d'exemple est mis à disposition des utilisateurs.

Après chaque modification du fichier conf_gn_module.toml, il faut se placer dans le back-end de GeoNature et lancer la commande ci-dessus dans virtualenv de python[14]:

```
khanh-chau@SI-DEV-STAGE-2:~/Documents/GeoNature$ cd backend/
khanh-chau@SI-DEV-STAGE-2:~/Documents/GeoNature/backend$ source venv/bin/activate
(venv) khanh-chau@SI-DEV-STAGE-2:~/Documents/GeoNature/backend$ geonature update module configuration suivi flore territoire
[sudo] Mot de passe de khanh-chau :
```

Quand cette commande est exécutée, le fichier module.config.ts (qui se trouve dans le dossier app du répertoire front-end) sera automatiquement synchronisé avec le fichier conf_gn_module.toml. Il permet donc la mise à jour de la configuration front-end du module.

3.2.3 Le répertoire Back-end

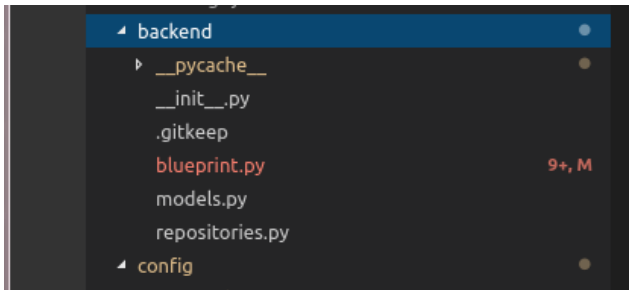


Figure 19: Le contenu du répertoire back-end

Le côté back-end était pour moi celui le plus redoutable tout au début du développement de l'application, car son langage de programmation est Python – celui que je connaissais très peu avant de commencer mon stage. De plus, je n'avais pas forcément saisi le principe du développement back-end au début. Finalement, avec les semaines de montée en compétence et grâce à l'aide de mes collègues, j'ai donc bien compris son fonctionnement, son architecture, sa structuration, et j'ai pu livrer un travail plutôt approuvable concernant cette partie back-end.

L'équipe de développement a choisi Python pour le développement back-end grâce à sa souplesse et sa simplicité à apprendre et à utiliser. Il est également le langage préférentiel de la communauté open-source SIG. Parallèlement, nous utilisons le framework Flask pour créer des routes qui définiront une Web API Rest.

---

[14] ***virtualenv de python*** : un outil pour créer des environnements virtuels afin de garder les dépendances requises par différents projets dans des emplacements séparés.

a. Fichier repositories.py

La fonction check_user_cruved_visit est déclarée dans ce fichier. Elle vérifie si l'utilisateur a le/les droit(s) sur une visite, en fonction de son CRUVED.

Le système CRUVED est mis en place pour la gestion des droits des utilisateurs de manière avancée dans GeoNature2. Il correspond à : **C**reate (Créer), **R**ead (Lire), **U**pdate (Mettre à jour), **V**alidate (Valider), **E**xport (Exporter) et **D**elete (Supprimer).

```python
from flask import Blueprint, request, session, current_app

from geonature.utils.errors import InsufficientRightsError
from pypnusershub.db.tools import get_or_fetch_user_cruved


def check_user_cruved_visit(user, visit, cruved_level):
    """
    Check if user have right on a visit object, related to his cruved
    if not, raise 403 error
    if allowed return void
    """

    is_allowed = False
    if cruved_level == '1':

        for role in visit.observers:
            if role.id_role == user.id_role:
                print('même id ')
                is_allowed = True
                break
            elif visit.id_digitiser == user.id_role:
                is_allowed = True
                break
        if not is_allowed:
            raise InsufficientRightsError(
                ('User "{}" cannot update visit number {} ')
                .format(user.id_role, visit.id_base_visit),
                403
            )
```

Figure 20: Une partie du code du fichier repositories.py

Notre fonction prend 3 paramètres obligatoires :

- Le rôle de l'utilisateur (un objet).
- La visite (un objet).
- La portée de CRUVED (un entier), donc
    - 1 : l'utilisateur n'a le droit de CRUVED que sur ses visites.
    - 2 : l'utilisateur a le droit de CRUVED sur ses visites et celle de son organisme.
    - 3 : l'utilisateur a le droit de CRUVED sur toutes les visites.

b. Fichier models.py

Ce fichier contient des classes nécessaires au fonctionnement du module. Elles sont obtenues grâce à SQLAlchemy. Ces classes sont en fait des tables SQL du schéma pr_monitoring_flora_territory, mais transformées en objets facilement manipulables via ses attributs. Puis, elles seront appelées par des routes se trouvant dans le fichier blueprint.py pour afficher/ rajouter/ exporter des données. L'ORM SQLAlchemy fait le lien entre le monde de la base de données et le monde de programmation objet.

```
        DB.Integer,
        ForeignKey(TNomenclatures.id_nomenclature),
        primary_key=True
    )
)


@serializable
# @geoserializable
class CorVisitGrid(DB.Model):
    '''
    Corespondance entre une maille et une visite
    '''
    __tablename__ = 'cor_visit_grid'
    __table_args__ = {'schema': 'pr_monitoring_flora_territory'}

    id_area = DB.Column(
        DB.Integer,
        ForeignKey(LAreas.id_area),
        primary_key=True
    )
    id_base_visit = DB.Column(
        DB.Integer,
        ForeignKey(TBaseVisits.id_base_visit),
        primary_key=True
    )
    presence = DB.Column(DB.Boolean)
    uuid_base_visit = DB.Column(UUID(as_uuid=True))


@serializable
@shapeserializable
class TVisiteSFT(TBaseVisits):
    '''
    Visite sur une ZP
    et corespondance avec ses mailles
    '''
    __tablename__ = 't_base_visits'
    __table_args__ = {
        'schema': 'gn_monitoring',
        'extend_existing': True
    }

    cor_visit_grid = DB.relationship(
        'CorVisitGrid',
        primaryjoin=(
            CorVisitGrid.id_base_visit == TBaseVisits.id_base_visit
        ),
        foreign_keys=[
            CorVisitGrid.id_base_visit,
        ]
    )
    cor_visit_perturbation = DB.relationship(
        TNomenclatures,
        secondary=corVisitPerturbation,
        primaryjoin=(
            corVisitPerturbation.c.id_base_visit == TBaseVisits.id_base_visit
        ),
```

