

RAPPORT DE STAGE

- UNIMA -

Giraud Victor



- UNIMA -

Avril à Juillet 2024

Étudiant : Giraud Victor

Bouquet Claire - Clément Poirier

Licence Professionnel SIG - Géomatique

REMERCIEMENT.....	5
1. INTRODUCTION.....	6
1.1 Présentation de l'UNIMA.....	6
1.1.1 Contexte Historique.....	6
1.1.2 Présentation de la structure.....	6
1.1.3 Présentation du pôle SIG.....	7
1.2 Présentation des missions du Stage.....	8
1.2.1 Mission Principale - Application Métier.....	8
1.2.2 Mission Secondaire - Logiciel Administratif de gestic financer des propriétaires.....	9
2. RÉCOLEMENT DES TRAVAUX D'ENTRETIEN DES MARAIS.....	10
2.1 Problématique.....	10
2.1.1 Présentation du problème.....	10
2.1.2 Etat des lieux.....	10
2.2 Création d'un cahier des charges.....	11
2.2.1 Critères de la solutions.....	11
2.2.1 Solutions envisagées.....	12
2.3 La solution développée.....	15
2.3.1 Elaboration du processus de réflexion.....	15
2.3.2 Schématisation du fonctionnement de la solution.....	15
2.3.3 Explication du fonctionnement de la solution.....	18
A. Le Bureau d'Études.....	18
B. Le conducteur de travaux.....	20
C. Les pelleteurs.....	21
D. Base de données.....	22
2.3.4 Elaboration d'un planning de travail.....	23
2.4 Les bases de données.....	23
2.4.1 La base de données UNIMA.....	23
2.4.2 Le schéma Régie.....	24
2.4.3 Le schéma d'historisation.....	24
2.4.4 La base de données LizMap.....	24
2.4.5 La Base de données test.....	24
2.5 Le Plugin QGIS.....	27
2.5.1 Bureau d'Études : Crédit du projet.....	27
A. L'interface utilisateur.....	27
B. Les problématiques.....	28
a. L'utilisation d'un référentiel.....	28
b. Crédit de la couche d'observation.....	29

c. L'ajout d'un fond de plan.....	30
d. Autres options réalisées par le plugin.....	30
e. Les étapes de vérifications.....	30
f. Programme complet.....	30
g. Les styles.....	31
2.5.2 Bureau d'Études : Export du projet vers le Cloud.....	34
A. L'interface utilisateur.....	34
B. Les problématiques.....	35
a. A quoi sert cette partie du plugin ?.....	35
b. Exporter les couches et les entités dans le nouveau projet.....	35
c. Création d'un fond de plan pour le cloud.....	36
d. Créer un projet dans un sous dossier.....	37
e. Les étapes de vérifications.....	38
f. Utilisation du plugin QFieldSync.....	38
g. Le programme complet.....	39
2.5.3 Conducteurs de travaux : Gestion de projet.....	40
A. L'interface utilisateur.....	40
B. Les problématiques.....	41
a. Télécharger un projet depuis le cloud.....	41
b. Se connecter à un compte QFieldCloud.....	42
c. Téléchargement d'un projet QFieldCloud pas encore présent sur la machine.....	43
d. Ouvrir un projet QFieldCloud déjà présent sur la machine.....	43
e. Mettre à jour un projet QFieldCloud stocké sur le cloud.....	43
f. Le programme complet.....	45
2.5.4 Conducteurs de travaux : Gestions des données.....	46
A. L'interface utilisateur.....	46
B. Les problématiques.....	47
a. Récupérer un projet cloud modifié par les pelleteurs.....	47
b. Exporter les données linéaire vers le schéma régie_travaux de la BD UNIMA.....	48
c. Récupérations des informations de la table de linéaire.....	48
d. Vérification de l'existence d'une entité.....	49
e. Exporter les données ponctuelles vers le schéma régie_travaux de la BD UNIMA.....	50
f. Export des potentielles photo vers le serveurs de stockage de l'Unima....	50
g. Le programme complet.....	50
2.6 Paramétrage des tablettes.....	51
2.7 Initialisation des CronTable.....	51

2.7.1 Cron Table et Fichier Bash.....	52
A. Fichier Bash.....	52
Fonctionnement.....	52
B. Les CronTab.....	52
Fonctionnement.....	52
2.7.2 Transfert des données du schéma de la régie travaux vers le schéma d'historisation.....	53
A. Contenu du fichier Bash.....	53
B. Explication de la requête SQL.....	54
C. Comportement de la Requête.....	56
D. Contenu du CronTab.....	56
2.7.3 Transfert des données du schéma d'historisation vers la BD LizMap.....	56
E. Contenu du fichier Bash.....	56
F. Explication de la requête SQL.....	57
G. Comportement de la Requête.....	59
H. Contenu du CronTab.....	59
2.8 Mise en place du LizMap.....	59
2.8.1 Contenu attendu du WebSig.....	59
2.8.2 Importation des données.....	60
A. Requêtes des ponctuelles.....	60
B. Requêtes pour le linéaire liées aux curage.....	61
C. Requêtes pour le linéaires des travaux annexes.....	61
2.8.3 Mise en forme des données.....	62
A. La symbologie.....	62
B. Les infobulles QGIS.....	63
2.8.3 Configuration du projet LizMap.....	63
A. Les options de la cartes.....	63
B. Couches.....	64
C. Fond de cartes.....	65
D. Tables attributaire.....	65
E. Mise en page.....	65
F. Localiser par couche.....	66
2.8.4 Présentation du LizMap.....	66
3. SUIVI ADMINISTRATIF DES ASSOCIATIONS SYNDICALE DE MARAIS.....	69
3.1 Problématique.....	69
3.1.1 Présentation du problème.....	69
3.1.2 Etat des lieux.....	69
3.1.3 Étapes de fonctionnement de la solution actuelle.....	70
3.2 Créations d'un cahier des charges.....	70

3.2.1 Critères de la solutions.....	70
Axes d'amélioration et nouvelles fonctionnalités souhaitées :.....	70
Fonctionnalités existantes à conserver :.....	70
3.2.1 Solutions envisagées.....	71
3.3 La solution développée.....	72
3.3.1 Elaboration du processus de réflexion.....	72
3.3.2 Schématisation du fonctionnement de la solution.....	73
3.3.3 Elaboration d'un planning de travail.....	75
3.4 Le Framework Django.....	76
3.4.1 Le principe de base.....	77
3.4.2 Les fichiers primordiaux.....	78
3.4.3 Les fichiers d'application externe.....	79
3.4.3 La page d'administration.....	80
3.5 Les bases de données.....	80
3.4.1 La base de données.....	80
3.4.2 Les models Django.....	83
3.5 L'application Django.....	84
3.5.1 L'identification.....	84
A. Logique.....	84
B. L'interface utilisateur.....	85
C. La page Web.....	85
a. Fonctionnement de Django.....	85
b. Dans l'application.....	86
D. Forms.py.....	86
E. Views.py.....	86
F.Urls.py.....	87
3.5.2 La gestion des propriétaires.....	88
A. Logique.....	88
B. L'interface utilisateur.....	89
C. Les pages Web.....	89
D. Models.py.....	89
a. Makemigrations & Migrate.....	89
b. Création d'un fichier migrations vide.....	90
c. Mise à jour du fichier Models.py.....	90
d. Tables utilisées.....	90
E. Forms.py.....	91
F. Views.py.....	92
Vue proprietaire.....	92
Vue create_proprietaire.....	92

