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

Projet informatique

Etudiants 2A

API cadastrale

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

Tuteur :
Mathe Thierry
Coach :
Rémi Pepin

2022 - 2023

14 octobre 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, l'activité français met 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. **passage trop rapide.**

Notre API doit tout d'abord permettre de constituer une base de données pour enregistrer les parcelles et communes contiguës. In fine, en s'appuyant sur cette base, elle doit pouvoir sortir les PC à cheval sur plusieurs communes. Nous nous appuyons pour construire notre base sur les fichiers GeoJSON du site https ://cadastre.data.gouv.fr/datasets/cadastre-etalab.

2 Analyse fonctionnelle

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 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 ?
- Y a-t-il un permis de construire sur deux communes ?

API aide à répondre à question pour répondre on utilise les 3 premières requêtes. Pour résoudre on use API

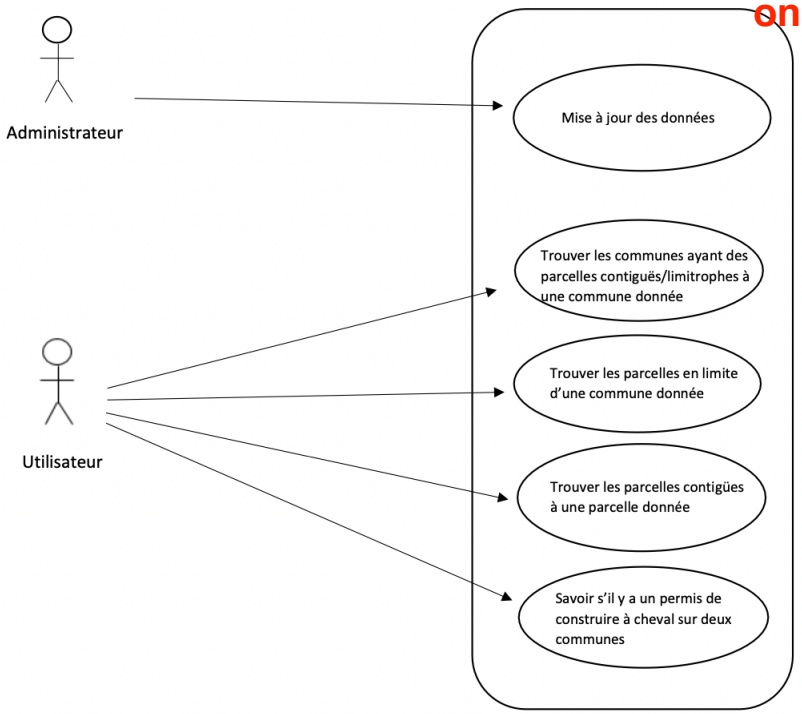


Diagramme des cas d'utilisation.

Les diagrammes d'activité qui vont suivre vont permettre de détailler les différentes étapes possibles du programme. La partie en vert représente les actions de l'administrateur tandis que la partie en bleu représente les actions de l'utilisateur. Les 4 diagrammes qui suivent ont la première partie identique. Cette partie correspond à la mise à jour des données.

Premièrement, l'administrateur choisit la zone géographique couverte par l'API. Le programme téléchargera ensuite les communes selon ces choix. Ensuite, le programme permet de trouver les communes adjacentes à chaque autre commune en comparant les coordonnées géographiques de celles-ci. Une commune est dite adjacente à une autre si elles partagent un segment en commun. Nous avons choisi de stocker les communes adjacentes dans la base de données car nous en avons besoin dans les différentes requêtes. Cela permet donc de gagner du temps. La base de données est détaillée plus précisément dans la partie 2.2. Ensuite l'utilisateur choisit l'année des données qu'il veut étudier. Ce choix de l'utilisateur n'aura pas d'impact sur la création de la base de donnée.

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.

L'utilisateur intervient après ce processus de création de base de données pour choisir une des différentes reqûete. Lorsque la base est déjà existante, les requêtes peuvent être executées directement sans avoir besoin de reconstruire la base.

Il est à noter que les résultats des requêtes devront pouvoir rester accessibles à l'utilisateur pour un certain temps afin que celui ci puisse lancer les calculs qui peuvent être longs et revenir pour voir le résultat quelque temps plus tard. Si certaines requêtes nécéssitent les résultats d'autres requêtes, ces résultats intermédiaires seront stockés dans des variables pour être réutilisés.