Figure 21: Une partie du fichier models.py

Vous pouvez constater que certaines classes ont les décorateurs @serializable et @geoserializable. Ce sont les décorateurs développés par l'équipe de développement de GeoNature. Le @serializable ajoute la méthode as_dict() à la classe décorée permettant de retourner un dictionnaire de l'objet (ex : {id_base_site : 7, nom_taxon : Potentilles du Dauphiné}), alors que le @geoserializable ajoute la méthode as_geofeature() permettant de transformer un objet en geojson.

c. Fichier blueprint.py

```python
@blueprint.route('/sites', methods=['GET'])
@json_resp
def get_sites_zp():
    '''
    Retourne la liste des ZP
    '''
    parameters = request.args
#    , TBaseVisits.id_base_visit

    q = (
        DB.session.query(
            TInfoSite,
            func.max(TBaseVisits.visit_date),
            Taxonomie.nom_complet,
            func.count(TBaseVisits.id_base_visit),
            BibOrganismes.nom_organisme
        ).outerjoin(
            TBaseVisits, TBaseVisits.id_base_site == TInfoSite.id_base_site
        ).join(
            Taxonomie, TInfoSite.cd_nom == Taxonomie.cd_nom).outerjoin(
            corVisitObserver, corVisitObserver.c.id_base_visit == TVisiteSFT.id_base_visit
        ).outerjoin(
            TRoles, TRoles.id_role == corVisitObserver.c.id_role
        ).outerjoin(
            BibOrganismes, BibOrganismes.id_organisme == TRoles.id_organisme
        ).distinct()
        .group_by(TInfoSite, Taxonomie.nom_complet, BibOrganismes.nom_organisme
                  )
    )

    if 'id_base_site' in parameters:
        q = q.filter(TInfoSite.id_base_site == parameters['id_base_site'])
    if 'id_application' in parameters:
        q = q.join(
            corSiteApplication, corSiteApplication.c.id_base_site == TInfoSite.id_base_site
        ).filter(corSiteApplication.c.id_application == parameters['id_application'])
    if 'cd_nom' in parameters:
        q = q.filter(TInfoSite.cd_nom == parameters['cd_nom'])
    if 'organisme' in parameters:
        q = q.filter(BibOrganismes.nom_organisme == parameters['organisme'])

    if 'year' in parameters:
        # relance la requête pour récupérer la date_max exacte si on filtre sur l'année
        q_year = (
            DB.session.query(
                TInfoSite.id_base_site,
                func.max(TBaseVisits.visit_date),
            ).outerjoin(
                TBaseVisits, TBaseVisits.id_base_site == TInfoSite.id_base_site
            )
            .group_by(TInfoSite.id_base_site)
        )
```

Figure 22: Une fonction dans blueprint.py

Les routes permettent d'interagir avec la base de données qui sont regroupées dans ce fichier. Chaque route interroge une ou plusieurs classes (qui se trouvent dans models.py de ce module ou dans gn_monitoring) en utilisant la méthode query()[15] d'une session donnée. Une session de SQLAlchemy permet de gérer les transactions SQL, c'est-à-dire un ensemble des requêtes. Elle est la source de toutes les instructions SELECT générées par l'ORM.

La plupart des routes ici utilisent la méthode HTTP GET. Elles demandent une ressource au serveur afin de récupérer des données. En fonction de la demande, le serveur va renvoyer une réponse (dont 2xx=succès ; 4xx=faute de la part du client ; 5xx : faute de la part du serveur).

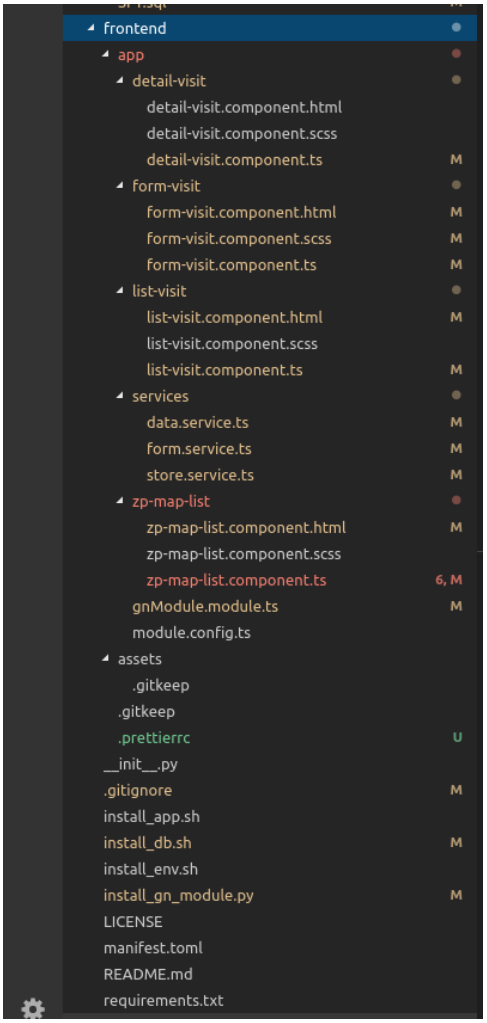
On utilise le protocole de communication HTTP via Python. Ce langage de programmation se charge de se connecter au serveur, d'envoyer/ de recevoir les requêtes et de fermer la connexion.

Pour pouvoir poster des visites dans la base de données, nous avons recouru à la méthode POST. Cette requête POST est envoyée via le formulaire HTML des visites (composant form-visit) et aura pour résultat un changement sur le serveur, une modification de la ressource. C'est également la seule route sur laquelle on fait appel à la méthode check_user_cruved_visit pour vérifier le droit des utilisateurs.

Tout au long du fichier, nous rencontrons souvent l'objet request de Flask, qui représente la requête exécutée par le client. La méthode request.args renvoie un dictionnaire contenant les paramètres présents dans l'URL (la partie dans l'URL après le point d'interrogation).