Vue update_proprietaire.....	93
3.5.2 La gestion des parcelles et des Association Syndicales.....	93
3.5.3 La gestion des imports et des conflits.....	94
A. Logique.....	95
B. Importation des données.....	96
a. Import du fichier.....	96
b. Fonction de nettoyage des données.....	96
c. Processus.....	97
C. Affichage des conflits.....	97
D. Gestions des conflits.....	98
E. Code complet.....	98
4. POUR ALLER PLUS LOIN.....	99
4.1 La solution de Récolement de l'information.....	99
4.1.1 État Actuel.....	99
4.1.2 Abonnement QFieldCloud Organization.....	99
4.1.3 Déploiement d'un Serveur QFieldCloud Personnel.....	99
4.2 L'application Django.....	100
4.2.1 Phase Expérimentale.....	100
4.2.2 Optimisation et Interface Utilisateur.....	100
4.2.3 Ressources Nécessaires pour Finalisation.....	100
4.2.4 Expérience et Problèmes.....	100
5. CONCLUSION.....	101
ANNEXE 1.....	102
Code du plugin Récolement Unima.....	102
etape1_dialog.py programme complet.....	102
etape2_dialog.py programme complet.....	114
etape3_dialog.py programme complet.....	132
etape4_dialog.py programme complet.....	139
ANNEXE 2.....	154
Code de l'application Django.....	154
d.....	154
ANNEXE 3.....	155
Documents annexes de la partie Récolement Unima.....	155
regex utilisé pour requête SQL ASAPerimetre.....	159
Sous-ensemble des travaux d'un marché résultant d'un découpage dans le temps.....	10
Référentiel existant à l'Unima et mis à jour par le pôle SIG, accessible depuis une base de données.....	10

REMERCIEMENT

Je tiens à remercier tout particulièrement mes maîtres de stage, Claire Bouquet et Clément Poirier, qui m'ont suivi et épaulé tout au long de ces quatre mois, et m'ont accompagné jusqu'à l'écriture de ce rapport. Leur aide précieuse a été inestimable tant sur le plan technique que personnel.

Je remercie également tout le personnel de l'Unima, qui s'est montré réceptif à mes demandes et questions, partageant leur enthousiasme tout au long de mon stage. Un merci spécial au pôle Prévention Inondation, qui m'a accueilli dans leurs bureaux à la fin de mon stage.

Je souhaite exprimer ma gratitude envers Thierry, mécanicien de l'Unima, pour son aide précieuse lors d'un problème sur mon véhicule.

Merci également à Mickaël, directeur de la régie travaux, pour sa réaction positive et enthousiaste à la conclusion de mon stage.

Je remercie Christophe Chastaing, directeur général de l'Unima, pour avoir autorisé cette expérience professionnelle et accueilli avec bienveillance les résultats de mes travaux.

Enfin, je tiens à remercier Alain Layec, qui a répondu à mes questions et m'a guidé au début du stage, notamment sur des problèmes rencontrés avec mes scripts Python.

1. INTRODUCTION

1.1 Présentation de l'UNIMA

1.1.1 Contexte Historique

L'UNIMA, fondée au début des années 1950 par le Conseil Général et les Associations Syndicales du marais du département, avait pour objectif de restaurer les réseaux hydrauliques négligés pendant et après la guerre. Initialement une association loi 1901, elle est devenue en 1953 l'union des 120 Associations Syndicales de Charente-Maritime, puis un Syndicat Mixte Ouvert en 1965 pour inclure les communes et regroupements de collectivités. Aujourd'hui, elle compte près de 250 membres couvrant plus des deux tiers de la Charente-Maritime.

À ses débuts, l'UNIMA offrait deux services principaux : un support administratif et financier pour les dossiers et la gestion de projets, et un outil opérationnel, la RIEM, pour la restauration et l'entretien des réseaux hydrauliques. Actuellement, elle dispose d'un bureau d'études, d'une équipe de travaux, de services administratifs, juridiques et financiers, et d'éclusiers pour son réseau.

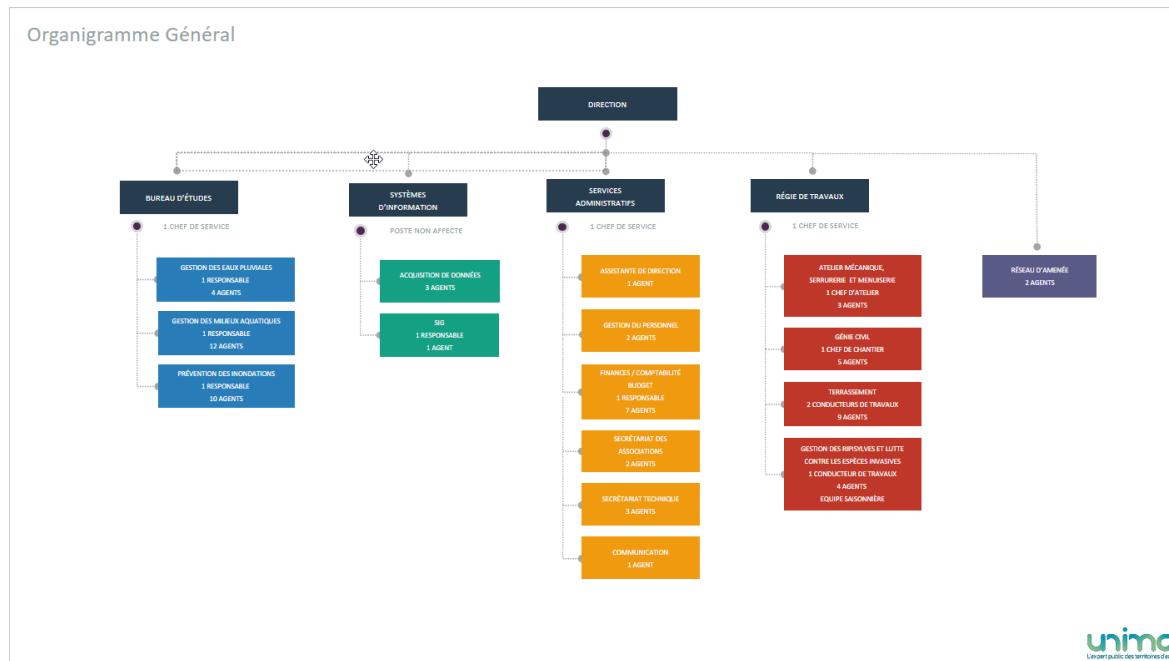
L'UNIMA reste engagée auprès de ses membres, offrant conseils et soutien sur les questions liées à l'eau et à la prévention des inondations. En tant que plateforme mutualiste, elle propose une large gamme de services pour répondre aux besoins de ses membres et des territoires, tout en restant fidèle à sa vocation de service public.

1.1.2 Présentation de la structure

La structure se compose de 4 services distincts :

- ⇒ Le bureau d'études produit des études techniques et réglementaires concernant la gestion des milieux aquatiques (marais, rivières), la prévention contre les inondations et la gestion des eaux pluviales,

- ⇒ La Régie de Travaux réalise l'entretien des canaux et des fossés dans les marais de Charente-Maritime, principalement avec du curage mécanique (pelleteuses), ainsi que la mise en place et la réfection des ouvrages hydrauliques de gestion,
- ⇒ Le réseau d'aménée assure la réalimentation en eau brute des marais de Rochefort depuis la Charente via un réseau de canaux,
- ⇒ Le service administratif propose un secrétariat auprès des propriétaires regroupés en Associations Syndicales de marais, pour assurer la levée de fonds nécessaires à l'entretien du réseau hydraulique syndical.