Le diagramme d'activité qui suit illustre le déroulement du programme lorsque l'utilisateur choisit la requête 1. Dans ce cas-là, il veut trouver les communes ayant des parcelles limitrophes à une commune qu'il aura choisi.

Le programme va demander à l'utilisateur la commune qu'il veut étudier, et celle-ci représentera l'entrée. Le programme s'assuera au préalable que la commune existe, ce qui permet de renvoyer directement une erreur à l'utilisateur si cette condition n'est pas vérifiée.

Ensuite, le programme doit seulement lire la base de données qui est déjà construite afin de trouver les communes adjacentes à celle qui a été sélectionnée. Le résultat est retourné à l'utilisateur sous forme de sortie Json.

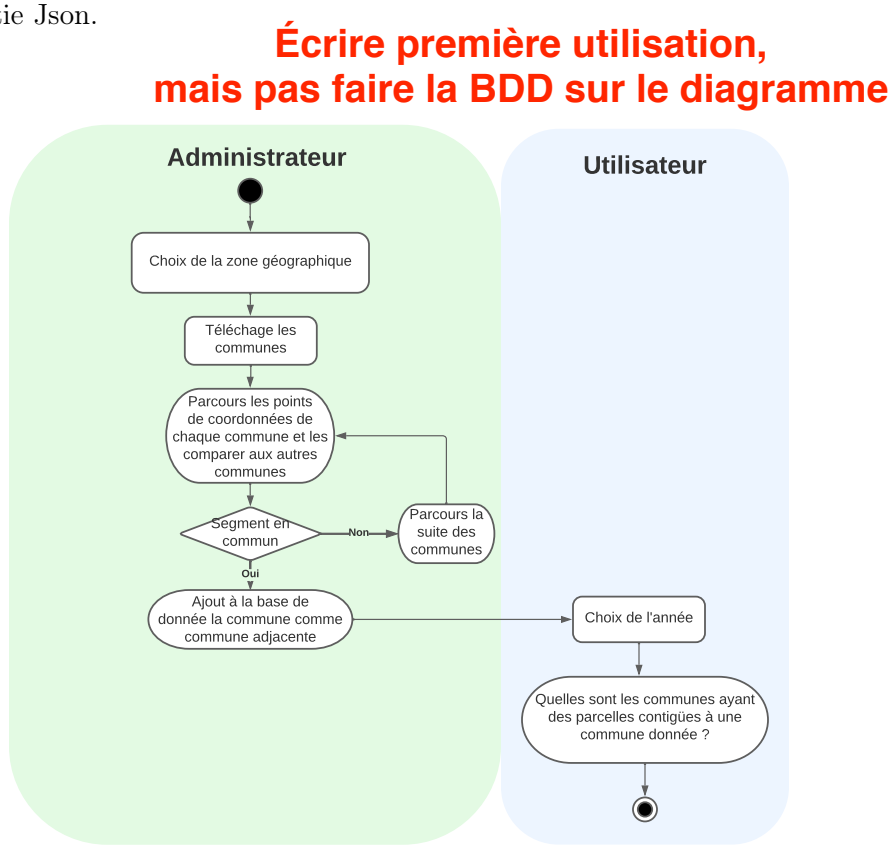


Diagramme d'activité requête 1.

Le diagramme d'activité qui suit illustre le déroulement du programme lorsque l'utilisateur choisit la requête 2. Dans ce cas-là, il veut trouver les parcelles en limite d'une commune qu'il aura choisi.

L'utilisateur fournira en entrée l'identifiant de la commune pour laquelle il veut connaitre les parcelles en limites. Le programme s'assuera également au préalable que la commune existe, 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 va comparer 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 comparé et retourne donc le résultat dans un fichier Json.