[15] ***La méthode query()*** : sert à construire une requête SQL.

3.2.4 La partie Front-end



C'est la partie où j'ai investi le plus de temps pour apprendre, concevoir, mettre en place, approfondir et modifier.

Le choix de technologies est porté sur Angular4 pour le front-end. C'est un framework lourd, mais adapté à un grand projet comme GeoNature2, notamment pour obtenir un découpage pertinent et fonctionnel en modules suffisamment autonomes.

Le cœur du front-end se retrouve dans le dossier app, lui-même est divisé en plusieurs composants. Chaque composant est comporté de 3 fichiers : un fichier HTML (la vue), un fichier TypeScript (le code fonctionnel) et un SCSS (la feuille de style).

Figure 23: Contenu du répertoire front-end du module Suivi Flore Territoire

L'organisation de l'affichage est presque similaire pour chacun des composants :

- A gauche de l'écran : un fond de carte. Ceci est possible à l'aide des composants cartographiques GeoNature et de la librairie JavaScript Leaflet.
- A droite de l'écran : le contenu de ce composant.

Elle se différencie bien-sûr en fonction de l'utilité et de la fonctionnalité de chaque composant.

a. Le composant zp-map-list

Ce composant permet d'afficher la liste de toutes les ZP suivies du module Suivi Flore Territoire, et leur représentation cartographique correspondante.

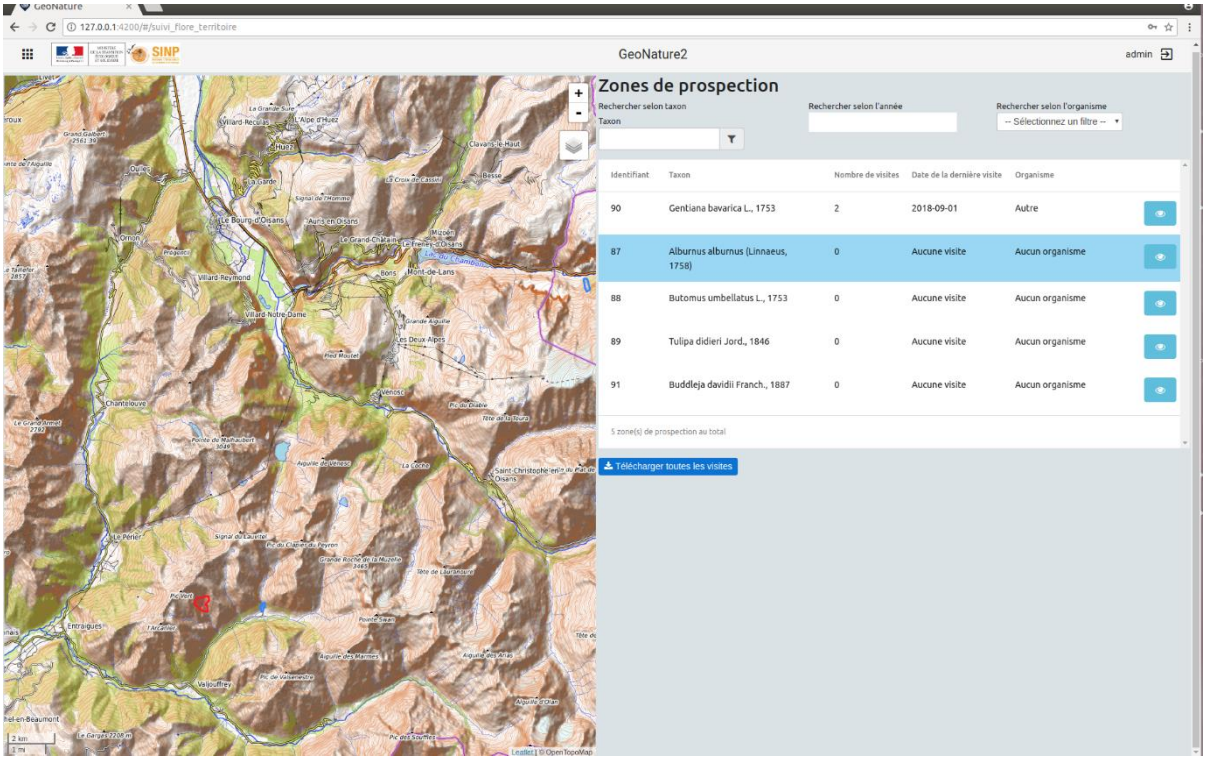


Figure 24: L'affichage du composant zp-map-list

Le tableau des ZP est obtenu grâce au sélecteur <ngx-datatable>. C'est un composant d'Angular permet de présenter les données volumineuses et complexes sous un format de table très flexible et légère. Nous pouvons la filtrer, la trier ou la paginer.

En plus des ZP, ce composant donne à l'utilisateur la possibilité de synchroniser la carte avec la liste des ZP. Le service MapListService de GeoNature offre des fonctions permettant facilement de réaliser.

- Quand l'utilisateur clique sur une ZP sur la carte, le MapListService expose la propriété selectedRow qui est un tableau contenant l'identifiant de la ZP choisie. L'élément sélectionné sera surligné dans la liste.
- A l'inverse, au clic sur une ligne du tableau, la fonction MapListService.onRowSelected() permet de zoomer sur la ZP sélectionnée et de changer la couleur de celle-ci sur la carte.

Pour exporter toutes les données sur cette page (c'est-à-dire toutes les visites existantes dans ce module), nous avons créé un composant dédié uniquement aux téléchargements, dont son sélecteur est <pnx-modal-download>. Quand on clique sur ce bouton, un modal de téléchargement sera affiché avec les formats de fichiers disponibles. Ce composant prend 2 inputs obligatoires :

- [pathDownload] : les données sont téléchargées grâce à ce chemin (interroge la base de données et le serveur).
- [exportFormat] : à quel format l'utilisateur veut exporter les données ? Dans le cadre de notre module, shapefile est le format par défaut, il y a également le format geojson et celui csv.