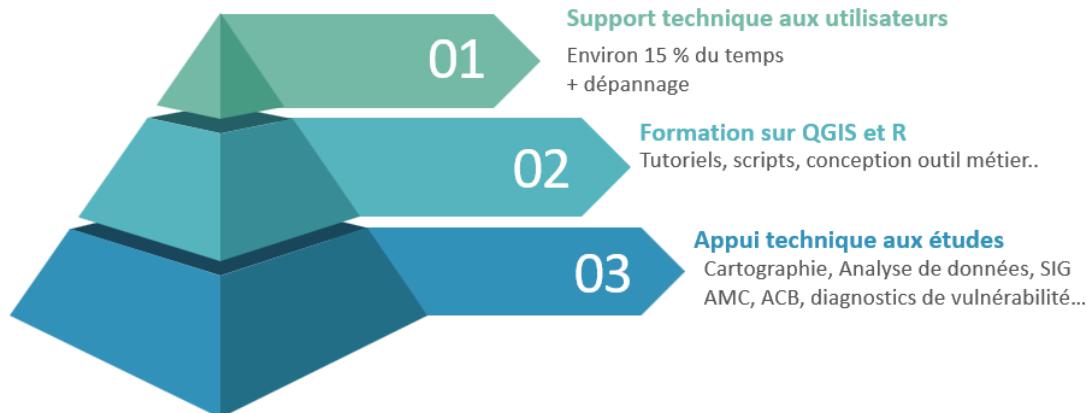


Organigramme général de l'Unima

1.1.3 Présentation du pôle SIG

Le pôle SIG de l'UNIMA, composé de Claire Bouquet et Clément Poirier, apporte un appui technique en cartographie, analyse de données, indicateurs statistiques et

conception d'outils métier. Ils offrent également des formations QGIS et un dépannage informatique.



Représentation du temps et des tâches réalisé par le pôle SIG

Le pôle SIG intervient dès la phase de devis et au début des projets pour mettre en place les ressources matérielles et logicielles nécessaires. Il réalise des tâches géomatiques comme les géotraitements, l'analyse spatiale, les statistiques, et produit des documents cartographiques. Le pôle SIG fournit un appui technique dans la réalisation des projets pour lesquels le Bureau d'Études est sollicité.

Le pôle est également responsable de la conception et de l'administration de bases de données géographiques (PostgreSQL/PostGIS), incluant des données institutionnelles et internes, tel que :

- Ouvrage et échelles limnimétriques,
- Limites des Association Syndicales de marais,
- Réseau hydraulique syndical,
- Levés topographiques,
- Travaux sur le réseau hydraulique (régie de travaux)

1.2 Présentation des missions du Stage

1.2.1 Mission Principale - Application Métier

Une application de récolelement des informations sur les travaux d'entretien du réseau hydraulique syndical sera développée. Ce projet vise à améliorer la gestion et la visualisation des travaux d'entretien, rendant l'outil accessible à tous les acteurs impliqués.

Les encadrants du stage ont fixé l'objectif de finaliser et déployer l'outil de manière opérationnelle. Ce projet "obligatoire" sera réalisé en collaboration avec la Régie de Travaux (conducteurs de chantiers, pelleteurs) et le bureau d'études chargé de produire des Plans Pluriannuels de Travaux pour l'entretien du réseau hydraulique.

1.2.2 Mission Secondaire - Logiciel Administratif de gestions financier des propriétaires

Dans un second temps, une application pour le suivi administratif des Associations Syndicales de marais sera développée. Contrairement au premier projet, celui-ci n'a pas d'objectif strict défini par les encadrants du stage. L'accent sera mis sur l'initiation du développement de l'outil et, éventuellement, sur son déploiement, selon les ressources disponibles.

Cette application permettra la consultation, la saisie d'informations, et la production de documents administratifs à partir de données cadastrales nominatives des Associations Syndicales de marais.

2. RÉCOLEMENT DES TRAVAUX D'ENTRETIEN DES MARAIS

2.1 Problématique

2.1.1 Présentation du problème

L'UNIMA souhaite améliorer la communication avec ses membres, en particulier les Associations Syndicales de propriétaires du marais et le conseil départemental, concernant le suivi des travaux d'entretien du réseau hydraulique. La solution technique doit intégrer et historiser les informations des travaux dans une base de données, puis les diffuser via le WebSIG.

De plus, cette solution doit être accessible à un public peu familier avec les SIG, comme les pelleteurs sur le terrain et certains membres du Bureau d'Études. Elle vise également à assurer une meilleure cohérence dans la documentation transmise à la Régie de Travaux de l'UNIMA, en se basant sur un réseau hydrique de référence.

2.1.2 Etat des lieux

L'UNIMA consacre des ressources considérables à l'entretien des réseaux hydrauliques des syndicats de propriétaires adhérents. Le processus commence par le Bureau d'Études, qui élabore en concertation avec les propriétaires, un Plan Pluriannuel de Travaux (PPT) en respectant les contraintes techniques et réglementaires. Ce plan inclut un résumé des contraintes environnementales, l'état initial du réseau, des schémas de bonnes pratiques pour la réalisation des travaux, et des cartes de localisation.

Les conducteurs de travaux reçoivent des plans papier au format A3 pour chaque tranche de travaux¹, bien que ces plans manquent de cohérence graphique uniforme en raison des pratiques variées au sein du Bureau d'Études. De plus, les géométries des

¹ Sous-ensemble des travaux d'un marché résultant d'un découpage dans le temps.

tronçons dans les cartes ne sont pas basées sur un référentiel standard du réseau hydrique existant² ; elles sont souvent redessinées arbitrairement à partir d'orthographie.

Les conducteurs de travaux utilisent ces plans pour organiser les chantiers et y notent les progrès et remarques, qui sont essentiels pour la facturation par l'UNIMA et le suivi de chantier, en prenant en compte le mètre linéaire curé et le temps d'entretien divers. Cependant, ces plans peuvent revenir dans des états variés, souvent difficiles à lire. Les annotations sont ensuite transférées dans des tableurs Excel pour la gestion administrative et la facturation.

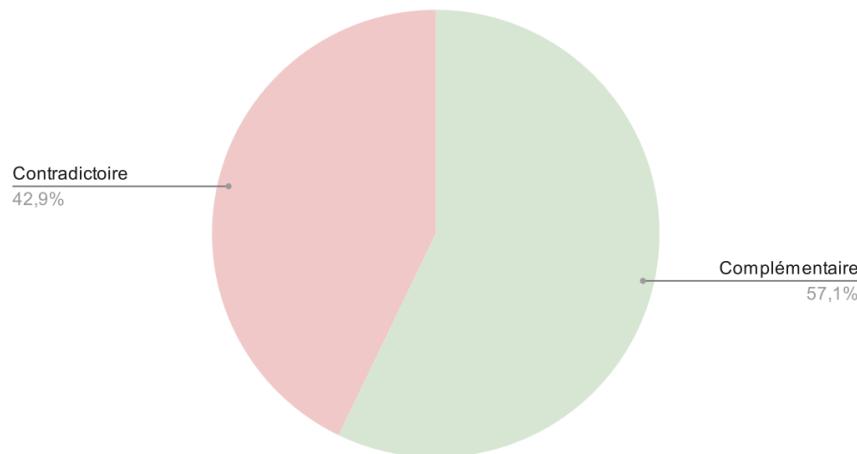
Il est à mentionner qu'une solution similaire a été mise en œuvre par la Communauté de Communes Marennes-Olérons, utilisant l'application GEO de Business Geographic. Cependant, cette solution est jugée trop coûteuse pour être adoptée par l'UNIMA.