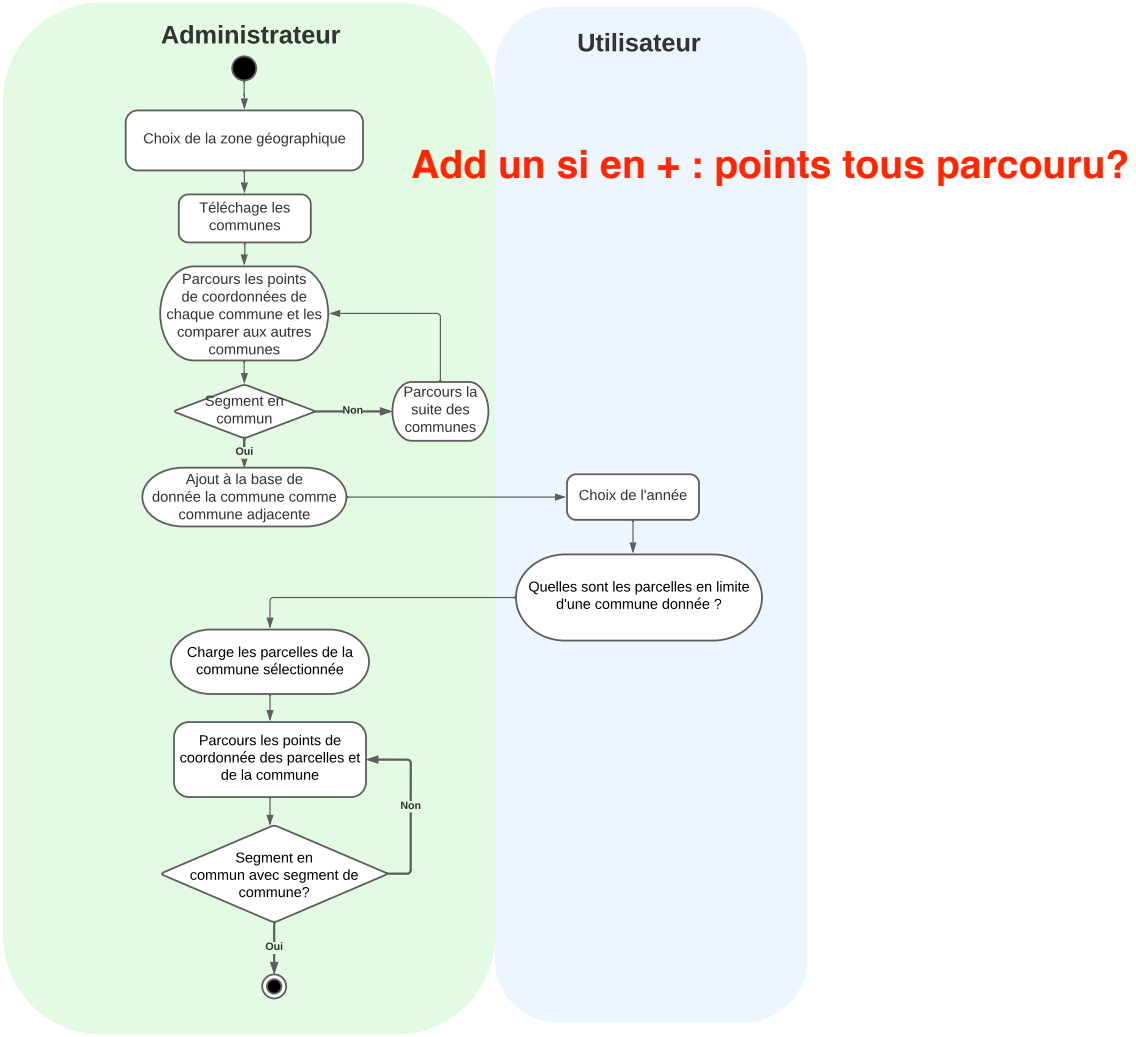


Diagramme d'activité requête 2.

Le diagramme d'activité qui suit va illustrer le déroulement du programme pour la requête 3. Dans ce cas-là, l'utilisateur veut trouver les parcelles contigües à une parcelle qu'il aura choisi.
Le programme va demander à l'utilisateur la parcelle dont il cherche les parcelles contigües, et celle-ci représentera l'entrée. De la même manière, le programme s'assure qu'il n'y a pas d'erreur.

Avant de charger les données, le programme va vérifier si la parcelle sélectionnée est en limite d'une commune. Autrement dit, l'exécution de cette requête va aussi passer 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 on peut récupérer la commune à laquelle elle appartient, et donc le programme pourra exécuter la requête 2. Cette stratégie permet de ne pas parcourir trop de points inutilement. De plus, si l'utilisateur a déjà exécuté la requête 2 au préalable, le résultat 2 est déjà connu. Si la parcelle est en limite de la commune, le programme chargera les parcelles de sa commune et des communes adjacentes, sinon la parcelle n'est pas en limite et donc le programme chargera uniquement les parcelles de la commune. Ensuite, de la même manière que dans les requêtes précédentes, le programme regardera s'il y a des segments en commun et les stockera. Une fois que tous les segments sont parcourus, le programme retournera le résultat dans un Json.

