Ecole nationale de la statistique
et de l'analyse de l'information

Projet informatique

Etudiants 2A

API cadastrale

Etudiants :
Marie Colin
Hadrien Gôme
Lucas Laloue
Tanguy Legrand
Julien Sawadogo

Tuteur :
Thierry Mathe
Enseignant :
Rémi Pepin

2022 - 2023

26 novembre 2022

Table des matières

1 Introduction

Aujourd'hui, les données sont de plus en plus accessibles à mesure que les technologies de l'information et de la communication se développent. Ainsi, les services publics français mettent un grand nombre de données à disposition du public, permettant ainsi à celui-ci d'avoir accès à une information de plus en plus importante. Un enjeu majeur est de contrôler la qualité de ces données. Il est possible pour n'importe qui, particulier comme professionnel, de développer des applications permettant de contrôler la qualité des données mises à disposition.

Par exemple, le service statistique du ministère de la Transition écologique met à disposition du public Sitadel, la base de données des permis de construire (PC) et autres autorisations d'urbanisme. Ce répertoire permet de publier des chiffres sur le nombre de PC autorisés et, donc, de connaître facilement le nombre de logements en construction. Cependant, un PC pouvant s'étendre sur plusieurs parcelles couvrant plusieurs communes, voire plusieurs départements, il est très compliqué de ne pas avoir de redondance dans la base. Les PC peuvent, par exemple, se retrouver dans la base de deux départements différents.

Ce phénomène est connu mais il pourrait être intéressant de le quantifier pour connaître sa fréquence, pour savoir si les erreurs engendrées ont un impact sur les chiffres de logements nouveaux publiés au niveau communal.

Une solution possible pour quantifier ce résultat est proposé par ce projet. L'utilisation d'une API va permettre à un utilisateur quelconque de répondre à ce type de question.

Notre API doit tout d'abord permettre de constituer une base de données pour enregistrer les parcelles et communes contiguës. Nous nous appuyons pour construire notre base sur les fichiers GeoJSON du site https ://cadastre.data.gouv.fr/datasets/cadastre-etalab.

2 Analyse fonctionnelle : le contexte de notre API

2.1 Description générale du fonctionnement du programme

Dans cette partie, nous allons décrire plus précisément le fonctionnement général de notre programme. Cela se fera notamment avec l'utilisation de diagrammes d'utilisation, d'activité et un diagramme des classes.

Le diagramme d'utilisation (figure 1) permet de mieux visualiser ce qu'un utilisateur pourra faire avec le programme. L'administrateur met à jour les données. Dans cette étape, une base de données est créée et est stockée quel que soit le choix de l'utilisateur. Cette base de données sera présentée dans la partie 2.2 de ce rapport.

Ensuite, l'utilisateur pourra émettre une requête et il recevra une réponse de la part de l'API sous forme de Json. L'utilisateur pourra répondre aux questions suivantes :

- Quelles sont les communes ayant des parcelles contiguës/limitrophes a une commune donnée ?
- Quelles sont les parcelles en limite d'une commune donnée ?
- Quelles sont les parcelles contiguës a une parcelle donnée ?

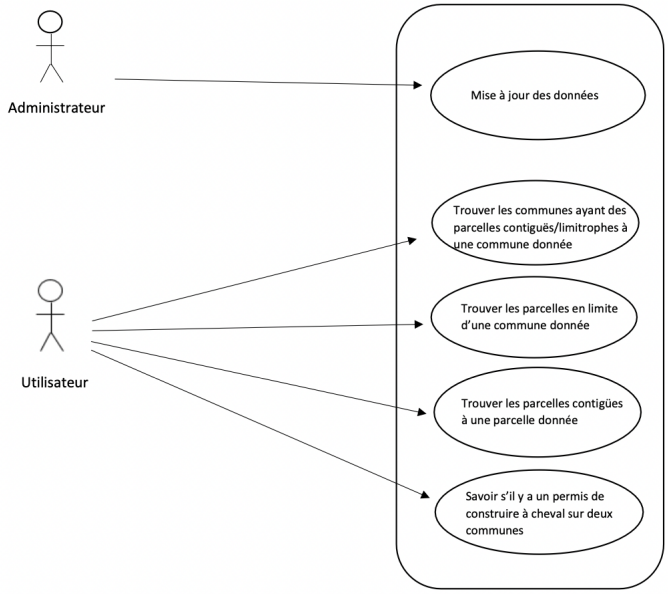

FIGURE 1 – **Diagramme des cas d'utilisation.**

Cette partie a pour but de détailler les différentes étapes possibles du programme. Plusieurs personnes interviennent : l'administrateur (qui représente notre API) et l'utilisateur.

Premièrement, l'administrateur choisit la zone géographique couverte par la base de données. Le programme télécharge ensuite les communes et les parcelles selon le choix de l'année et des départements souhaités par l'administrateur. Des tables contenant les communes adjacentes entre elles, les noms des communes et les parcelles en bordure de chaque commune sont alors stockées dans la base de données et accessibles "directement".

Cependant, l'utilisateur peut vouloir s'intéresser à une autre année et à d'autres départements lorsqu'il utilise l'API. C'est pour cette raison que nous avons fait fonctionner notre programme de la manière suivante : Lorsque l'utilisateur effectue une requête pour répondre à l'une des trois questions possibles, il entre en paramètres un identifiant de commune ou de parcelle, en fonction de la question, et une date. Si l'identifiant fait référence à un département de la base et si l'année est la même que celle des données de la base, l'API renvoie une réponse en lisant simplement le contenu de la base de données et en faisant de légers traitements si nécéssaire. La réponse est alors très rapide. Si au contraire, les paramètres ne correspondent pas à ceux de la base de données, l'API va automatiquement télécharger les données nécéssaires sur le site du cadastre. Ces données seront ensuite traitées et la réponse envoyée à l'utilisateur de la même manière que dans le premier cas. La seule différence est le temps de traitement qui pourra être légèrement plus long (2 min maximum).

La méthode de téléchargement choisie, qui consiste à laisser l'administrateur choisir la zone géographique permet de faire face aux problèmes de stockage et de mémoire. En effet, les données qui vont être utilisées sont trop lourdes pour être téléchargées en même temps. De plus, il est inutile de traiter une zone géographique ou une année qui ne nous intéresse pas car cela peut allonger le temps de calcul. Cela ne gène cependant pas l'utilisation de l'API qui peut s'adapter à n'importe quelle requête comme nous l'avons vu dans le paragraphe précédent.

L'utilisateur intervient après ce processus de création de base de données pour choisir une des différentes requêtes. Lorsque la base est déjà existante, les requêtes peuvent être exécutées directement sans avoir besoin de reconstruire la base. C'est le cas par exemple de la requête 1 : "Quelles sont les communes ayant des parcelles contiguës/limitrophes a une commune donnée ?".

Si certaines requêtes nécéssitent les résultats d'autres requêtes, ces résultats intermédiaires sont stockés dans des variables pour être réutilisés.

Les résultats finaux sont stockés dans un Json. Pour les requêtes 2 et 3, il est plus simple de comprendre le fonctionnement de notre API en utilisant des diagrammes d'activités.

Le but de la requête 2 est de connaître les parcelles en limite d'une commune donnée, ce qui est décrit dans le diagramme qui suit.
L'utilisateur fournit en entrée l'identifiant de la commune pour laquelle il veut connaitre les parcelles en limites. Le programme s'assure également au préalable le format de la requête et en particulier de ses paramètres est le bon, ce qui permet de renvoyer directement une erreur à l'utilisateur en cas de problème. Pour répondre à cette question, le programme charge les parcelles de la commune sélectionnée, et compare les points de coordonnées de la communes et de toutes les parcelles contenues dans celle-ci. Si une parcelle et la commune partagent un segment en commun, alors la parcelle est en limite de la commune et est donc stockée. Le programme s'arrête quand tous les points ont été comparés et retourne donc le résultat dans un fichier Json.