2.2 Crédit d'un cahier des charges

2.2.1 Critères de la solutions

Pour mieux comprendre les besoins de chaque acteur impliqué, les premières semaines de stage ont été dédiées à l'organisation de plusieurs réunions. Celles-ci ont mis en lumière les points communs et les différences entre le Bureau d'Études et la régie quant aux méthodes de travail et aux fonctionnalités attendues pour la solution. Les conclusions de ces discussions (**voir ANNEXE**) ont fourni des orientations claires sur les fonctionnalités attendues de cet outil.

² Référentiel existant à l'Unima et mis à jour par le pôle SIG, accessible depuis une base de données



Répartitions des demandes de la Régie travaux et du Bureau d'Étude sur les fonctionnalités de l'outils

Après consultation avec le pôle SIG, nous avons résolu les divergences entre les demandes de la régie et du Bureau d'Études. Les critères principaux de la solution ont été définis comme suit :

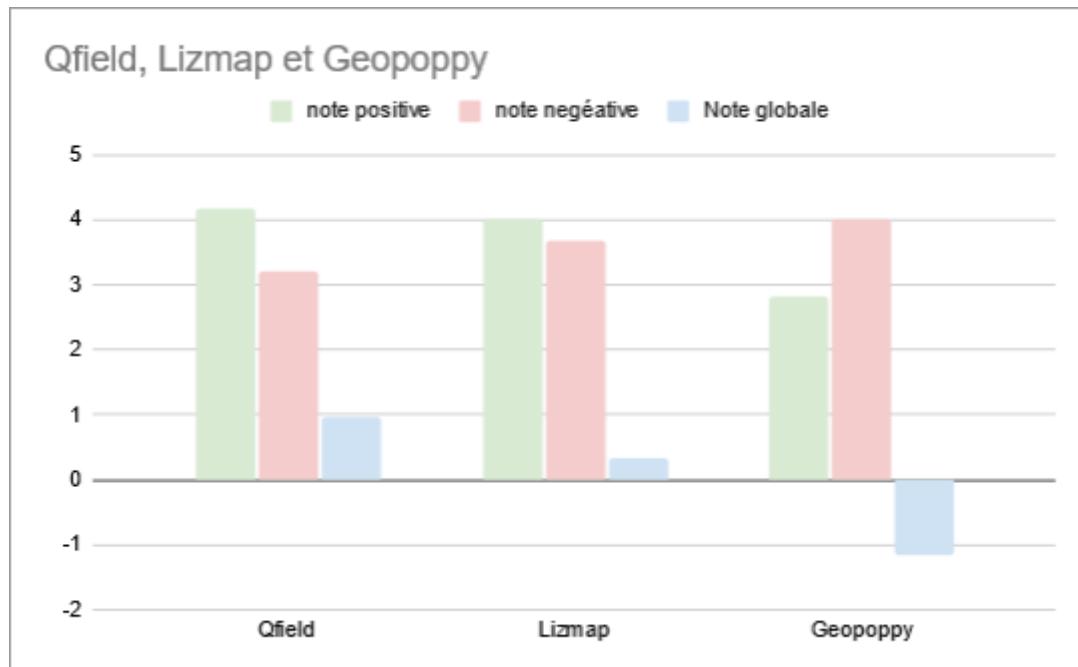
- ⇒ Utilisation d'un référentiel hydraulique déjà existant pour un récolement et une historisation efficace des données.
- ⇒ Mise en place d'une charte graphique cohérente pour les Plans Pluriannuels de Travaux afin d'aider la régie travaux dans la préparation des chantiers
- ⇒ Création d'une base de données pour répondre aux besoins spécifiques de l'application.
- ⇒ Utilisation de tablettes sur le terrain pour faciliter la prise de notes et la clarté des informations transmises.
- ⇒ Possibilité d'utilisation hors connexion en raison de zones blanches potentielles dans les marais.
- ⇒ Historisation des travaux pour une meilleure anticipation des projets futurs à partir des données archivées.
- ⇒ Synchronisation entre les bases de données interne et externe (LizMap) pour une mise à jour automatique du WebSIG.
- ⇒ Visualisation via WebSIG pour faciliter la consultation et l'accès aux données pour les partenaires de l'Unima.

2.2.1 Solutions envisagées

Il a été envisagé 3 outils différents pour créer la solution :

- ⇒ LizMap
- ⇒ QField
- ⇒ GeoPoppy

Les différents avantages et inconvénients de ces outils ont été synthétisés dans un tableau (**VOIR ANNEXE**) et pondérés pour donner une note globale, qui aide à mieux identifier outils appropriés.



Notation des différents outils envisagés dans le cadre de la création de la solution

Dans un premier temps, l'outil LizMap était privilégié en raison de la familiarité des membres du pôle SIG avec cet outil. LizMap permet non seulement la visualisation mais aussi l'édition des données pour suivre en temps réel l'avancement des travaux.

Cependant, LizMap présente plusieurs problèmes significatifs dans ce contexte. L'intégration des informations stockées dans la BD LizMap vers la BD interne de l'Unima ne peut être automatisée et nécessite l'intervention d'un administrateur. De plus, l'utilisation d'un WebSIG exige une connexion Internet constante, contrairement aux exigences du cahier des charges, bien que cette contrainte soit de moins en moins réelle. Enfin, l'interface peu intuitive pour les pelleteurs a été critiquée, car elle n'est pas adaptée à des utilisateurs non familiers avec la géomatique.

Explication

Les zones blanches, caractérisées par l'absence de couverture réseau, ont toujours existé dans le marais. Leur présence s'explique principalement par deux facteurs :

- Éloignement des centres d'activité humaine : Les marais sont isolés et éloignés des centres urbains et des infrastructures réseau.
- Couverture arbustive : La densité de la végétation, notamment les zones arbustives, qui interfère avec les signaux de télécommunication.

De nombreux efforts ont été déployés pour réduire ces zones blanches. Les initiatives comprennent l'installation de nouvelles infrastructures, l'amélioration des technologies existantes et la collaboration avec les opérateurs télécoms pour renforcer la couverture réseau.

Grâce à ces efforts, les zones blanches se réduisent progressivement. Il est prévu que, dans un futur proche, cette problématique sera complètement résolue, offrant ainsi une couverture réseau homogène à travers tout le marais.

GeoPoppy a été envisagé brièvement mais son utilisation peu intuitive au niveau matériel (Kit à monter soit même pour le hardware et le software) et la fréquence insuffisante des retours des tablettes³ ont rendu la mise à jour des données trop irrégulière.