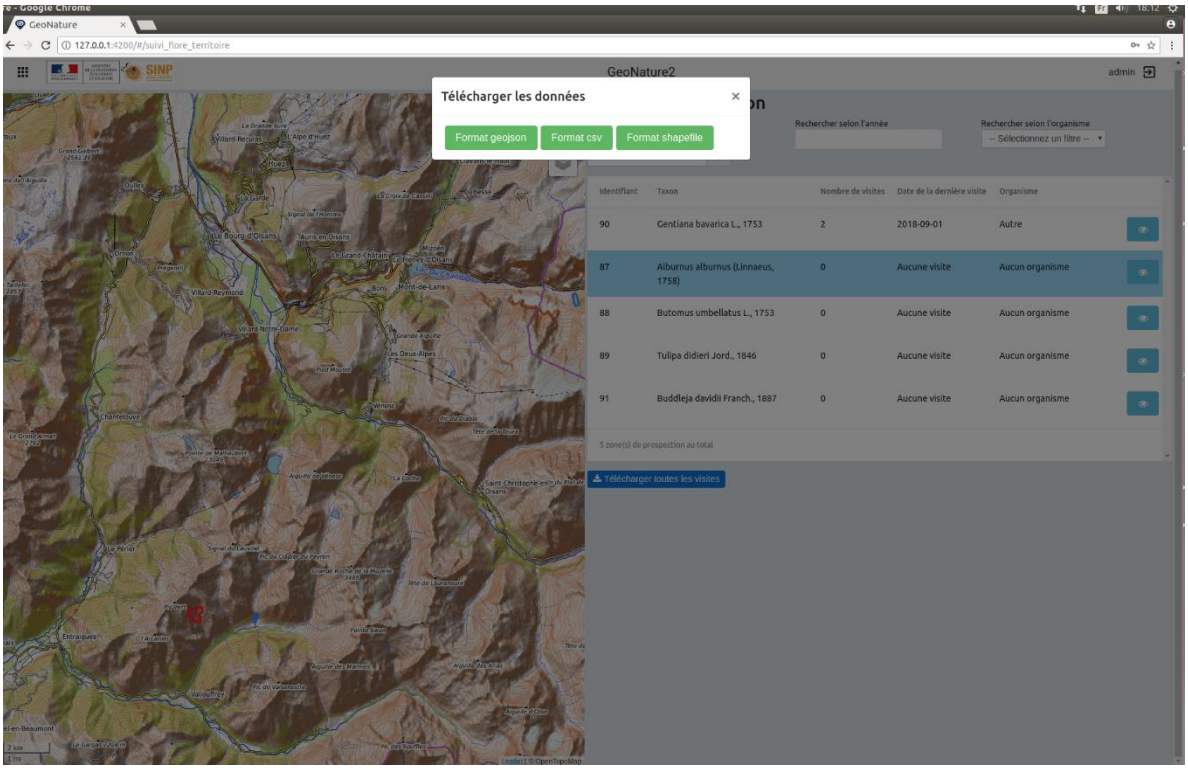


Figure 25: Le modal pour télécharger les données

b. Les services

Ce répertoire a pour but de centraliser des parties de notre code et des données qui sont utilisées par plusieurs parties de notre module. Les services permettent de :

- Ne pas avoir le même code doublé ou triplé à différents niveaux de l'application. Cela facilite la maintenance, la lisibilité et la stabilité du code.
- Ne pas copier inutilement des données. Cela évite de nombreuses erreurs potentielles et allège le poids du code.

Pour être utilisé, un service doit être injecté (décorateur @Injectable). Nous avons injecté les 3 services : Store, Form, et Data dans gnModule.module pour rendre disponible une seule instance par service à tous les autres composants de notre module. Ensuite, depuis chaque partie où nous voulons utiliser le ou les services, nous les déclarons comme argument de son constructeur.

StoreService

Ce service regroupe toutes les données qui sont utilisées plusieurs fois par plusieurs parties du module. Elle dispose également des méthodes communes utilisant ces données communes.

FormService

Ce service est conçu pour construire une instance du formulaire, qui sera ensuite utilisée par le composant form-visit. Nous avons fait appel à la méthode group de FormBuilder, qui est une classe utilitaire[16] avec des méthodes facilitant la création d'objet FormGroup[17]. Elle prend comme argument un objet où les clés correspondent aux noms de contrôles souhaitées (les champs de formulaires), et les valeurs correspondent à leur valeur par défaut.
Une visite est identifiée par la date de visite et l'observateur qui a fait la visite. Par conséquent, nous y avons ajouté les Validators afin que ces champs deviennent obligatoires pour l'utilisateur.

DataService

Ce fichier contient des routes dont nous avons besoin pour faire des appels au back-end, afin d'enregistrer ou charger les données, ou encore d'effectuer des calculs ou des modifications des données que nous ne souhaitons pas faire faire par le frond-end.

Angular met à disposition un service appelé HttpClient qui permet de créer et d'exécuter des appels HTTPs (fait en AJAX - Asynchronous JavaScript and XML) et de réagir aux informations retournées par le serveur.

Ensuite, en fonction de la demande de chaque composant, nous allons recourir à une ou plusieurs routes déclarées dans ce fichier : retourner toutes les ZP ou une ZP spécifique en fonction du paramètre passé, afficher les informations détaillées d'une visite, poster une visite dans la base de donnée, exporter les visites, etc.

---

[16] *Classe utilitaire* : Une classe sans état, ne contient que des méthodes statiques (méthodes de classes) et ne peut pas être instanciée. Elle dispose des méthodes pour plusieurs autres classes.

[17] *FormGroup* : une classe d'Angular, permet de regrouper plusieurs FormControl (chaque champ de saisie d'un formulaire est relié à un FormControl) en un objet.

c. Le composant list-visit

Toutes les visites d'une ZP sélectionnée seront affichées grâce à ce composant. Il contient des informations générales et détaillés du site, ainsi le nombre de visite effectué sur cette zone, la date de chaque visite, le nom d'observateur et d'une manière global le contenu de la visite et son avancement (via la colonne Présence/ Absence et nous comparons avec le nombre de maille en total).