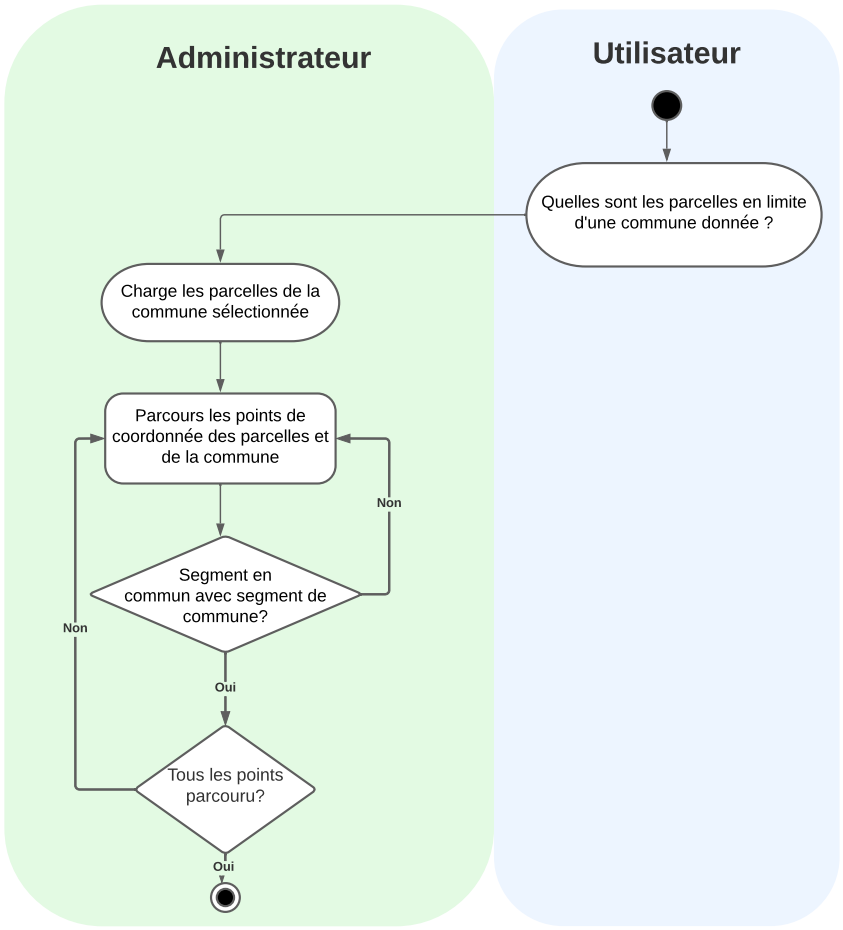

FIGURE 2 – **Diagramme d'activité de la requête 2**

La requête 3 renvoie les parcelles contiguës à une parcelle que l'utilisateur a choisie.
Le programme demande à l'utilisateur la parcelle dont il cherche les parcelles contiguës. Celle-ci correspond au paramètre d'entrée. Le programme s'assure que le paramètre d'entrée correspond bien à un identifiant de parcelle.
Avant de charger les données, le programme vérifie si la parcelle sélectionnée est en limite d'une commune afin de savoir s'il est nécéssaire de charger les parcelles des communes contigues. Autrement dit, l'exécution de cette requête passe aussi par l'exécution de la requête 2 qui va chercher si la parcelle sélectionnée est en limite d'une commune. Grâce à l'identifiant de la parcelle, il est possible de récupérer la commune à laquelle elle appartient, et donc le programme peut exécuter la requête 2. Cette stratégie permet de ne pas parcourir trop de points inutilement.

Si la parcelle est en limite de la commune, le programme charge les parcelles de sa commune et des communes adjacentes, sinon la parcelle n'est pas en limite et donc le programme charge uniquement les parcelles de la commune. Ensuite, de la même manière que dans les requêtes précédentes, le programme regarde s'il y a des segments en commun et les stocke. Une fois que tous les segments sont parcourus, le programme retourne le résultat dans un Json.

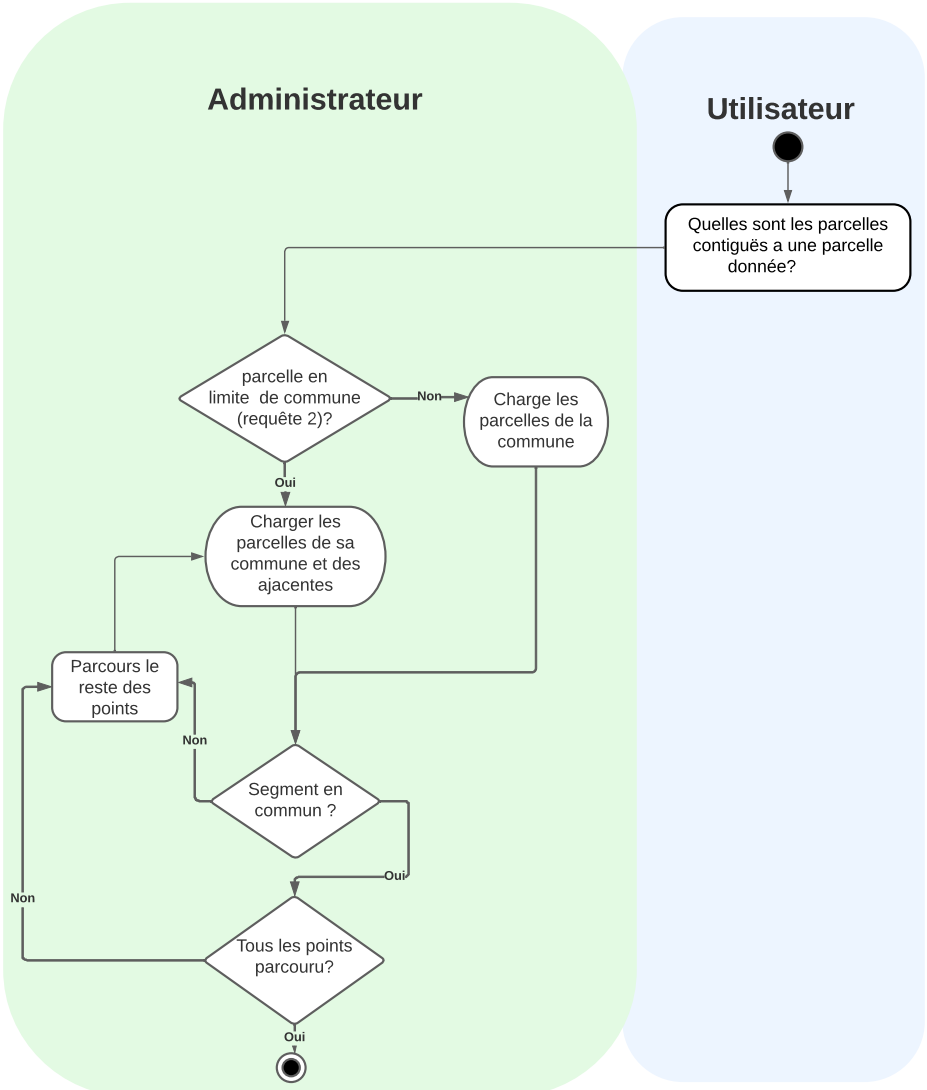

FIGURE 3 – **Diagramme d'activité de la requête 3.**

2.2 La base de donnée

Une base de données est un ensemble structuré de données, mémorisé sur un support permanent. Elle permet le stockage d'informations qui sont couramment utilisées sous forme de tables. Les diagrammes précédents montrent que la réponse d'une requête d'un utilisateur est un ensemble de communes ou un ensemble de parcelles ; c'est pourquoi nous avons créé une base de données contenant trois tables. La première contient les identifiants et les noms des communes, la deuxième contient les associations de communes adjacentes et la troisième contient les parcelles en bordure de chaque commune pour l'année est les départements choisis par l'administrateur lors de l'initialisation de la base.

Ainsi, le diagramme de données de notre API se présente sous la forme suivante :

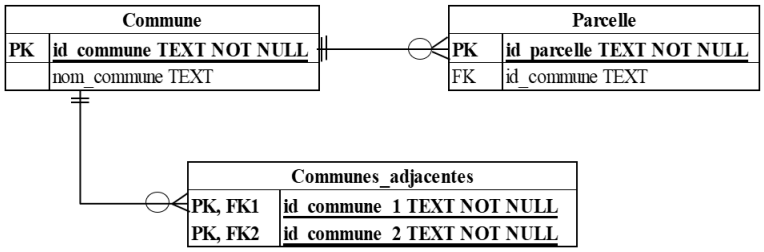

FIGURE 4 – **Diagramme de données.**

La base de données Commune nous permet d'enregistrer les identifiants et les noms des différentes communes pour une année et une zone géographique données. Cette base est construite grâce à une DAO qui récupère les données téléchargées sous forme de fichiers GeoJSON sur le site https ://cadastre.data.gouv.fr/datasets/cadastre-etalab par le client et transformées par la couche Creation_BDD. La base Parcelle est construite de la même façon et elle contient le numéro identifiant des parcelles en bordure des communes et le numéro identifiant la commune dans laquelle se trouve la parcelle. Quant à la base qui contient les différentes communes adjacentes, elle est construite avec le fichier GeoJSON des communes et grâces aux coordonnées géographiques des différentes communes. Dans cette base, chaque commune a p lignes qui correspondent aux nombre de communes qui lui sont adjacentes, déterminées grâce à un calcul fait par la couche Creation_BDD. Il faut noter que pour réduire la taille de la base, nous avons tenu compte du fait que si la commune A est contiguë à B alors la commune B est contiguë à A.