Finalement, QField semble être la meilleure option pour répondre aux exigences du cahier des charges. QFieldCloud résout le problème de la périodicité des retours des tablettes en utilisant le stockage cloud de opengis.ch. Bien que le formulaire d'attributs

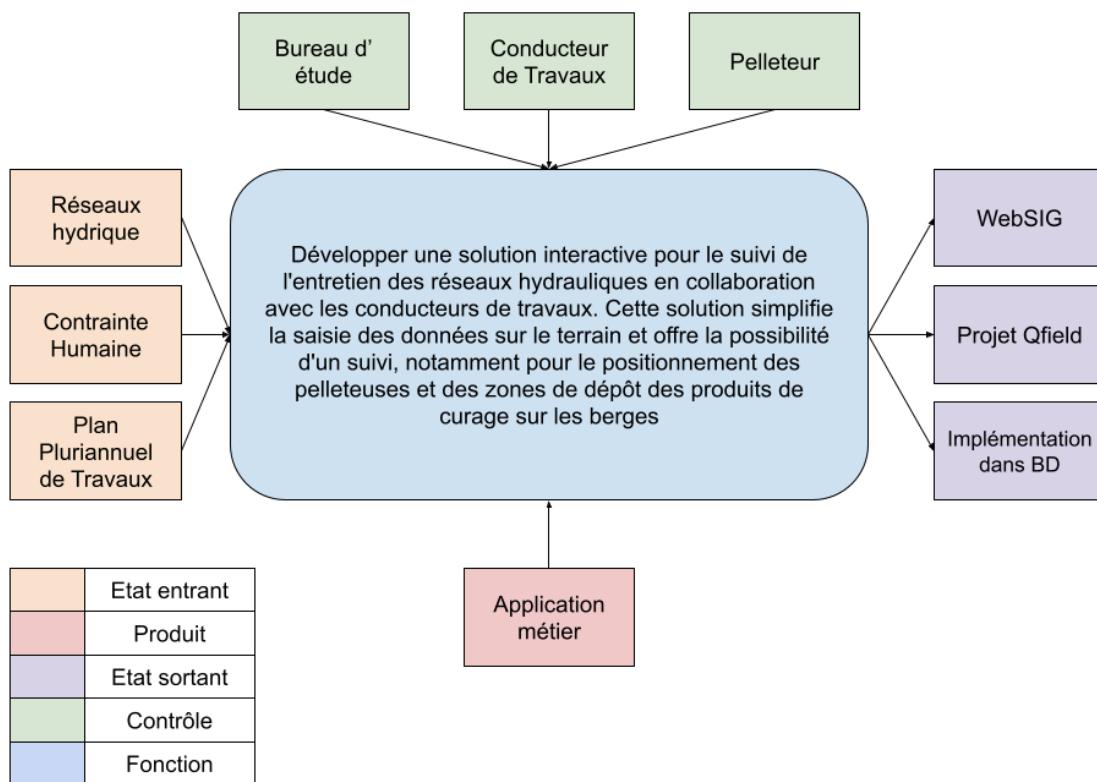
³ Les pelleteurs en possession de tablette viennent rarement sur le site de l'Unima, de l'ordre d'une fois par mois.

de QField ne supporte pas directement les fichiers .ui pour une personnalisation avancée, il offre suffisamment de flexibilité via l'importation de formulaires de type glisser-déposer depuis QGIS. De plus, QField et le plugin QFieldSync simplifient l'envoi vers les bases de données de l'Unima et LizMap en utilisant un projet QGIS local pour la mise à jour des données dans ces bases.

2.3 La solution développée

2.3.1 Elaboration du processus de réflexion

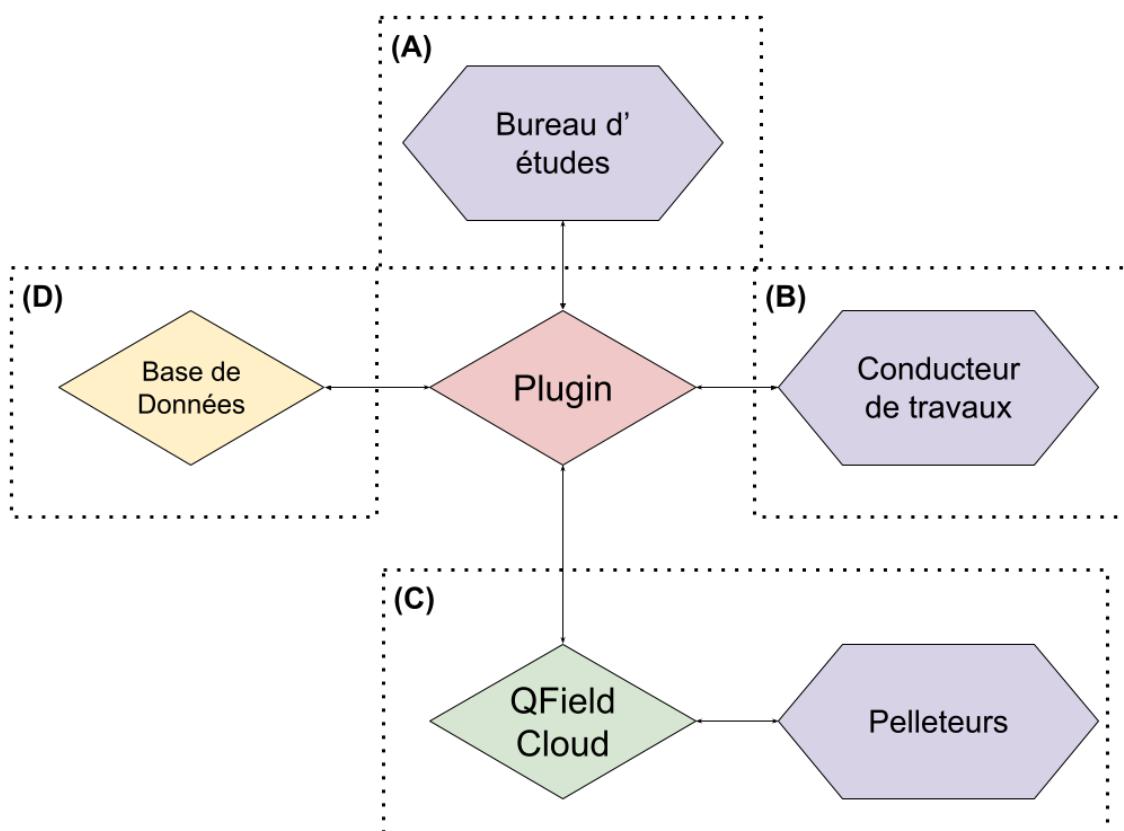
Le processus de réflexion pour mettre en place la solution a été une succession d'étapes avant d'arriver à un schéma cohérent de penser. Dans un premier temps un SADT de niveau 0 a permis de remettre en contexte ce que devait apporter la solution :



SADT de niveau 0 pour la solution de récolelement d'informations sur les travaux de curages

2.3.2 Schématisation du fonctionnement de la solution

Une fois le cadre du projet établi, une schématisation du fonctionnement de la solution a été mise en place. Plusieurs versions (**VOIR ANNEXE**) ont été réalisées, avant d'arriver à la version finale.