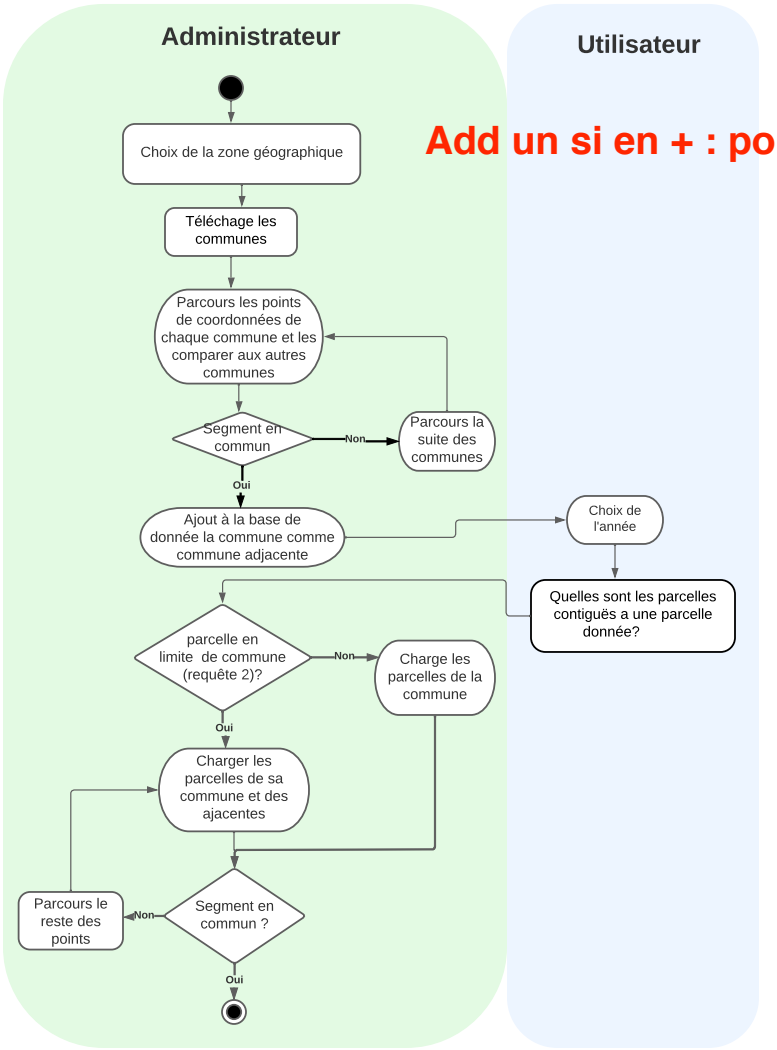


Diagramme d'activité requête 3.

Le dernier diagramme d'activité illustre le déroulement du programme lorsque l'utilisateur choisit la requête 4. Dans ce cas-là, il veut savoir s'il y a un permis de construire à cheval sur deux communes. L'utilisateur devra fournir l'identifiant d'un permis de construire au programme. Celui-ci vérifiera si ce permis de constuire est répertorié dans d'autres communes et si oui, il donnera le nombre de communes sur lesquelles il se trouve.

Pour cette requête, le programme va commencer par trouver une parcelle sur laquelle se trouve le permis de constuire. Il fera ensuite appel à la seconde requête afin de trouver les parcelles adjacentes. Sur ces parcelles adjacentes, il effectuera un balayage des permis de constuire pour voir si le même permis que celui ayant été fourni par l'utilisateur apparait d'autres fois. Si c'est le cas, le programme regardera dans quelle(s) commune(s) il se trouve pour pouvoir enfin donner le nombre de communes différentes ayant répertorié ce même permis de construire. Le résultat sera retourné au format Json.