2.3 L'architecture macro de notre API

L'architecture macro de notre API repose sur le langage python. Notre script python est sur un serveur. L'utilisateur qui veut répondre à certaines requêtes utilise des requêtes HTML qui lui permettront d'utiliser notre API. Ces requêtes sont plus précisément reçues par le control layer de notre application qui va les transférer au service layer pour le traitement. Ce dernier utilisera toutes les autres couches du système pour répondre au cas par cas aux questions de l'utilisateur. Il renverra ensuite les réponses au control layer pour qu'il puisse à son tour les transmettre à l'utilisateur.

Au démarrage, l'API est mise en route en lancant le script __main__ à la racine. Les utilisateurs peuvent ensuite utiliser des requêtes HTML en respectant la syntaxe de l'API pour obtenir des

réponses à leurs questions. Les syntaxes à respecter sont présentées dans la partie 4.1.2 à la figure 7.

Les technologies utilisées comprennent les méthodes de la Programation orientée objet. En effet, notre travail a été découpé en couche, et chaque couche contient des classes qui communiquent entre elles par des relations, comme par exemple l'héritage. Nous avons également utilisé des principes de la programmation comme separation of concerns ou test driven developpement (présentée dans la partie 4.3). Ces deux pratiques permettent de limiter les erreurs . Par ailleurs, notre projet repose sur du téléchargement de fichiers GeoJson, de la lecture de page web pour connaitre les années disponibles sur le site et de la communication avec une base de données SQL.

3 Requête SQL : création de la base

La création de la base de données se fait en utilisant des requêtes SQL, les données brutes sont transformées en données plus intelligibles. La figure 5 montre un exemple de données brutes.

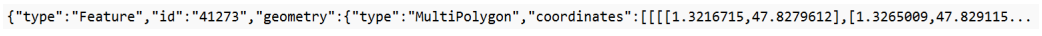

FIGURE 5 – **Exemple de données récupérées par le web service**

Le code de la création de la base de données est stocké dans un fichier qui se nomme init_basepy dans le dossier creation_BDD. Comme présentée dans la partie 2.2, la base de données comprend trois tables, dont une qui peut être optionnelle, ce sont les tables suivantes : commune, commune_contigüe et parcelle.
Elle interagit seulement avec la partie DAO de la couche métier. Elle prend les données qui ont été téléchargées et dézippées par le client puis mises en forme par creation_BDD.

La première étape de la création de la base de données consiste à vérifier que les tables n'existent pas, et de les supprimer si elles existent. En effet, si l'utilisateur demande une autre année que celle stockée en base, il faut recréer la base de données et donc supprimer l'ancienne.

La première table qui est créee est la table commune : une option y est ajoutée : "CASCADE". Cette option permet que si une commune est supprimée dans la table commune, cela impacte les autres tables en les supprimant ou en supprimant la commune ou la parcelle spécifiée dans les tables parcelles et commune_contigue. En effet, si une commune disparait de notre base, il n'est pas normal qu'une parcelle de cette même commune soit encore présente dans la base de données.

Les attributs sont stockées en format texte dans chaque table grâce à l'utilisation de l'option "VARCHAR".
La table parcelles est facultative et regroupe les parcelles des divisions administratives choisies. Elle est optionnelle car très lourde, mais son absence implique un nombre de fonctions limitées.

La base de données est reliée à des modules pythons tel que execution_creation_bdd ou execution_creation_bdd_communes , qui permettent de remplir la base. Ils font face aux problèmes de stockage. En effet, il n'est pas possible de rapidement télécharger toutes les parcelles de toutes les communes : les données sont trop lourdes. Ces modules permettent de décomposer la France en département, et de télécharger les communes ou parcelles du département indiqué et de ses départements limitrophes.

Cette base de données est stockée et est utilisé par le reste de notre API.

4 Architecture du code

4.1 La couche métier (business layer) : le cœur du projet

La couche métier est la partie du code qui apporte une plus-value. Ici, elle permet de répondre aux trois questions du projet : trouver les communes adjacentes à une autre commune, trouver les communes ou parcelles adjacentes à une parcelle. L'algorithme renvoie un message expliquant que la parcelle n'est pas adjacente à une autre commune si celle-ci n'est pas en bordure de sa commune. Nous avons regroupé notre code selon le principe de la separation of concerns (séparation des responsabilités). Ceci permet de découper le code en parties isolées qui traitent chacune un aspect précis de l'algorithme général. Cette couche est divisée en trois sous-couches.

4.1.1 Diagramme des classes objets métier

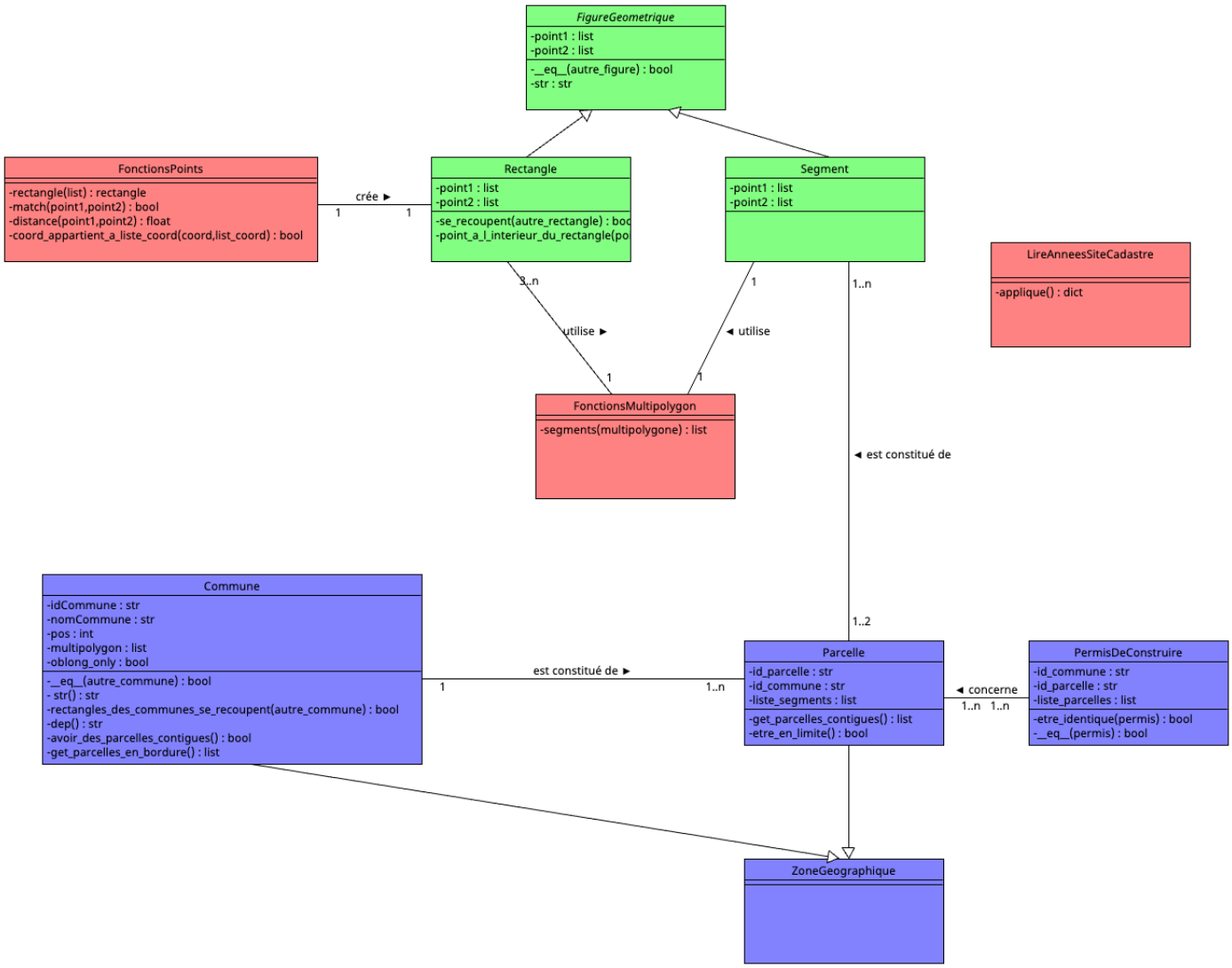

FIGURE 6 – **Diagramme des classes.**

Le but de notre architecture de code est de regrouper les classes en fonction de leur nature. En vert, les classes représentant des figures géométriques, en rouge les fonctions et en violet les objets réels propres à nos données. Chaque classe contient les méthodes en lien avec l'objet qu'elle représente et les classes de différentes natures s'utilisent entre elles.