Version simplifiée de la solution développée

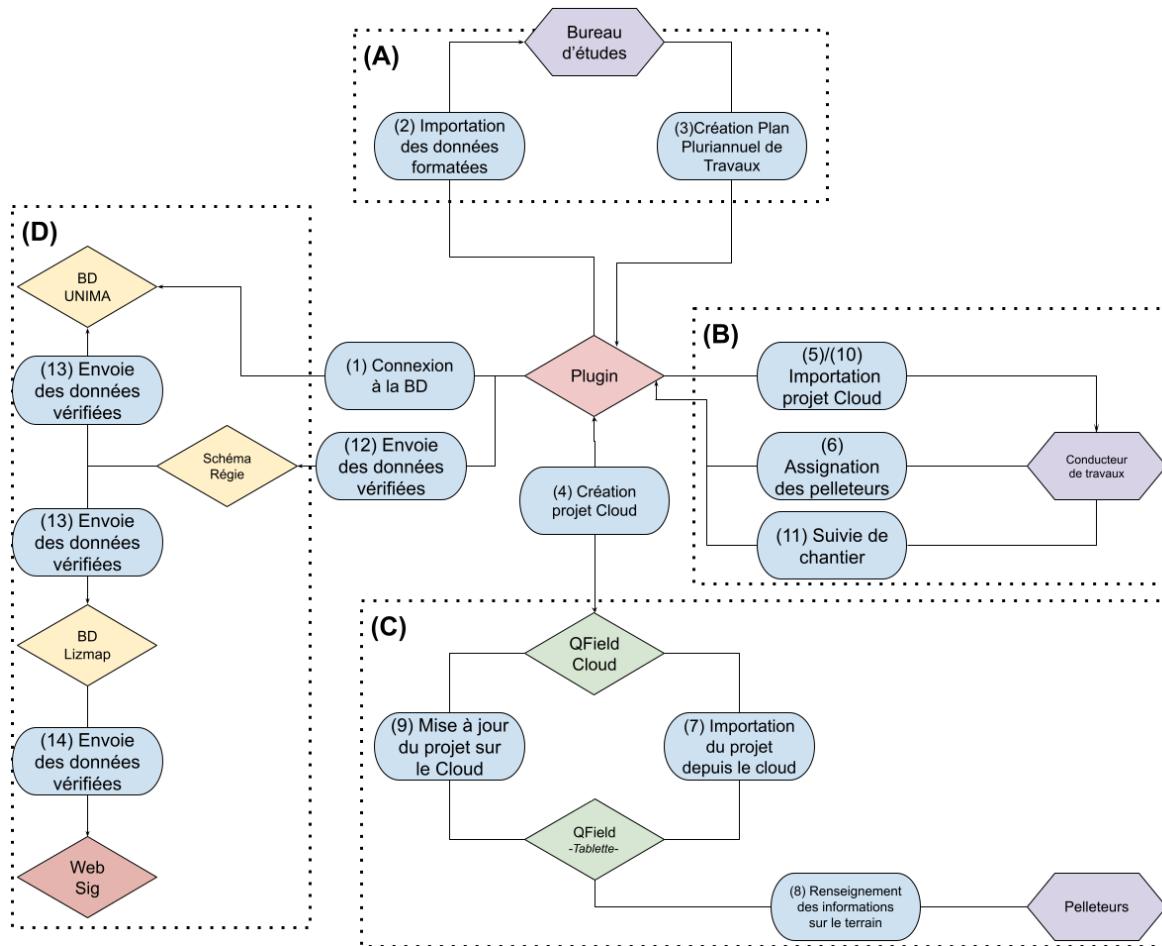


Schéma détaillé de la solution développée

2.3.3 Explication du fonctionnement de la solution

La solution fait donc intervenir 3 acteurs principaux :

- ⇒ Le Bureau d'Études
- ⇒ La Régie de Travaux
- ⇒ Les Pelleteurs

Et 6 éléments techniques

- ⇒ Un plugin QGIS codé en python
- ⇒ Un serveur externe (via QFieldCloud)
- ⇒ Du matériel portable avec les tablettes terrains
- ⇒ Une application android (QField)
- ⇒ Trois bases de données
- ⇒ Un WebSig Lizmap.

Cette solution se décompose en 14 étapes distinctes, découpées en 4 blocs.

A. *Le Bureau d'Études*

Les personnels du Bureau d'Études sont les premiers acteurs à intervenir au cours du processus. Ils sont chargés de créer le Plan Pluriannuel de Travaux (PPT) comme décrit précédemment.

Pour les aider dans leurs tâches, harmoniser les données et la symbolologie associée à ces études, une partie du plugin leur est dédié. Ils ont aussi la responsabilité de créer le projet à envoyer sur le cloud (appelé Projet Cloud).

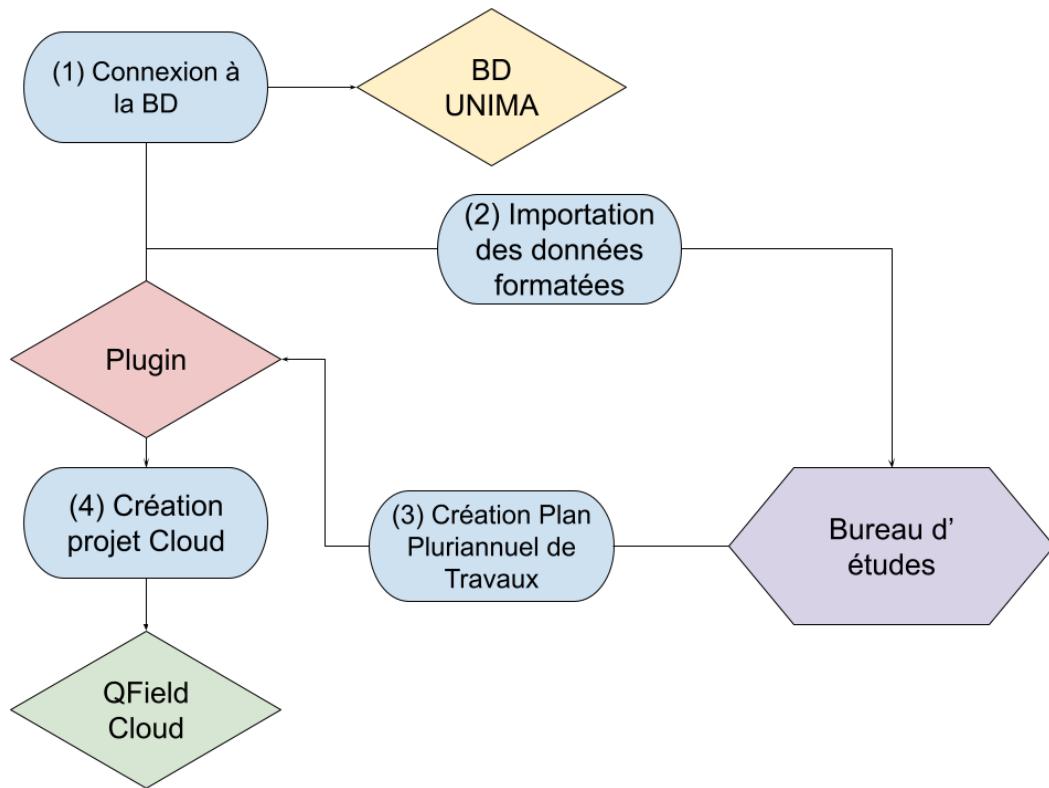


Schéma du bloc Bureau d'Études

B. Le conducteur de travaux

Le conducteur de travaux désigné pour superviser les travaux demandés est le deuxième acteur à intervenir. La présence du projet dans le cloud permettra une simplification de l'échange d'informations. Il pourra télécharger le projet, le vérifier, si besoin le modifier, avant de le renvoyer dans le cloud à l'intention des pelleteurs qu'il désignera pour réaliser le chantier.