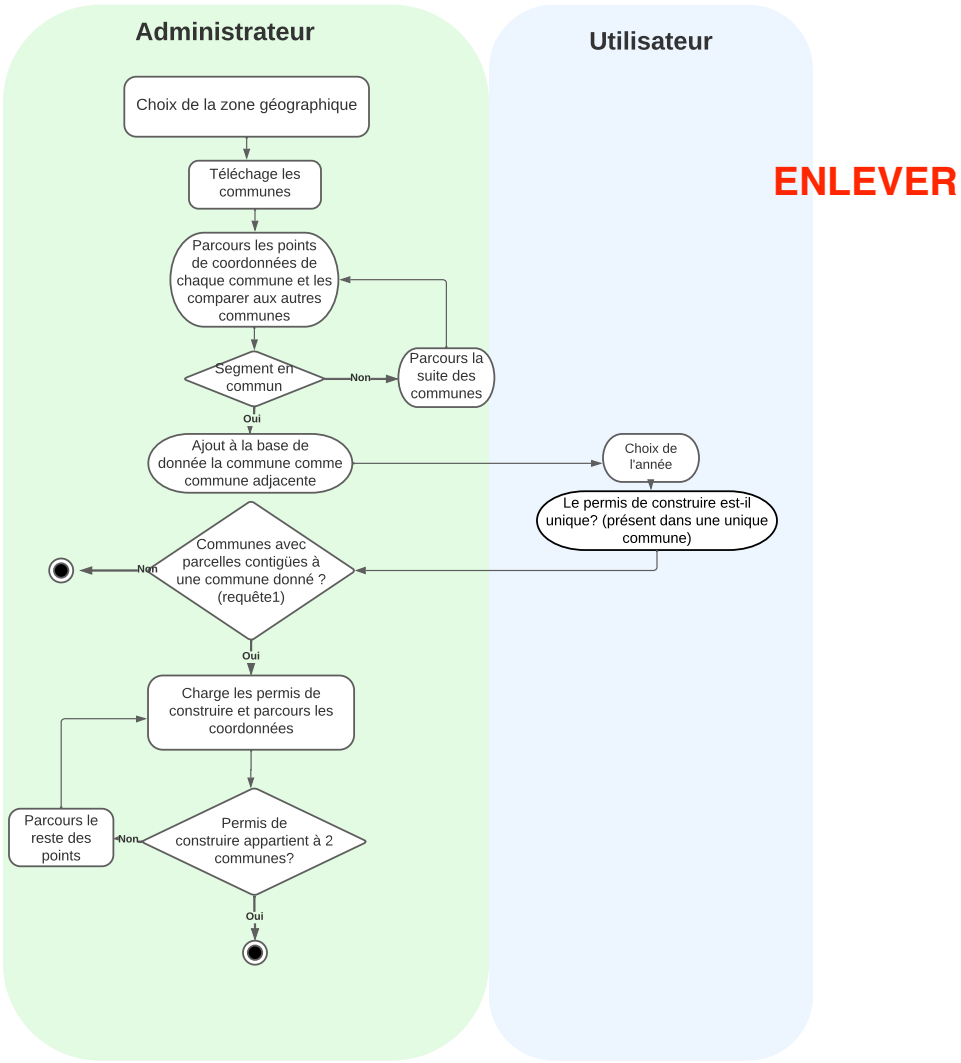
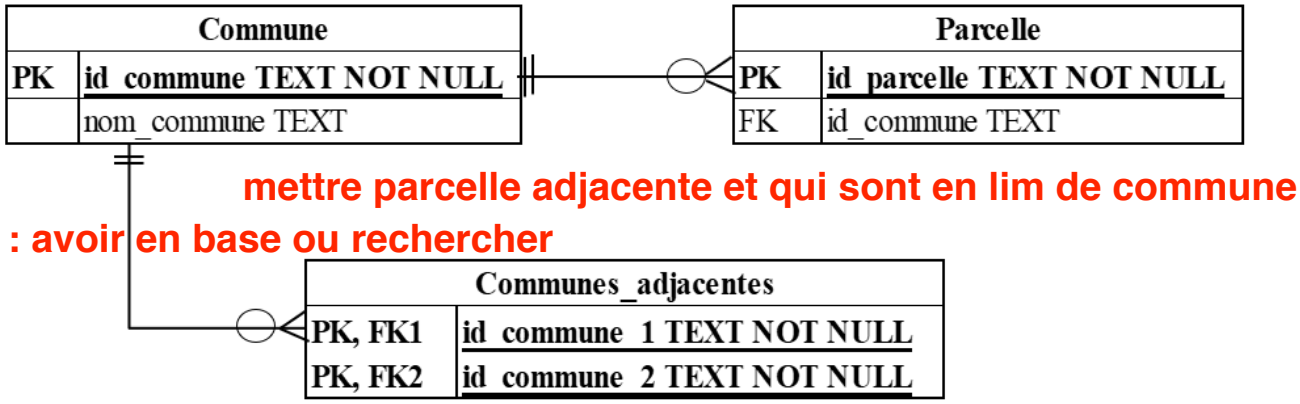


Diagramme d'activité requête 4.