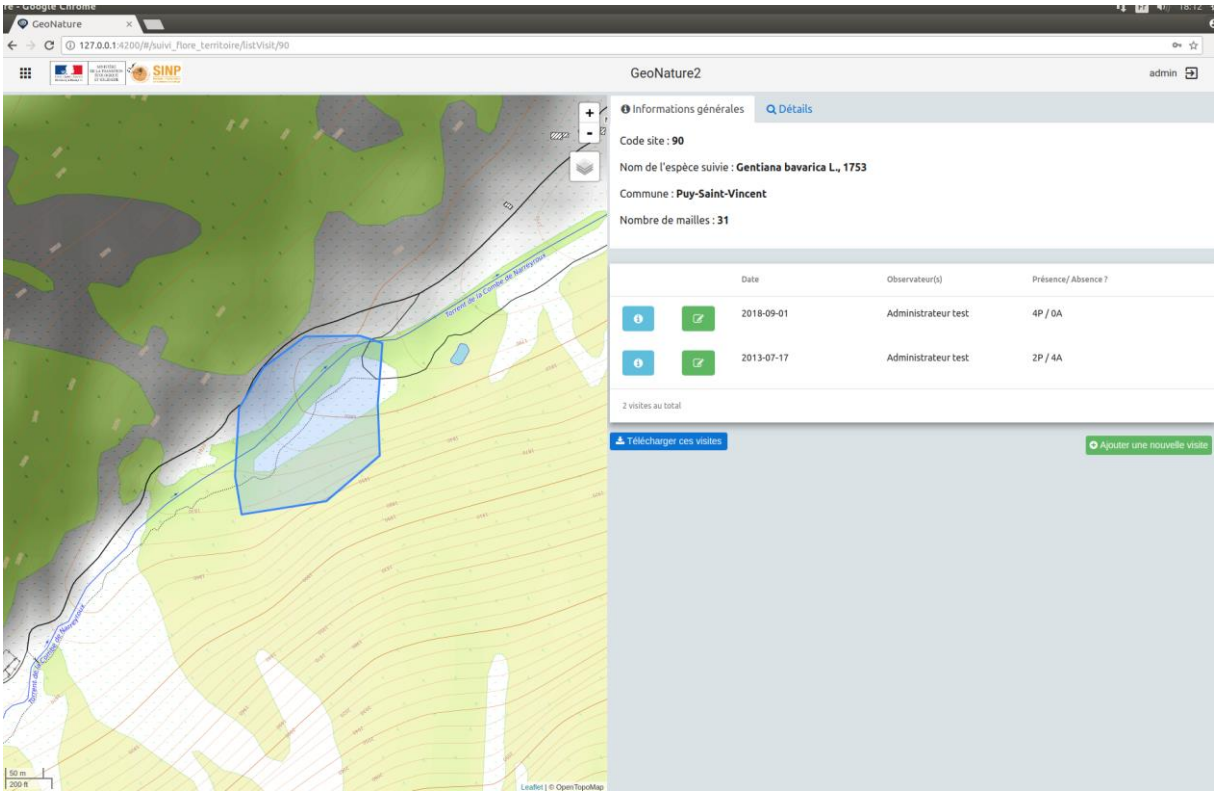


Figure 26: L'affichage du composant list-visit

Nous avons également la possibilité de visualiser en détail cette visite, ou la modifier. Par ailleurs, le bouton « ajouter une nouvelle visite » permet à l'utilisateur de créer une nouvelle visite et d'enregistrer des informations propres à cette visite. Il est également possible de télécharger toutes les visites sur cette ZP.

d. Le composant form-visit

Ce composant propose à l'utilisateur un formulaire pour qu'il puisse poster une nouvelle visite, ou éditer une visite déjà existée.

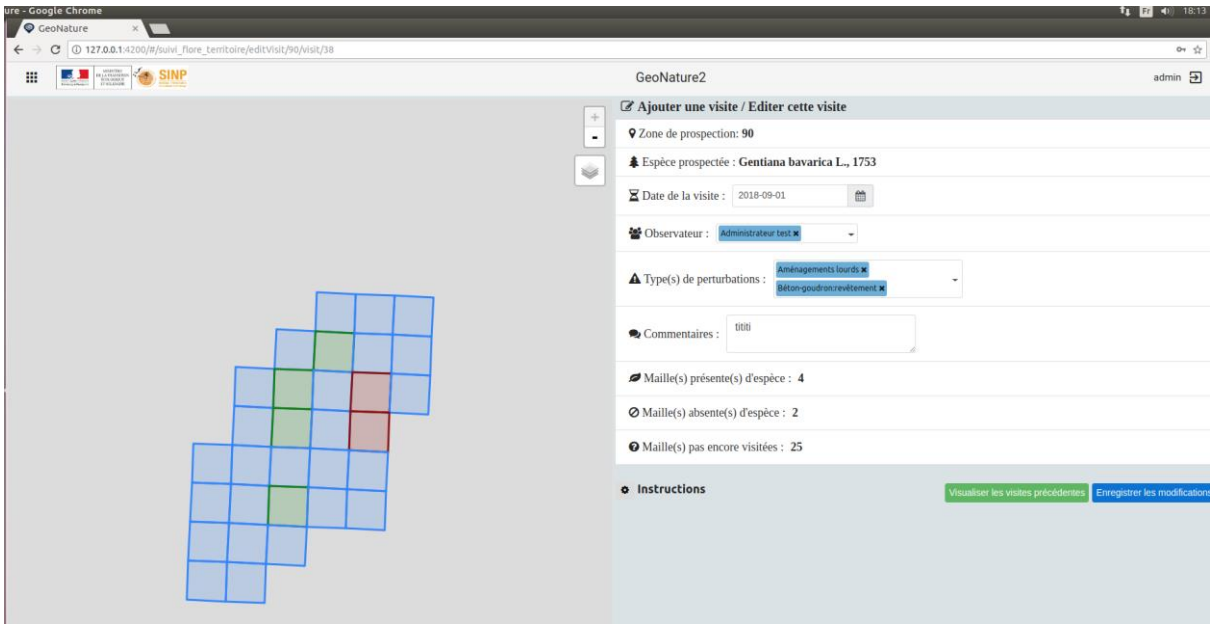


Figure 27: L'affichage du composant form-visit (edit)