En violet, nous avons implémenté les classes de zone géographique. La classe Zone géographique est une classe abstraite. Elle permet de regrouper la classe Commune et la classe Parcelle qui ont plusieurs points communs, à commencer par le fait qu'elles représentent toutes deux des territoires mais ont aussi des attributs en commun. La classe permis de construire a été mise dans la même catégorie de classes mais nous avons choisi de ne pas la faire hériter de zone géographique car ses attributs sont légèrement différents. En effet, contrairement aux autres classes violettes, le permis

de construire peut être attaché à plusieurs communes à la fois. Ces classes utilisent les formes géométriques afin par exemple de définir les rectangles encadrant les communes et les parcelles.

En rouge, nous avons codé des classes contenant des fonctions sur les points, sur les multiplolygones et une autre classe contenant une fonction pour lire les années sur le site du cadastre. Les classes fonctions points et fonctions multipolygone répondent aux très nombreux besoins de notre API en matière de traitement de tableaux de coordonnées. Ces classes de fonctions utilisent également les formes géométriques car elles peuvent transformer des listes de points ou des multipolygones en formes géométriques. La classe pour lire les années sur le site du cadastre est quant à elle utilisée par la couche control pour renvoyer une erreur à l'utilisateur s'il entre une année non disponible sur le site. Elle lit pour cela la page web de téléchargement des données et cherche les éléments du texte qui ont le format d'une date.

Pour finir, en vert, nous regrouperons les classes Segment et Rectangle au sein d'une classe mère figure géométrique. Ces classes auront pour attributs deux points de coordonnées les définissant totalement. En effet, un segment sera défini par ses deux extrémités et un rectangle sera défini par les coordonnées de deux de ses coins opposés. Pour ces deux classes nous redéfinieront la notion d'égalité pour que le segment [a,b] soit égal au segment [b,a] et pour que deux rectangles soient égaux si les points choisis pour les définir sont égaux à permutation près. La classe Rectangle implémentera également une fonction se_recoupent() qui retournera la réponse à la question : les rectangles se recoupent-t-ils ? Cette fonction nous sera très utile pour pouvoir limiter la complexité de nos calculs lorsque nous chercherons des territoires proches les uns des autres. En effet, au lieu de comparer les coordonnées directement, nous pourrons voir si les rectangles encadrant ces territoires se recoupent et nous ne regarderons les coordonnées que si cette condition est vérifiée.

Remarquons que le diagramme présenté ci-dessus n'est pas une présentation exhaustive des classes du programme. Seules les classes permettant d'implémenter des objets métier ont été représentées conformément à la consigne mais d'autres classes telles que des classes de DAO ou pour créer la base de données sont nécessaires.

4.1.2 La couche contrôle (controler layer) : récupère les entrées de l'utilisateur et lui renvoie la réponse

La couche contrôle layer se compose d'une unique classe Controle_layer qui est composée d'une unique méthode lancer qui va préciser une URL associée à une fonction qui communiquera avec l'utilisateur. L'utilisateur précise les paramètres, et cette classe vérifie les entrées et sorties saisies, ces paramètres sont ensuite passés en arguments à une fonction du service layer. Si les entrées ne sont pas bonnes (un identifiant de commune incorrect, une année pas présente sur le site) cela renvoie un message d'erreur à l'utilisateur. Dans le cas de l'année, cela lui renvoie un message détaillant une requête HTML à faire pour savoir quelles années sont disponibles.

Chaque requête a été définie dans le control layer à l'aide du package FastAPI. Il y a donc trois requêtes définies pour répondre aux différentes questions du sujet et trois requêtes additionnelles pour obtenir une aide (figure 7) qui renvoie toutes les commandes acceptées par l'API, obtenir les informations de la base (figure 8) ou encore obtenir les années disponibles sur le site du cadastre. Si une année est rajoutée sur le site, cela sera donc automatiquement affiché.

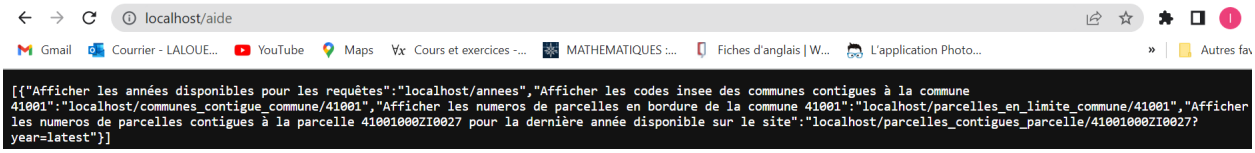

FIGURE 7 – **Illustration de app.get("/aide")**

Il y a également dans cette classe une fonction bdd, qui permet de voir quelle année et quel département est dans la base de données. Il ne peut y avoir qu'une année en même temps dans la base de données. En revanche, plusieurs départements peuvent être disponibles en même temps. Cela est illustré par la figure ci-dessous.

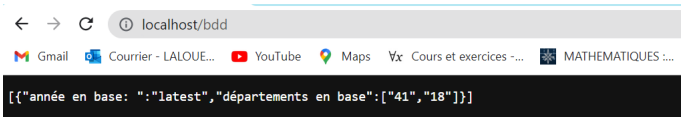

FIGURE 8 – **Illustration de la fonction bdd**

4.1.3 La couche service (Service Layer) : répond aux questions du problème

La couche service est le coeur de l'application. Elle communique avec la couche contrôle, pour récupérer les entrées de l'utilisateur, et avec la couche DAO pour avoir accès aux données. Elle va également utiliser les méthodes et classes détaillées dans la partie sur le diagramme des classes. En effet, le service a besoin de fonctions propres aux objets qui sont stockés dans la couche métier. Par exemple, à chaque fois qu'on devra manipuler une commune, le service va pouvoir utiliser les méthodes tel que __eq__ (qui teste l'égalité entre 2 segments). C'est dans la couche service que se trouve le code qui répond aux demandes de l'utilisateur. Cette couche sera plus amplement détaillée dans la partie 5.1 qui suit, car nous pensons que c'est la classe centrale.

4.1.4 La couche DAO (DAO Layer) : récupère les données demandées par la couche service

La couche DAO (pour Data Access Object, Objet d'accès à la donnée) permet de faire le lien entre l'application et la base de données, elle est donc en lien avec la couche service et la base de données. Elle permet de modifier la base de données en fonction des besoins de l'utilisateur (ajouter, supprimer ou modifier les données). Elle est utilisée pour remplir la base initialisée mais ce n'est pas elle qui la crée. La figure 5 illustre un exemple de données à écrire dans la base de données.

```
>>> a["table id / nom"][:4]
[['41273', 'VIEVY LE RAYE'], ['41080', 'FAVEROLLES SUR CHER'], ['41144', 'MONTEAUX'], ['41180', 'PONTLEVOY']
]
>>> a["table des communes adjacentes"][:4]
[['41001', '41103'], ['41001', '41102'], ['41001', '41124'], ['41001', '41236']]
>>> a["tableau id commune / id parcelle en bordure de la commune"][:4]
[['41001', '410010000A0024'], ['41001', '410010000A0023'], ['41001', '410010000A0028'], ['41001', '410010000
A0029']]
>>>
```

FIGURE 9 – **Exemple de données à écrire dans la base de données**

La couche DAO est composée de 7 classes. Toutes permettent d'insérer ou récupérer un certain type d'objet dans la base de données. Les fonctions de chacune de ces classes sont adaptées

au contexte des objets qu'elles manipulent. Par exemple, la classe de la DAO spécialisée dans les communes permet de récupérer les communes contigues à une commune stockée dans la base.

4.2 La couche client (client layer) : le web service qui récupère les données

La couche Client comprend une unique classe : téléchargement. Elle communique uniquement avec les couches Service Layer et Creation_BDD.
Cette classe comprend trois méthodes : telecharge_communes qui permet de télécharger les communes de la france entière, telecharge_parcelles_par_communes et telecharge_parcelles_par_departement qui permettent respectivement de télécharger les parcelles par communes et par département.
Chaque méthode vérifie avant toute opération si les fichiers sont déjà présents sur l'ordinateur. Si c'est le cas, elle ne les retéléchargent pas.
telecharge_commune prends en entrée un chemin vers un dossier où sera téléchargé le fichier et une année, qui sera latest (la dernière année) par défaut. Cette méthode télécharge toutes les communes de France, et permet également de dézipper le fichier en lecture seulement.
Les méthodes "telecharge_parcelle" permettent de télécharger les parcelles pour une commune choisie ou un département choisi. Elles prennent en entrée un chemin également, une année, et un identifiant de commune ou de département.
Toutes ces méthodes retournent des tableaux de dictionnaires python qui peuvent être utilisés par les autres fonctions du programme.

4.3 Les tests

Nous avons testé au maximum les classes avec des tests unitaires. Ces tests sont essentiels car ils permettent de vérifier que les parties du code source marchent indépendamment des autres. Ils sont très utiles pour vérifier que le code fonctionne bien, mais également car ils ont été codés avant certaines modifications. On a pu donc vérifier que la modification du code n'impacte pas le fonctionnement de l'API. En effet on a essayé de respecter au maximum la pratique du test driven developpement qui vise à écrire les tests avant de coder.