2.2 La base de donnée

Une base de données est un gros ensemble structuré de données, mémorisé sur un support permanent. Elle permet le stockage d'informations qui sont couramment utilisées. Les diagrammes précédents montrent qu'un utilisateur aura pour réponse à sa requête un ensemble de communes ou un ensemble de parcelles ; ce qui nous amène à créer une base de données des communes (respectivement de parcelles) qui contiendra l'ensemble des communes (respectivement des parcelles) pour une année de référence choisie. Une autre base de données qui contiendra les communes qui sont contiguës sera créée afin de facilement répondre à la question des communes limitrophes à une commune donnée et du fait que la question de commune contiguë revient beaucoup (exemple pour déterminer les parcelles contiguës à une parcelle donnée, si cette parcelle est en limite d'une commune alors il faut aussi charger les parcelles des communes qui sont contiguës à la commune de la parcelle). Ainsi, le diagramme de données de notre API se présente sous la forme suivante :

stocker toutes les parcelles => prends bcp de place



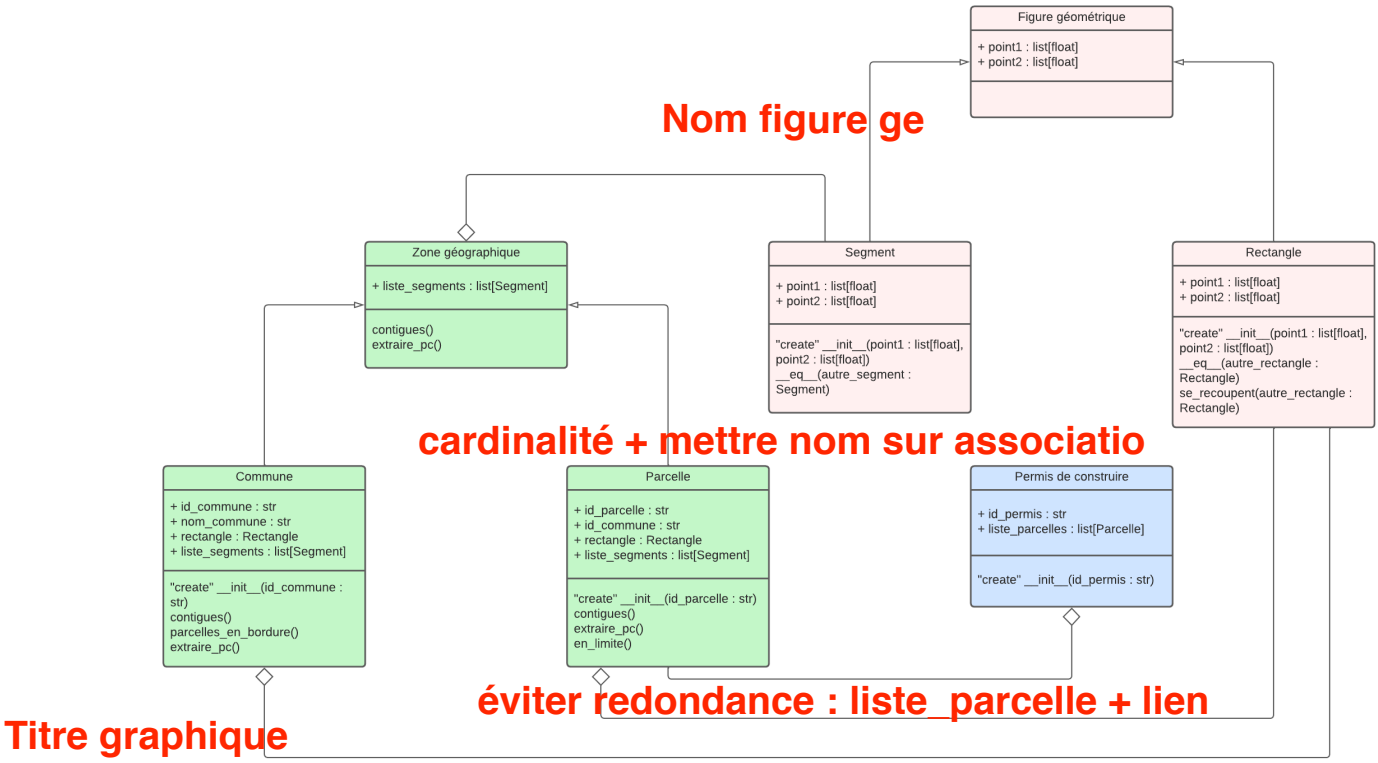
mettre parcelle adjacente et qui sont en lim de commune