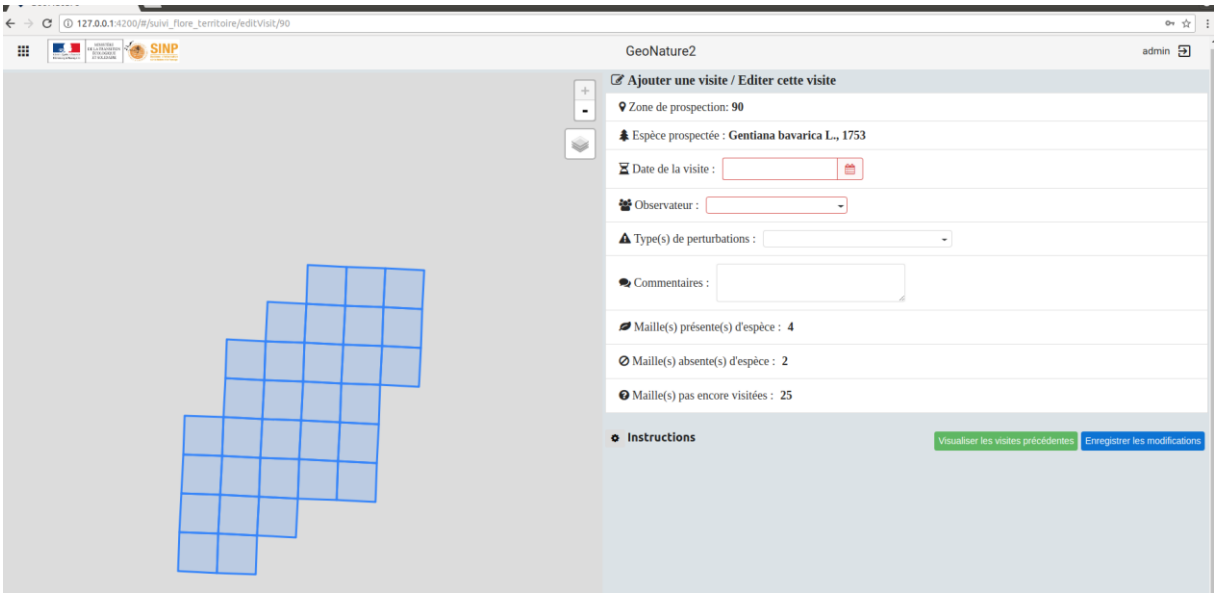


Figure 28: L'affichage du composant form-visit (add)

Au début, nous avons eu 2 composants différents pour cette action : un pour créer une nouvelle visite, et un pour éditer une ancienne. Afin d'optimiser le résultat du travail tout en simplifiant et en allégeant les instructions, nous avons factorisé les composants permettant de créer et d'éditer une visite en un seul composant. Autrement dit, nous avons unifié et rassemblé les suites de code

identiques et dispersées en un seul composant, pour améliorer la lisibilité du code et en faciliter la correction et les modifications ultérieures.

Sur cette page, l'utilisateur est amené à renseigner la date de visite, son nom ou le nom de la personne ayant fait la visite, les perturbations observées sur cette ZP et les commentaires s'il y en a. Ce formulaire est obtenu grâce à l'instance de FormGroup que nous avons créé dans FormService, et également aux composants formulaires de GeoNature.

Il est possible également de saisir sur la carte les mailles présentes et absentes d'espèce suivie.

- Toutes les mailles sont colorées en bleu par défaut.
- Quand une espèce est identifiée en tant que « présence » dans une maille, l'utilisateur clique gauche sur cette maille : sa couleur change en vert.
- Quand une espèce est identifiée en tant que « absence » dans une maille, l'utilisateur clique droite : sa couleur change en rouge.
- Si l'utilisateur s'est trompé et a cliqué sur une maille qui n'a pas encore été visitée, il a la possibilité de cliquer deux fois sur cette maille pour qu'elle redevienne bleu (l'état initial).

Pour les utilisateurs « novices », un bloc d'instructions sera affiché quand l'utilisateur clique sur le bouton Instructions.

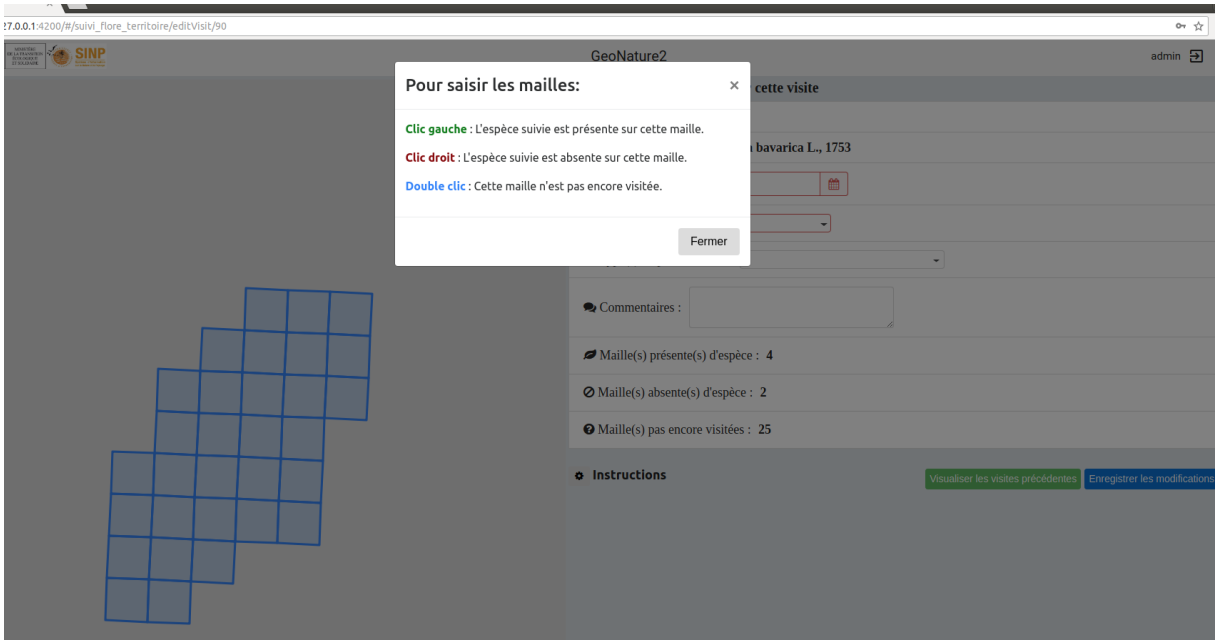


Figure 29: Le modal d'instructions

Après avoir rempli ce formulaire de visite, l'utilisateur peut poster la nouvelle visite/ enregistrer les modifications de l'ancienne visite, ou abandonner la saisie. Ils seront ensuite ramenés à la page de liste des visites d'une ZP