Nous avons respecté le format suivant pour les tests unitaires :
- GIVEN : qui représente l'initialisation, les inputs de notre fonction
- WHEN : Ce qui est à tester
- THEN : La vérification.

Un exemple d'un de nos tests est visible sur la figure suivante.

```python
class TestRectangle(TestCase):

    def test(self):
        #given
        rec1=Rectangle([2,2],[20,20])
        rec2=Rectangle([1,1],[10,10])

        #when
        d=rec1.se_recoupent(rec2)

        #Then
        self.assertTrue(d)


if __name__=="__main__":
    unittest.main()
```

FIGURE 10 – **Exemple de test unitaire**

5 Fonctionnement du code

5.1 Zoom sur le processus central de notre code : Service Layer

Le Service Layer est constitué de 3 classes : CommunesContiguesCommune (récupère les communes contigües à une commune donnée), Parcelles_contigues_parcelle_donnee (récupère les parcelles contiguës à une parcelle donnée) et ParcellesEnLimiteCommune (qui sert à récupérer les parcelles en limite d'une commune donnée).
Chaque classe est constituée de méthodes, mais celles-ci ne sont pas autonomes, elles ont besoin des autres couches pour fonctionner. Par exemple, quand l'utilisateur entre une parcelle précise, il y aura besoin de la DAO également pour communiquer avec la base de données, mais également du contrôle layer qui va renvoyer le résultat à l'utilisateur

Les différentes classes ne sont pas équivalentes en terme de difficulté à coder et de temps d'execution. La classe CommunesContiguesCommune vise à trouver les communes contigües à une commune sélectionnée par exemple. Dans le meilleur des cas, les communes sont déjà dans la base de données et une fonction de la DAO renverra le résultat. Dans le pire des cas, les données ne seront pas prêtes tout de suite et le programme devra tourner un petit peu plus longtemps en faisant appel à de nombreuses couches pour télécharger les bonnes données, trouver la réponse attendue par l'utilisateur et la lui renvoyer.

Pour illuster l'importance de cette couche nous avons choisi l'exemple de la requête 3. Cette requête vise à chercher les parcelles contigües à une parcelle donnée. La fonction dans la classe Parcelles_contigues_parcelle_donnee se nomme applique est prend en entrée l'identifiant de la parcelle et en option le chemin local où seront téléchargées les données et une année. Elle retourne une liste de parcelles adjacentes. Premièrement, la fonction va utiliser le département de la parcelle et l'année pour voir si la base de données stockée peut être utile pour trouver les parcelles contigües. Si les informations ne sont pas dans la base de données, les calculs seront plus longs. Si par contre, la base de données est utile, les calculs sont plus rapides. En effet, il n'est donc pas nécéssaire de télécharger les parcelles des communes contigues car les parcelles qui les entourent sont les seules qui peuvent être adjacentes à notre parcelle. Il suffit donc de les lire dans la base de données. Il peut se présenter 2 cas si la base est utile : soit la parcelle est en bordure de la commune soit elle est au milieu et donc dans ce dernier cas le programme est très rapide.
Pour exécuter cela, le programme va utiliser différentes classes telles que AnneeDAO() pour notament vérifier si la base est utilisable, Telechargement() pour télécharger les données dans le cas où la base de données n'est pas utilisable, FonctionMultipolygon(),etc.. Elle va donc utiliser différentes couches, qui elles mêmes dépendent d'autres couches.

5.2 Cas d'utilisations

Cette partie va présenter 2 cas d'utilisations : la requête 1 (quelles sont les communes ayant des parcelles limitrophes à une commune donnée ?) et la requête 3 (quelles sont les parcelles contiguës à une parcelle donnée ?). Grâce à la requête HTML aide, l'utilisateur aura la syntaxe exacte à écrire pour chaque requête et va pouvoir exécuter les requêtes qu'il souhaite.

Dans l'exemple qui suit, l'utilisateur veut connaître les communes contiguës à la commune 41001. L'année n'étant pas indiqué, c'est donc l'année par défaut qui est sélectionée : la dernière. Il va utiliser une requête HTML dans la bare URL et le résultat lui sera retourné. La capture d'écran ci-dessous illustre le résultat de cette requête.

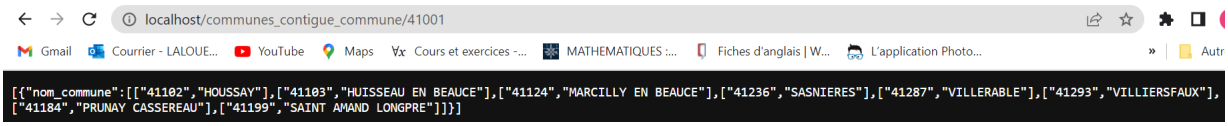

FIGURE 11 – **Résultat d'une requête de type 1**

Dans le deuxième exemple, l'utilsateur veut maintenant savoir les parcelles contigües à la parcelle 41001000ZI0027 pour le 01.10.2022. De la même manière l'utilisateur entre la requête dans la bare URL et le résultat apparaît.

FIGURE 12 – **Résultat d'une requête de type 3**

5.3 Piste d'amélioration

Notre code fonctionne et permet de répondre aux requêtes demandées. Cependant, avec plus de temps, nous aurions pu rajouter des fonctionnalités intéressantes et répondre à la question bonus. Seul le manque de temps nous a empêchés de les coder.

Premièrement, pour la base de données nous aurions aimé pouvoir coder une option qui permettrait de rajouter un département plutôt que de devoir re-téléchager la base en entier. Cela permettrait de faire évoluer la base au lieu de la recréer quand on souhaite la modifier.
Deuxièmement, on aurait pu permettre à l'administrateur de modifier la base depuis l'API. Cela peut être fait très facilement mais pour des raisons de sécurité, cela aurait nécessité un système de hashage des mots de passe, ce qui demande du temps et cela aurait également nécéssité d'assurer la confidentialité des requêtes HTML.
Dernièrement, on aurait pu également documenter tout le code, et pas seulement quelques classes si nous avions eu plus de temps.

6 Organisation et planning

6.1 Diagramme de Gantt

Le diagramme de Gantt suivant permet de voir l'avancée de notre projet. Il est disponible en capture d'écran ci-dessous, mais également via ce lien :
`https://docs.google.com/spreadsheets/d/15VbUVyspn55kSploETHSIAMp236r_j1m/edit?usp=sharing&ouid=117271999739385553876&rtpof=true&sd=true`

L'organisation de ce projet se découpe en 3 grandes phases.

La première correspond à l'étude préalable et à la conception générale du logiciel. C'est la phase la plus importante de ce projet. Elle permet à tous de comprendre le sujet, de savoir comment le programme va fonctioner, ce qu'il va pouvoir faire, et comment il sera fait. Premièrement, nous avons passé un temps important à la compréhension du contexte : sur les communes et parcelles (savoir comment on va définir le fait qu'elles soient contiguës ou en limite, comprendre comment les données sont sur le site où nous les récuperons). Nous avons donc dû définir et comprendre des notions essentielles pour la suite. Ensuite, il était essentiel que nous nous sentions tous à l'aise avec Git, car c'est l'outil que nous avons tous utilisé. Il était également essentiel de comprendre ce qu'était une API, sachant que le cours sur les API arrivait plus tard. Après, nous nous sommes concentrés sur l'analyse fonctionnelle de notre API, c'est à dire la réalisation des différents diagrammes. Nous avons donc dû réfléchir à quels diagrammes étaient pertinents.

La deuxième phase correspond à la phase de programmation. Cette partie correspond à la conception et à la production du code. C'est la phase la plus longue de notre projet. Nous avons commencé très tôt une partie de cette phase, notamment la création de la base de données car cela nous a permis de nous éclairer sur les diagrammes de cas et d'activité. Ensuite, durant nos réunions nous avons planifié notre répartition du travail : chacun s'occupait d'une couche et le Service Layer a été codé à la fin. La répartition s'est faite selon nos préférences et notre niveau en programmation. Cependant, lorsqu'une personne était bloquée, elle était bien sûr aidée par les autres. Nous passions également un moment pour accorder nos codes. Enfin, nous avons réalisé des test unitaires et documenté la couche Service_layer.

La dernière phase correspond à la réalisation du rapport final et de la soutenance. Nous avons pris en compte toutes les modifications demandées par notre tuteur, et avons modifié notre rapport d'analyse. Nous avons rédigé les différentes parties, relu, et mis en page le rapport. Enfin, après avoir rendu le rapport et le code, nous allons réfléchir à la soutenance : comment nous allons présenter notre projet ? Comment allons nous montrer les capacités de notre API ?