Il intervient de nouveau en fin de processus pour vérifier les informations remontées par ses conducteurs d'engins. Une fois la vérification terminée, il aura la charge d'envoyer les données dans une base de données.

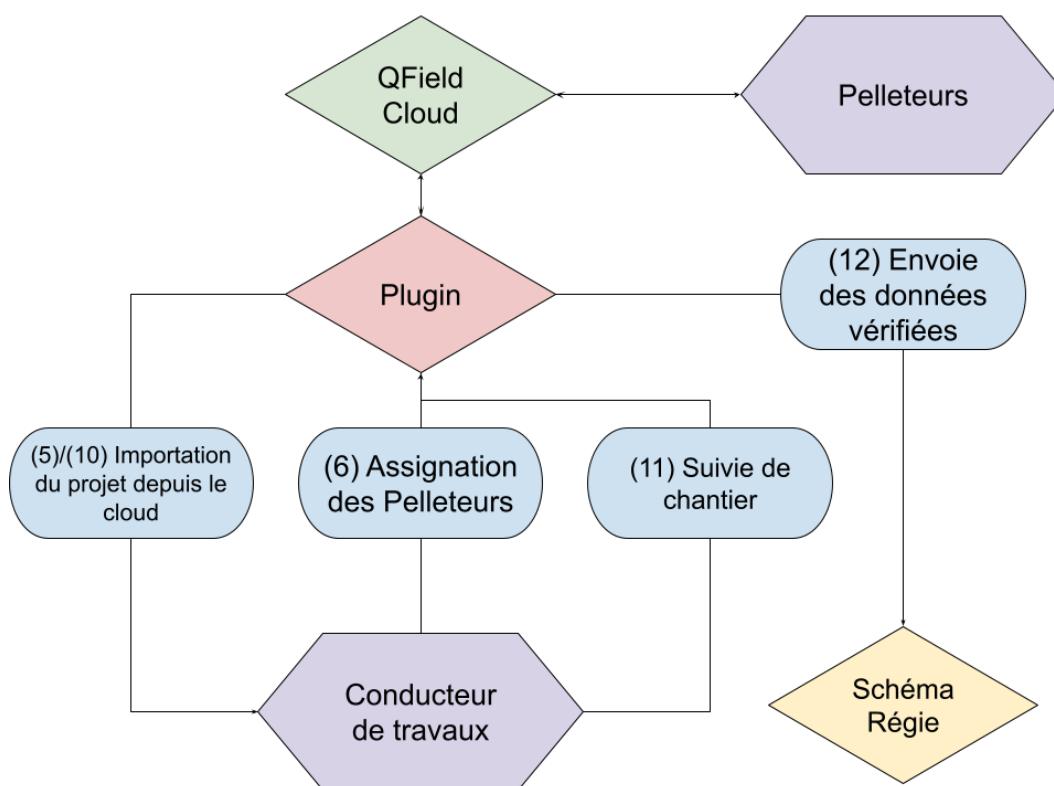


Schéma du bloc conducteur de travaux

C. Les pelleteurs

Les pelleteurs seront les principaux acteurs concernant la prise d'information sur le terrain. Après avoir téléchargé le projet sur les tablettes via l'application QField, ils devront renseigner les informations via des formulaires au sein de cette dernière. Une fois les modifications effectuées, ils devront les remonter grâce à la synchronisation de QField, synchronisant de ce fait le projet sur le cloud.

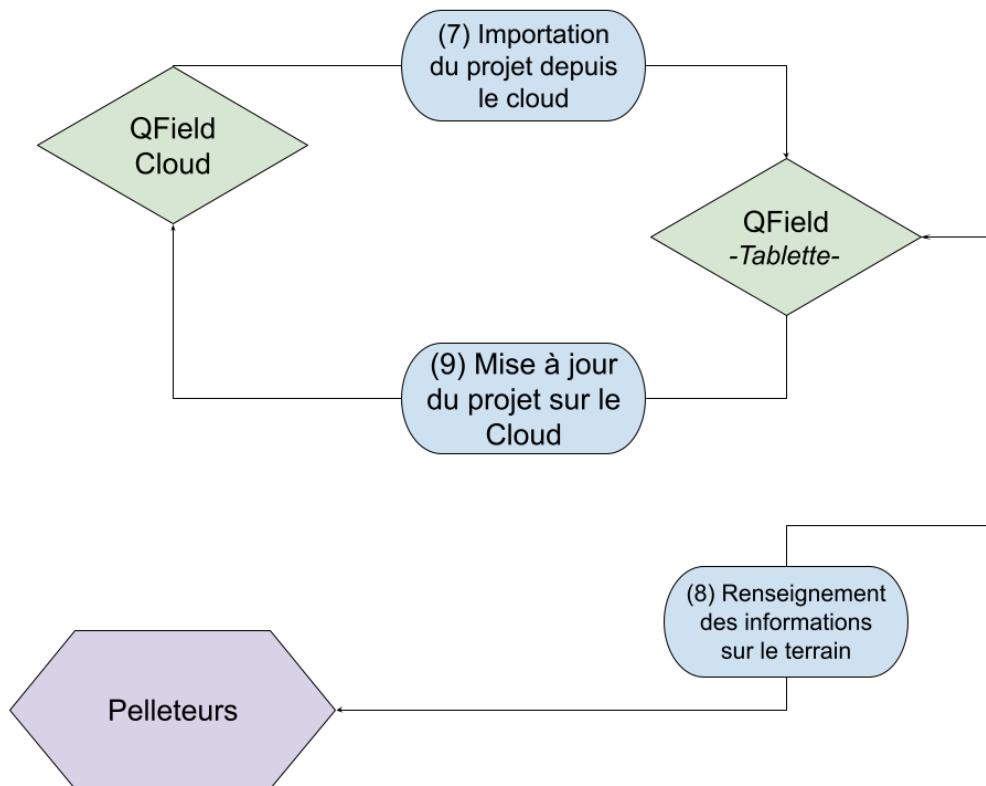


Schéma du bloc pelleteur

D. Base de données

Finalement, une fois les données envoyées sur la base de données Régie, des tâches planifiées (crontab) se chargeront de mettre à jour les bases de données UNIMA et LizMap périodiquement.

La base de données de UNIMA sera actualisée quotidiennement, tandis que celle de LizMap sera mise à jour hebdomadairement.

Définition

Crontab , aussi appelé Cron, est la troncation de chrono table qui signifie « table de planification »

Crontab est un programme qui permet aux utilisateurs des systèmes Unix d'exécuter automatiquement des scripts, des commandes ou des logiciels à une date et une heure spécifiée à l'avance, ou selon un cycle défini à l'avance.

Enfin, le WebSig de l'Unima fonctionnant sur LizMap, avec un projet connecté à la base de données du même nom, sera mis à jour automatiquement.