Idée : avoir en base ou rechercher

La base de données Commune nous permettra d'enregistrer les identifiants et les noms des différentes communes pour une année et une zone géographique données. Cette base sera construite grâce à une DAO qui téléchargera les fichiers GeoJSON sur le site https ://cadastre.data.gouv.fr/datasets/cadastre-etalab. La base Parcelle sera construite de la même façon et elle contiendra le numéro d'identifiant de la parcelle et le numéro d'identifiant de la commune dans laquelle se trouve la parcelle. Quant à la base qui contiendra les différentes communes adjacentes, elle sera construite avec le fichier GeoJSON des communes et grâces aux coordonnées géographiques des différentes communes. Dans cette base, chaque commune aura p lignes qui correspondent aux nombre de communes qui lui sont adjacentes déterminer grâce à une DAO. Il faut noter que pour réduire la taille de la base, nous tiendrons compte du fait que si la commune A est contiguë à B alors la commune B est contiguë à A.

2.3 Conception

2.3.1 Diagramme des classes objets métier



2.3.2 Description des classes et de leurs méthodes

Le but de notre architecture de code est de regrouper les classes selon leur nature à l'aide de l'hérédité. Ainsi, trois types de classes apparaissent avec les zones géographiques, les figures géométriques et les permis de construire. Chaque classe implémente des méthodes et fonctions propres au concept qu'elle représente pour en faciliter la manipulation et pour rendre le code plus lisible.

En vert, nous avons implémenté les classes zones géographiques. La classe Zone géographique est une classe abstraite ayant les méthodes abstraites contigue() et extraire_pc. En effet, Commune et Parcelle qui sont ses deux classes filles auront toutes les deux besoin d'une méthode contigue() qui ne fera peut être pas exactement la même chose mais dont une grande partie du code sera identique. En effet, nous devrons retrouver les communes contigues à une commune donnée mais aussi retrouver les parcelles contigues à une parcelle donnée ce qui est exactement la même chose en terme de code. D'autre part, nous aurons besoins d'extraire les permis de constuire appartenant à une commune donnée et à une parcelle donnée ce qui justifie la présence des méthodes extraire_pc() dans chaque classe. La classe Parcelle disposera en plus de cela d'une fonction en_limite() qui retournera un booléen permettant de répondre à la question : Cette parcelle se trouve-t-elle en limite de commune ? La classe Commune disposera quant à elle d'une fonction parcelles_en_bordure() qui retournera la liste des parcelles en bordure de la commune.

En bleu, la classe permis de construire implémentera la méthode contigue() qui permettra de

connaitre la liste des permis de constuire contigus à un permis de constuire donné afin de répondre aux questions optionnelles du sujet.

Pour finir, en rose, 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 seront nécessaires.

3 Organisation et planning

3.1 Description 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. Adrien s'est ensuite occupé de tous 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.

Nous nous sommes répartis la réalisation des différents diagrammes qui permettront 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 et Lucas du diagramme de Classes.

Une fois les diagrammes réalisés et l'architecture du programme définie, Nous avons commencé à réaliser le dossier d'analyse. Adrien 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 particuliérement Adrien pour l'orthographe.

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

Julien n'ayant pu arriver en France que fin septembre, il a été plus difficile pour lui de s'impliquer dans ce début de projet. Il a en revanche pu suivre le déroulé du projet dès son arrivée et a notamment réalisé le diagramme de base de données. Il a également participé et aidé Lucas et Adrien à coder.

3.2 Diagramme de Gantt

Le diagramme de Gannt suivant permet de voir l'avancée que nous prévoyons. Il est disponible en capture d'écran ci-dessous, mais également via ce lien :
`https://docs.google.com/spreadsheets/d/16AqELGveprqY_fO-S1e2oJz8LSEnJkeu/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 fonctionner, ce qu'il va pouvoir faire, et comment il sera fait. Celle-ci est actuellement presque terminé : il ne reste que la partie descriptive : rédiger le rapport en entier et également le mettre en page sur latex.