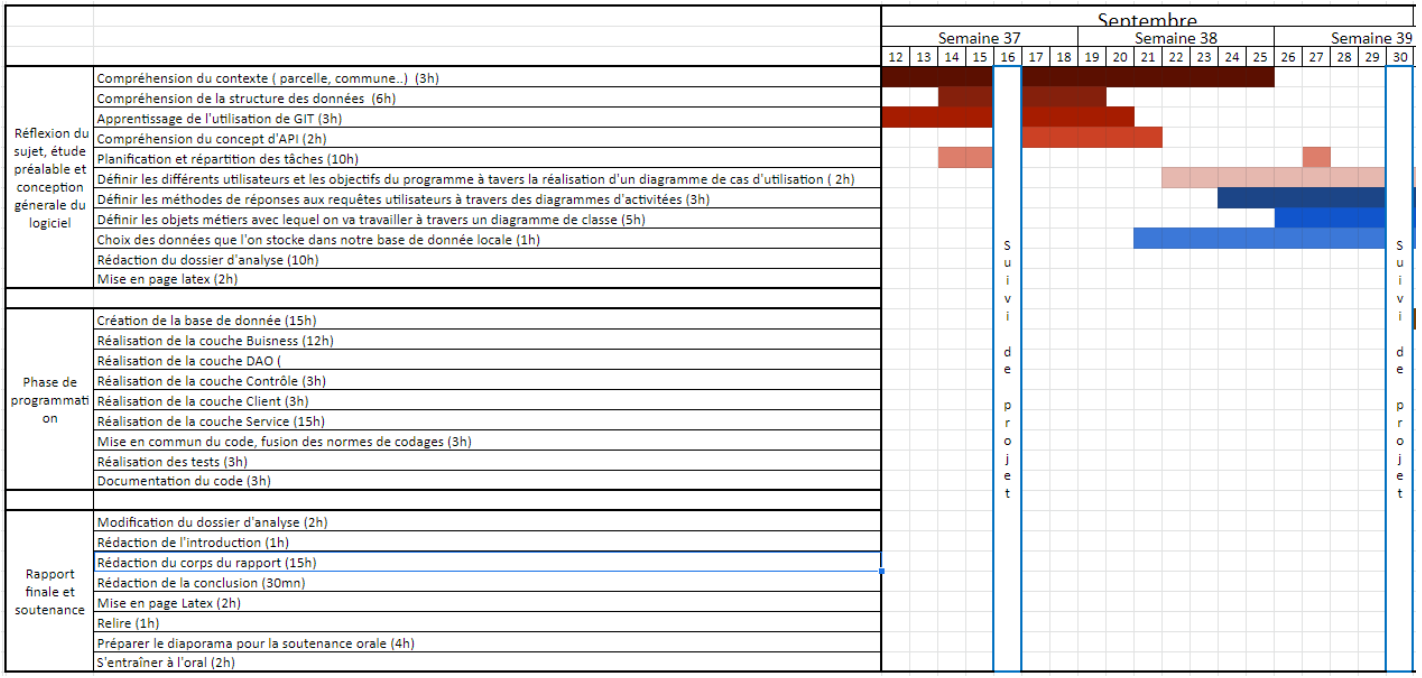

FIGURE 13 – **Diagramme de Gant partie 1**

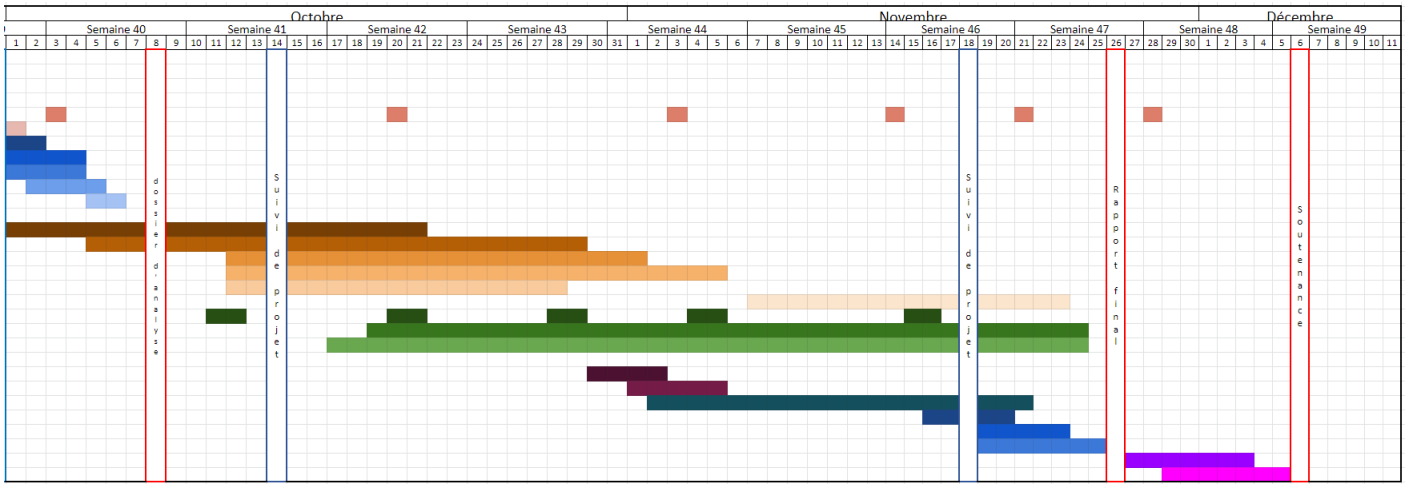

FIGURE 14 – **Diagramme de Gant partie 2**

6.2 Répartition des tâches

La répartition des tâches dans le groupe s'est faite de manière assez naturelle selon les points forts et les envies de chacun. Pour commencer ce projet dans la bonne direction, nous avons d'abord appréhendé et compris le sujet tous ensemble. Hadrien s'est ensuite occupé de nous aider à installer Git sur nos PC respectifs et nous a expliqué comment nous en servir. Nous avons ensuite créé notre projet sur Gitlab. Afin de maintenir une bonne communication dans le groupe, nous avons créé un groupe messenger et nous avons convenu de nous voir une fois par semaine pour faire le point.

Nous nous sommes répartis la réalisation des différents diagrammes qui permettent d'expliquer le fonctionnement de notre programme. Nous avons bien-sûr tous réfléchi sur chacun des diagrammes, mais c'est Marie qui s'est occupée de mettre au propre et finaliser les diagrammes de cas d'utilisation et d'activité, Tanguy qui s'est occupé du diagramme de Gantt, Lucas et Hadrien du diagramme de classes et Julien du diagramme de base de données.

Une fois les diagrammes réalisés et l'architecture du programme définie, nous avons commencé à réaliser le dossier d'analyse. Hadrien a rédigé l'introduction, Marie a rédigé la description des diagrammes de cas d'utilisation et d'activité, Lucas a écrit la description du diagramme de classe et Tanguy la répartition des tâches. Marie et Tanguy se sont ensuite occupés de la mise en forme du document Latex, puis tout le groupe a relu le dossier d'analyse et plus particulièrement Hadrien pour l'orthographe.

En parallèle, Lucas a commencé à réaliser le code de la base de donnée, et Hadrien a commencé à coder les objets métiers.

Ensuite, une fois le rapport d'analyse rendu nous avons commencé à coder. Nous nous sommes répartis le code dans un premier temps par couche. Julien devait s'occcuper de la couche DAO, Marie de la couche client, Hadrien de la couche contrôle, Tanguy de la couche Business Layer et Lucas de la création de la base de données.

Mais en pratique, l'organisation a changé et le code s'est réparti de manière différente, Marie a codé la couche Client pendant que Tanguy et Hadrien continuaient à coder la couche Buisness. Lucas a créé la base de données tout en codant la couche contrôle avec Julien qui s'est également occupé de la couche DAO.
Tout le groupe s'est occupé de réaliser des tests et de rédiger la documentation au fur et à mesure.

Une fois que ces différentes couches étaient presque terminées, Julien et Lucas ont continué à travailler sur la partie code en s'occupant de la couche Service pendant que Hadrien, Marie et Tanguy se sont occupés de la rédaction du rapport final. Marie a rédigé la partie requête SQL et fonctionnement du code, Hadrien a rédigé la partie Architecture du code et Tanguy à pris en compte les remarques sur le dossier d'analyse en améliorant le Gantt et en complétant la partie Répartition des tâches.

7 Notes personnelles

7.1 Marie Colin

7.1.1 Ressenti générale

Dans le cadre de ce projet informatique j'ai eu l'occasion d'apprendre beaucoup de chose sur le code Python mais également sur le travail en groupe dans le cadre d'un projet informatique comme celui-ci. Contrairement aux autres projets que nous avions faits, nous avons utilisé un nouvel outil Git et étions plus nombreux. Il était donc nécessaire de beaucoup plus communiquer entre nous pour ne pas créer de conflits que cela soit sur Git ou entre nous.