e. Le composant detail-visit

Ce composant comprend tous les détails d'une visite déjà réalisée : son appartenance géographique (à quelle ZP?), l'espèce suivie, sa date, etc. Le fonds de carte offre à l'utilisateur la possibilité de visualiser quelles sont les mailles présentes, absentes de l'espèce et les mailles non visitées. En fonction de leur statut enregistré dans la base de données, nous colorons par défaut ces mailles tout en respectant le code de couleur indiqué dans la partie form-visit.

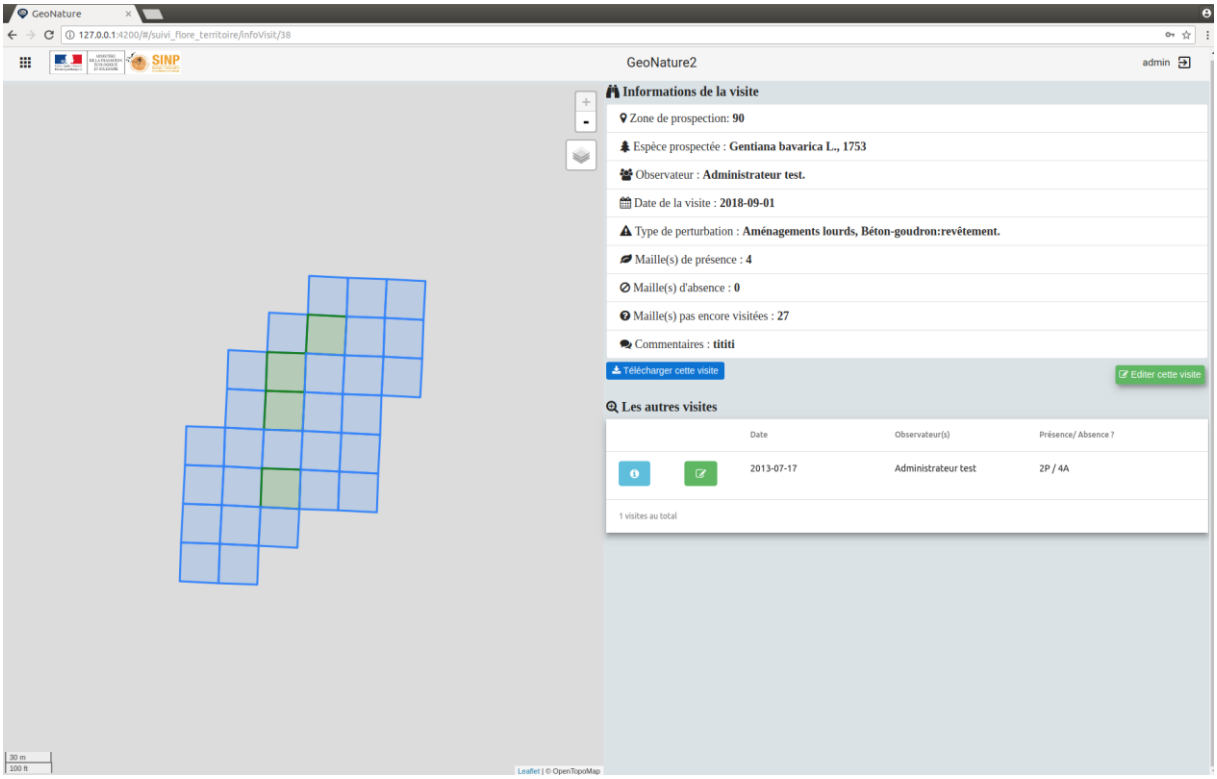


Figure 30: L'affichage du composant detail-visit

Par ailleurs, l'utilisateur peut aussi accéder aux autres visites de la ZP directement à partir de cette page grâce à la table des autres visites affichées juste en bas du bloc d'informations. Il est également possible d'exporter cette visite, ou de l'éditer.

4. Les tests

Cette étape est incontournable de tout développement informatique. Son objectif principal est d'identifier un nombre maximum de comportements problématiques de l'application et de vérifier que le livrable répond bien aux besoins demandés.

4.1 Les tests unitaires

Ces tests cherchent à tester individuellement les composants de notre module. Chaque composant est testé de manière séparée.

Les tests unitaires sont souvent réalisés d'une manière automatisée, afin d'effectuer plusieurs vérifications rapidement, mais aussi de répéter cette opération à l'infini.

Faute de temps, nous n'avons pas pu écrire des tests unitaires automatisés. Je les ai faits manuellement.

4.2 Les tests d'intégration

Après avoir faire des tests unitaires, les tests d'intégration sont exécutés pour s'assurer le bon fonctionnement de la mise en œuvre conjointe des différents composant entre eux, et dans leur environnement exploitation définitive.

De même, les tests d'intégration sont effectués assez souvent une fois que j'ai tout écrit les codes fondamentaux du module, mais encore une fois, de manière manuelle seulement.

Néanmoins, GeoNature utilise Travis CI[18] pour lancer des tests automatisés à chaque commit. Après chaque modification effectuée et poussée sur le dépôt GitHub, ces tests assurent le fonctionnement général, la stabilité du code, et aussi la non-régression[19]. Ceci permet d'éviter les gros bugs difficiles à déceler.