La deuxième phase correspond à la réflexion sur le sujet. Cette partie correspond à la conception et à la production du code. C'est la phase la plus longue de notre projet. Nous avons déjà commencé 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.

La dernière phase correspond à la réalisation du rapport final et de la soutenance.

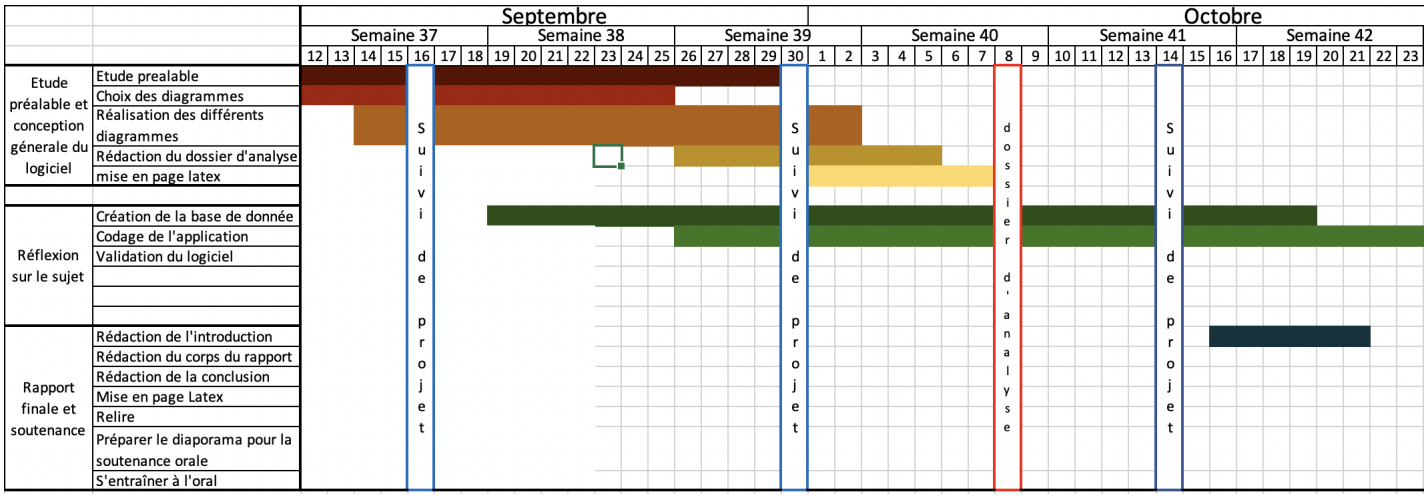


Diagramme de Gant partie 1

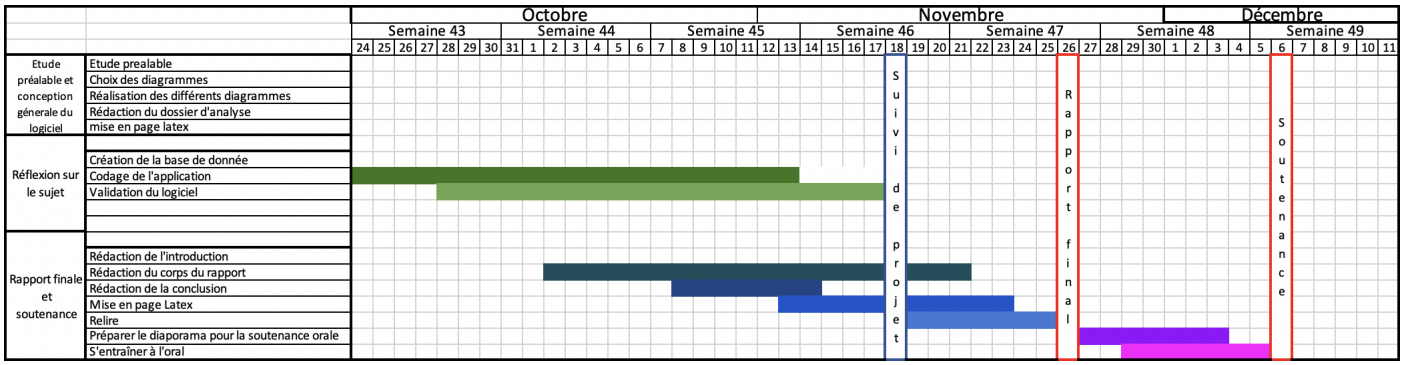


Diagramme de Gant partie 2