Je pense qu'il est également important de parler du sujet : Api cadastrale. Je ne voulais absolument pas ce sujet, tout simplement car il ne me parlait absolument pas. Les autres étaient plus facilement compréhensibles, et étaient pour la plupart des jeux. Il était donc plus dur de s'imaginer, dans notre cas, comment le programme pouvait être. Quand nous avions eu ce sujet, nous n'avions pas encore commencer les cours de compléments informatiques : nous n'avions donc pas encore vu les différentes couches. Je ne comprenais donc pas notre sujet et la manière dont on allait le réaliser. Je n'étais pas super enthousiaste à l'idée de commencer ce projet informatique. Après réflexion, notre sujet est très intéressant pour plus tard, et surement l'un des plus utiles : je n'ai jamais été confronté à un problème de stockage et de données lourdes. Ce sont des notions qui seront surement très utiles dans le futur. On a dû parfois trouver des compromis entre rapidité et utilité.

7.1.2 Le travail de groupe

Pour ce qui concerne l'entente dans le groupe : nous n'avons eu aucun problème, nous nous entendons assez bien, et l'ambiance était toujours sympathique. Nous avons eu plutôt de la chance, car ce n'est pas le cas dans tous les groupes en raison de la composition imposée des groupes. Nous avons essayé de nous voir souvent (une fois par semaine), en s'adaptant à nos contraintes respectives. On communique également beaucoup sur Messenger. Du fait de notre grand nombre (5), il était parfois difficile de se comprendre, et parfois nous étions contre-productifs : deux personnes codaient la même chose. Cela nous a créé un petit conflit sur Git, et nous a fait perdre un peu de temps. C'est ce qui nous est arrivé avec Hadrien.
Nous n'avions pas tous le même niveau dans le groupe, on s'est donc partagé le travail selon nos envies et nos qualités respectives. Nous avons pu avoir un cours sur Git par Hadrien, qui a été très utile pour la suite du projet, car le cours sur Git était le dernier enseigné. Julien et Lucas, ont pu nous aider également pour comprendre ce qui n'allait pas dans notre code.

7.1.3 Ma participation au projet

Pour la répartition du travail dans le cas de la programmation, je me sentais personnellement plus à l'aise avec la couche client, car j'avais mieux compris le TP qui la concernait. Les garçons ont tout de suite accepté. Même si c'était la couche que je devais coder, je pouvais compter sur le groupe quand j'avais des problèmes et notamment Lucas qui m'a bien aidé.
Bon comme d'habitude, les projets à l'ENSAI sont trop courts, et nous ne pouvons pas rendre exactement ce que nous souhaitons. Nous avons toujours des choses à rendre à l'ENSAI, ce qui fait que si on ne se donnait pas des tâches à faire toutes les semaines on est submergé. Mais il faut

admettre que ce projet s'est différencié des autres par les 3 jours d'immersions, qui étaient assez cool à avoir. Le fait que cela soit trop court, pénalise les moins forts du groupe, en tout cas c'est ce que j'ai ressentie. Je n'allais pas au même rythme que deux des garçons (Julien et Lucas), et après il semblait trop tard pour coder avec eux : j'allais les ralentir. Alors, il était plus simple pour moi, de comprendre leur code et d'écrire le rapport, que de coder de nouvelles parties telle que la couche service. J'ai participé donc activement à la réalisation du rapport et j'ai pu compter très fortement sur Lucas et Julien pour m'aider à comprendre des parties de code.

7.2 Hadrien Gôme

7.2.1 Ma participation au projet

Travaillant dans un service informatique de production avant de réussir le concours, j'avais eu une petite expérience avec Git. J'ai donc pu l'introduire auprès de mes camarades en leur expliquant le principe de fonctionnement et les commandes nécessaires à connaître lors d'un cours d'introduction. J'ai codé mais beaucoup moins que Julien et Lucas et beaucoup moins que ce que j'aurais aimé faire pour m'améliorer. J'ai participé à la rédaction du rapport, particulièrement à sa relecture, étant assez bon en orthographe et en grammaire.

7.2.2 Le projet avec le groupe

Le projet s'est très bien passé avec mon groupe. Je trouve que chacun a essayé d'apporter sa pierre à l'édifice. Je suis dans une situation familiale compliquée en ce début d'année : ma compagne suit une formation à l'autre bout de la France et je suis donc seul avec mon enfant d'1 an en semaine. J'ai apprécié que mes camarades soient tolérants avec moi sur mon implication dans le projet. J'ai essayé de faire le maximum mais je n'ai pas pu consacrer autant de temps que je l'aurais voulu au projet. J'aurais vraiment apprécié m'améliorer beaucoup plus en python.
Il n'y a pas eu de point de tension dans le groupe. Je pense que Marie a bien surveillé les différentes dates de rendu et a planifié notre travail et nos rendez-vous pour mettre en commun ce que nous avions fait. Cela a évité que nous nous retrouvions débordé à un moment quelconque du projet.

J'ai passé de bons moments à travailler avec ce groupe. Je pense que nous étions assez en phase sur notre manière de travailler et que nous étions complémentaires sur nos savoir-faire respectifs, ce qui fait que chaque membre du groupe avait sa place.

7.2.3 Les enseignements que j'en tire

Si je devais refaire ce projet, j'essaierais de plus travailler dès le départ : en effet, le semestre avançant, il devient de plus en plus difficile de dégager du temps pour travailler avec les partiels qui se rapprochent et le travail demandé dans les autres matières. Ma compagne étant partie début octobre, j'aurais peut-être dû travailler plus intensément au mois de septembre, quand j'avais plus de liberté le soir.
Même si je n'ai pas écrit autant de code que j'aurais souhaité, le résultat de notre travail me permettra, en le relisant et en l'étudiant à tête reposée, de progresser dans le développement. En effet, venant de la production informatique, j'aimerais après l'école travailler en temps que développeur, même si je n'ai pas beaucoup codé dans ma vie. Ce travail de groupe est donc inspirant pour mes perspectives d'avenir.

7.3 Lucas Laloue

7.3.1 impression générale

J'ai trouvé ce projet enrichissant en raison de la diversité des méthodes de programmation employées. Cela nous a donné l'occasion d'apprendre comment fonctionne une DAO, une API et de progresser en Python et en SQL. De plus, ce projet a amélioré nos méthodes de travail car coder à cinq sur le même programme nécessite plus de bonnes pratiques de programmation et oblige à beaucoup discuter, nous coordonner et réfléchir en amont. J'ai d'ailleurs eu l'occasion de participer activement à cette organisation en tant que chef de groupe. Bien que quelques ratés aient pu avoir lieu à cause d'une communication oubliée de temps en temps, nous avons globalement très bien réussi à travailler sans générer de conflits sur git et sans trop de redondance dans les travaux des membres du groupe. De plus, l'entente entre nous a été très bonne et j'ai eu la chance de travailler dans un groupe efficace et motivé. Nous nous sommes beaucoup entraidés sur les différentes tâches.

7.3.2 Participation individuelle au projet

Pour ma part, j'ai eu l'occasion de coder la création de la base de données. Cette couche récupère les données brutes téléchargées et dézippées par la couche client. Elle les traite ensuite pour préparer les tableaux de communes et parcelles tels qu'ils doivent être dans la base de données puis les transfère à la DAO pour qu'elle les insère. Ce travail était très minutieux car le traitement de tableaux en python nécessite beaucoup de rigueur dans le code (précision des indices, agencement des blocs de code) mais aussi de la réflexion car la taille des tableaux ne laisse pas de place à de la complexité algorithmique inutile. Environ la moitié du temps a d'ailleurs été consacré à de l'optimisation de code basée sur la théorie d'une part et sur la pratique d'autre part. Cette optimisation a plutôt bien fonctionné car la création d'un tableau d'associations de communes contiguës, la création d'un tableau contenant les parcelles en bordure de chaque commune et la création d'un tableau contenant les identifiants et les noms des communes prend environ 25 minutes en tout pour l'import de deux départements entiers en tenant compte des communes contiguës à ces départements. La toute première version du code prenait pourtant beaucoup plus de temps et n'importait pas les parcelles.

J'ai aussi participé au code de chaque couche, parfois pour une classe, parfois beaucoup plus. Cela m'a permis de comprendre le programme dans sa globalité et d'être assez efficace pour trouver les bugs et les corriger quand il y en avait. Une autre partie qui m'a demandé du travail est le service layer que j'ai codé avec Julien. Il permet de faire le lien entre toutes les fonctions du programme pour répondre à une requête transmise par l'utilisateur. Le code de cette couche a nécessité de réutiliser toutes les fonctions du code mais aussi d'en créer de nouvelles pour considérer les cas ou l'utilisateur entre une commune qui n'est pas dans la base. Le programme que nous avons écrit ensemble cherche alors sur le site du cadastre les bonnes données et les exploite pour répondre à la question malgré l'absence de la commune dans la base.