4.3 Les tests système

Le but de cette phase est de vérifier la conformité de l'application développée avec le cahier des charges initial, et aussi si le travail répond bien à nos attentes sur le plan technique : est ce que les informations postées sont bien stockées dans la base de données ? L'utilisateur est-t-il bien informé de l'obligation de certains champs vides ? Recevra-t-il un message en cas de succès/ d'échec de l'enquête ?

---

[18] *Travis CI* : un logiciel libre qui fournit un service en ligne utilisé pour compiler, tester et déployer le code source des logiciels développés, notamment en lien avec le service d'hébergement du code source GitHub (Wikipédia).

[19] *Non-régression* : pour s'assurer, après chaque modification, que des défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel.

Ces tests ont relevé certains points nécessitant de travailler davantage, notamment les messages de signalement et les validateurs au niveau des boutons.

4.4 Les tests d'acceptation

Les tests d'acceptation visent à confirmer si le produit final correspond bien aux besoins des utilisateurs finaux. Cette phase de test cherche d'abord à faire valider le développement entre différentes équipes du projet, puis auprès des utilisateurs.

La réunion du 4 septembre 2018 entre l'équipe du développement du PNE et l'équipe du CBNA avait donc pour cet objectif. Globalement, le travail fourni était bon et convenable aux objectifs fixés, même s'il restait quelques bugs mineurs à corriger et quelques détails ergonomiques à revoir. Les autres demandes concernant l'ajout de quelques options supplémentaires ont été prises en compte et seront priorisées, en fonction du temps restant de mon stage.

Bientôt, notre module sera présenté auprès du public du réseau Alpes-Ain. Nous recevrons à ce moment les premiers retours des autres utilisateurs du réseau Flore Sentinelle, qui pour l'instant ne connaissent pas le fonctionnement de notre module. Selon les feedbacks, nous pourrons améliorer et adapter le module pour qu'il devienne hautement acceptable et appréciée.

5. La finalisation du projet

Actuellement, nous sommes en phase de finalisation du projet.

Le module développé est bien fonctionnel. Cependant, pour que le livrable soit entièrement satisfaisant, à la fois sur le contenu logique et l'interface graphique, il nécessite des modifications et des ajouts supplémentaires, à savoir :

- Compléter la partie de l'ergonomie informatique.
- Créer une fonction au niveau SQL afin de récupérer la valeur d'une variable.
- Créer une fonction permettant de charger un GPX sur la carte.
- Ajouter le champ de recherche des ZP d'une commune.

D'ici jusqu'à la fin de mon stage, il reste plusieurs actions à déployer, représentant pour moi un challenge copieux mais alléchant.

C. Les autres activités

1. Les sorties de terrain.

Depuis le début du stage, j'ai pu participer à trois sorties de terrains : deux avec l'équipe du CBNA, et une avec certains membres du pôle connaissance du PNE.

Grâce à ces sorties, surtout les 2 dernières, j'ai donc mieux compris l'intérêt du développement des protocoles Suivi Flore Territoire et Suivi Station. Même si le temps ne nous a pas permis de développer d'autres modules que celui présent, ces visites de terrain m'ont beaucoup aidé à réfléchir et à me projeter dans la construction et l'organisation des codes, ainsi que les fonctions intéressantes à utiliser pour développer le module demandé.

De plus, ces sorties de terrain m'ont appris énormément de choses à propos des espèces faune et flore, de l'histoire des sites visités et aussi de l'esprit d'équipe. Cela est très enrichissant et formateur pour moi, non seulement en terme de connaissances théoriques/ pratiques, mais il m'a donné également le sentiment d'accomplissement et de plénitude.

2. Workshop Geonature

Le 24 et 25 septembre 2018, les parcs nationaux de France et l'Agence française pour la biodiversité organisent un workshop technique GeoNature2, auquel je participerai. Ce workshop est destiné à des géomaticiens, informaticiens et développeurs avec 3 ateliers pour leur permettre de prendre en main l'outil :

- Le déploiement/ L'installation de de GeoNature2.
- La manipulation de la base de données.
- Le développement.

Ces journées me seront bénéfiques pour pouvoir approfondir mes connaissances sur GeoNature, partager et aussi échanger nos expériences sur le développement des modules autonomes de GeoNature avec d'autres participants

V. Conclusion

Du mois de mai jusqu'au mois de septembre 2018, avec la participation active de tous les membres de l'équipe, le module Suivi Flore Territoire pourra enfin être déployé très prochainement. Une grande partie de nos objectifs fixés au début de notre stage ont été atteints. Néanmoins, certaines perspectives envisagées ne sont malheureusement pas réalisées, notamment le développement du module Suivi Station, ou encore la mise en place d'un système de traduction pour la recherche du taxon (la recherche actuelle ne prend en compte que des noms en latins (nom scientifique) et pas en français (nom usuel)). Ils seront des pistes de travaux pour nos futures actions.

D'un point de vue personnel, la possibilité d'effectuer ce stage est l'opportunité la plus géniale que je n'aurais jamais imaginé. Etant hésitée au moment où j'ai dû donner la réponse aux différentes propositions de stage, je suis finalement ravie d'avoir fait le bon choix. Au niveau du contenu informatique, il s'agit d'un stage le plus complet possible : j'ai pu acquérir et consolider mes compétences en Python (back-end), Angular (front-end), PostgreSQL (gestion de la base de données), Flask, API-Rest, Bootstrap, les scripts bash Linux. Cette expérience m'a confortée dans mon projet professionnel, m'a encouragé davantage de continuer et de persévérer en ce domaine passionnant de la programmation et du développement web.

Pour finir, j'ai également eu énormément de chance de pouvoir travailler dans une ambiance de travail aussi dynamique et agréable, avec des collègues à la fois compétents, toujours à l'écoute et très dévoués. Ces 5 mois de stage resteront profondément dans ma mémoire, impacteront fortement mon style de travail, et également, seront gravés à jamais dans mon cœur.

VI. Bibliographie

https://github.com/PnX-SI/GeoNature

https://github.com/PnX-SI/gn_module_suivi_flore_territoire

http://www.ecrins-parcnational.fr/

http://www.cbn-alpin.fr/

https://docs.python.org/3/

https://angular.io/docs

http://flask.pocoo.org/