7.3.3 Bilan sur le travail en groupe et le résultat

La gestion de ce projet nous a amenés à nous répartir les tâches chaque semaine. Nous avons commencé par tous travailler sur le rapport d'analyse pour le premier rendu en nous répartissant les diagrammes. Ensuite, nous nous sommes répartis les différentes couches du programme. Enfin, nous nous sommes séparés en deux groupes à l'approche du rendu : le groupe qui devait finir le code était composé de Julien et moi-même. Le groupe qui devait rédiger le rapport écrit était constitué de Marie, Hadrien et Tanguy. Nous avons ensuite tous relu le rapport et apporté les modifications

nécessaires.

Je suis satisfait de la qualité de travail que nous avons produit. Je pense que le programme remplit le cahier des charges et va un peu plus loin sur certains aspects tels que la simplicité d'utilisation pour l'administrateur qui n'a qu'à exécuter une fonction pour initialiser la base ou encore les aides pour les utilisateurs de l'API leur permettant de trouver les années disponibles et de voir des exemples de requêtes en faisant localhost/annees et localhost/aide. Avec plus de temps, j'aurais aimé pouvoir me pencher sur la question bonus concernant les permis de construire qui n'a pas été traitée. Je ne regrette cependant pas de ne pas l'avoir faite étant donné le temps dont nous disposions car nous avons préféré assurer l'absence de bugs dans le programme en le testant sous tous les angles et en appliquant différents correctifs. Avec encore un peu plus de temps, nous aurions également pu mettre en place une possibilité pour l'administrateur d'initialiser la base en passant lui aussi par l'API avec une fonction spéciale lui demandant un mot de passe. Cela n'a pas été fait car il fallait mettre en place une façon sécurisée pour l'administrateur d'envoyer son mot de passe à l'API et nous avons préféré éviter de le faire trop vite pour ne pas risquer de générer des failles de sécurité.

Pour terminer, je suis satisfait de l'entente générale entre les membres du groupe qui est toujours restée bonne aussi bien dans les périodes calmes que dans les périodes de tension plus proches des deadlines. Tout le monde a fourni des efforts pour faire avancer le projet et j'ai pris beaucoup de plaisir à travailler avec les membres du groupe. Je pense que nous avons tous progressé en gestion de projet informatique et en programmation. Nous nous étions fixés pour objectif de faire une application fonctionnelle, répondant au cahier des charges et sans bugs et ces objectifs ont été remplis selon moi.

7.4 Tanguy Legrand

7.4.1 Ressenti générale

J'ai trouvé que ce projet était le plus intéressant parmi ceux que j'ai pu faire depuis le début de ma scolarité à l'ENSAI. Même si au départ j'étais assez déçu du sujet qui nous avait été attribué (c'était notre 7ème choix.). Le sujet était finalement très intéressant et nous a permis de nous confronter au développement de A à Z d'un programme répondant à une problématique précise.

J'ai eu la chance d'être dans un groupe très soudé et compréhensif. Tout le monde essayait d'apporter son aide, de faire avancer le projet et on était tous très compréhensif vis-à-vis du niveau de chacun et des contraintes d'emploi du temps. Travailler avec ce groupe était un plaisir et la bonne humeur était toujours présente.

Dans un premier temps, il a été difficile de savoir par où commencer et de visualiser comment notre programme s'articulerait, cette première partie était très floue et on avançait à tâtons. Nous avons quand même tous pu participer à cette étape qui fût très enrichissante et qui m'a permis de mieux comprendre comment appréhender le début de ce genre de projet.

7.4.2 Mon principal regret

J'attendais fortement la partie codage car j'aime beaucoup programmer et je comptais sur cette partie pour m'améliorer et rattraper ce "retard" que j'avais l'impression d'avoir depuis le projet informatique de l'année dernière où j'avais beaucoup moins pratiqué que les autres. Mais j'ai été très déçu de cette partie. On s'était réparti une couche chacun et je devais personnellement m'occuper

de la couche Buisness, mais le jour où l'on a commencé à coder ensemble, je me suis rendu compte qu'une grande partie de la couche Buisness avait déjà été codé par quelqu'un d'autre et qu'il ne me restait plus grand-chose à faire. N'étant pas le plus à l'aise en programmation du groupe, j'ai eu énormément de mal à suivre le rythme du reste du groupe et à continuer de coder avec eux, je me suis donc contenté d'essayer de comprendre le code qu'il produisait au fur et à mesure, d'apporter mon aide ponctuellement sur certaines classes ou pour la documentation et les tests et surtout d'avancer sur la partie rédaction du rapport avec Marie. Personnellement, le gros point noir de ce projet est donc que, malgré une réelle envie d'apprendre et de participer au code, je n'ai pas pu suivre le rythme du groupe et n'ai donc pas pu participer et progresser autant que je l'aurais voulu au niveau de la programmation pour ne pas ralentir le groupe.

7.4.3 Ma participation au projet

Même si je n'ai pas pu apporter au groupe l'aide que j'aurais voulu au niveau du code, j'ai quand même pu beaucoup aider sur la partie d'étude préalable et de conception générale du logiciel ainsi que sur la rédaction du rapport et la mise en page latex. J'ai bien sûr aussi participé à la bonne humeur du groupe.

7.4.4 Bilan et enseignement

Je garderais un très bon souvenir de ce projet et de mon groupe, je n'ai pas du tout vu ce projet comme une contrainte, mais plus comme un moyen d'apprendre. Je suis maintenant à l'aise avec l'utilisation de GIT et la gestion de projet informatique à plusieurs, j'ai aussi beaucoup appris sur comment appréhender et commencer la conception d'un logiciel. Et j'ai bien sûr aussi développé de nouvelles compétences de programmation.

L'enseignement principal que je tire de ce projet est de me mettre à commencer à coder le plus tôt possible par rapport au reste du groupe, car une fois que l'on a pris du retard, il devient très difficile de suivre et de continuer de participer à la réalisation du code ce qui gâche un peu le projet.

7.5 Julien Sawadogo

7.5.1 Participation au projet

Arrivé un peu en retard à l'ENSAI, j'ai trouvé que mon groupe avait un peu avancé avec la rédaction du rapport mais pas encore avec le code. Ainsi j'ai eu à compléter le rapport avec la rédaction de la partie concernant la base de données. Pour la suite du projet, étant donné que je suis beaucoup plus à l'aise avec la programmation, j'ai le plus participé à l'écriture des différents codes pour les différentes couches de notre API. J'ai précisement codé la couche DAO, la couche Service et la couche contrôleur. J'ai codé moins que Lucas mais néanmoins j'ai eu assez a codé, pas autant que je voulais. Cela m'a permis de bien maitriser les cours de complément informatique que j'ai presque raté du faite de mon retard.

7.5.2 Le travail avec le groupe

Le travail en groupe s'est très bien passé, chacun a apporter son plus dans la réalisation de l'API et la rédaction du rapport. Chacun a essayé de s'appuyer sur ses points forts pour faire avancer le projet. J'ai beaucoup aimé travaillé dans ce groupe. En effet, alors que je n'étais pas encore à l'ENSAI, les membres de mon groupe m'ont contacté et on a eu à travailler en ligne où ils m'ont

expliqué un peu le projet.
De plus, à mon arrivée, je n'avais pas fait le cours de complément informatique et de remise à niveau informatique, les membres de mon groupe ont accepté de me faire confiance quand j'ai decidé de coder la partie DAO. En plus ils étaient tous là pour m'expliquer les trucs que je ne comprenait pas et aussi à l'écoute de mes propositions concernant les choses qu'ils avaient fait durant mon absence. J'ai beaucoup aimé travailler avec Lucas et Marie. Lucas, le chef de groupe, a su bien organiser les différentes tâches pour chaque membre du groupe et était toujours là quand quelqu'un avait besoin de lui. Il a été un bon chef de groupe en motivant tout le monde et en étant l'exemple à suivre par son travail acharné. Quant à Marie, elle était le moteur du groupe. Elle programmait les séances en ligne et en présentiel du groupe et a fait le gros du travail de rédaction du rapport.

7.5.3 Enseignements tirés

J'ai beaucoup aimé ce projet car j'ai appris beaucoup de choses en python et aussi pu mettre en pratique mes connaissances en programmation python. Aussi, ce projet est en adéquation avec le cours de complément informatique qui permet de mettre en pratique le cours. Si je devais reprendre ce projet, j'aurais aimé coder encore plus mais surtout écrire plus le rapport qui permet d'expliquer en des mots simples l'utilisation, le fonctionnement et la conception de notre API. J'aurai voulu qu'on réduise le nombre de personnes par groupe et rallonger la durée du projet afin que chaque étudiant puisse bien coder et beaucoup écrire le rapport d'analyse. Les 3 jours d'immersions ont été nécessaires et cool pour la réalisation de ce projet.