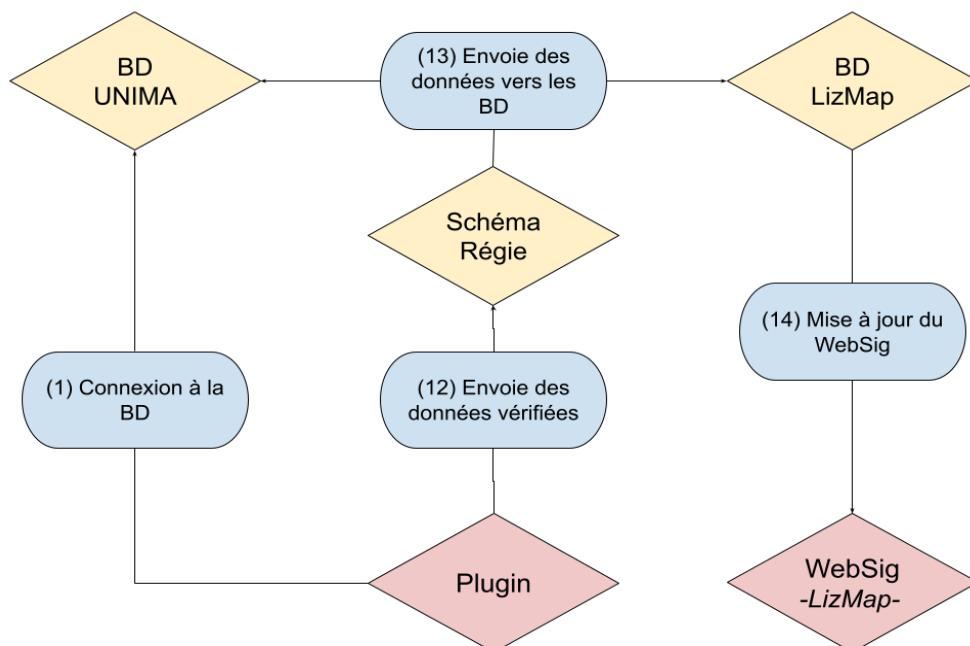


Schéma du bloc pelleteur

Cette solution a d'abord été testée en réalisant chaque étape de la solution manuellement. Cela a permis de mettre à jour plusieurs problèmes, que nous développerons ultérieurement, mais aussi de changer la procédure mise en place, en témoignent les schémas en annexe (**VOIR ANNEXE**).

2.3.4 Elaboration d'un planning de travail

Enfin, un diagramme de Gantt a été produit pour une meilleure organisation du temps de travail

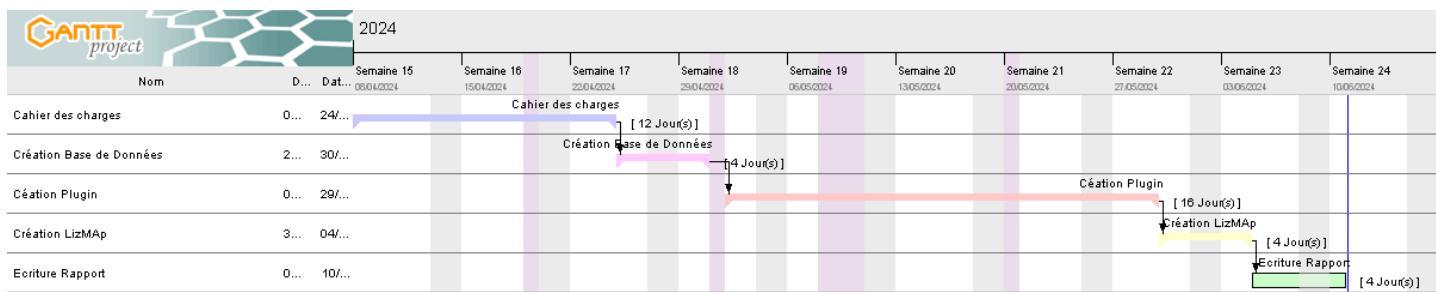


Diagramme de GANTT prévisionnel du projet

Le travail est divisé en quatre grande étapes:

1. La réalisation du cahier des charges vu précédemment
2. La création d'une base de données pour stocker les informations
3. La partie développement du plugin en Python
4. La mise à jour du Lizmap

Enfin du temps est alloué à la rédaction du rapport en fin de projet pour une meilleure gestion du temps de ce dernier.

2.4 Les bases de données

2.4.1 La base de données UNIMA

Cette base de données constitue le principal référentiel d'information de l'Unima. Un droit de lecture sur cette base est nécessaire pour importer le référentiel du réseau hydrique (table *reseau_syndical*) ainsi que les limites actualisées des associations syndicales des propriétaires de marais (table *limites_as*), droits qui sont accordés à tous les membres de l'Unima.

Le plugin se connectera à cette base en utilisant les connexions existantes stockées dans QGIS à l'Unima. Afin d'importer les tables de manière efficace

2.4.2 Le schéma Régie

Le schéma *regie_travaux* sera un nouvel ajout à la base de données UNIMA, spécialement créé pour la régie de travaux. Les membres de la régie auront tous les droits sur ce schéma qui servira de base tampon. Ce dispositif est essentiel pour prévenir toute manipulation imprudente pouvant entraîner la modification ou la suppression de données critiques. Les informations saisies dans ce schéma seront quotidiennement sauvegardées et transférées vers la base de données principale de l'UNIMA, dans un schéma également créé pour répondre aux besoins de la solution, assurant la sécurité des données.

.

2.4.3 Le schéma d'historisation

Les données seront sauvegardées dans le schéma *historisation_curage*, qui reprend la structure du schéma *regie_travaux*. Ce schéma sera utilisé pour historiser les données et faciliter la consultation des informations sauvegardées ultérieurement. À partir de ce schéma, les données seront envoyées vers la base de données LizMap de manière hebdomadaire.

2.4.4 La base de données LizMap

La base de données LizMap est hébergée à l'extérieur du site de l'Unima. Elle sert de stockage pour les données utilisées dans le WebSIG Lizmap. Un nouveau schéma y sera créé, reprenant la même structure que les autres tables mentionnées précédemment.

2.4.5 La Base de données test

Cette base de données a été créée pour réaliser les tests, disposant des droits maximaux sur cette base (lecture, écriture, deletion), elle a permis de réaliser la structure qui sera copié sur les bases actives du projet (BD UNIMA et BD LizMap).

Elle sera dépendante de 2 tables externes déjà renseignées par l'Unima (*reseau_syndicale*⁴ et *limites_as*⁵) et nécessite la création de 4 tables supplémentaires (*pelleteur*, *observation_type*, *historisation_observation*, *historisation_travaux* et *lineaire_curage*).

Définition des tables

reseau_syndicale : Référentiel du réseaux hydrique présent sur le territoire de la Charente-Maritime

limites_as : Données listant les caractéristiques et l'emprise des associations syndicales des propriétaires de marais que gère l'Unima

pelleteur : Stocke les informations lié aux pelleteurs réalisant les travaux

limites_as : Stocke les différents types d'observations ou de travaux ponctuels pouvant être réalisés

historisation_observation : Table historisant les observations et travaux ponctuels et leurs localisations

historisation_travaux : Table historisant les différents travaux réalisés sur le réseaux hydrique

lineaire_curage : Table listant les géométrie des travaux réalisés sur le réseaux hydrique

La base de données sera construite selon le diagramme suivant :

⁴ Référentiel du réseaux hydrique présent sur le territoire de la Charente-Maritime

⁵ Données listant les caractéristiques et l'emprise des associations syndicales des propriétaires de marais que gère l'Unima

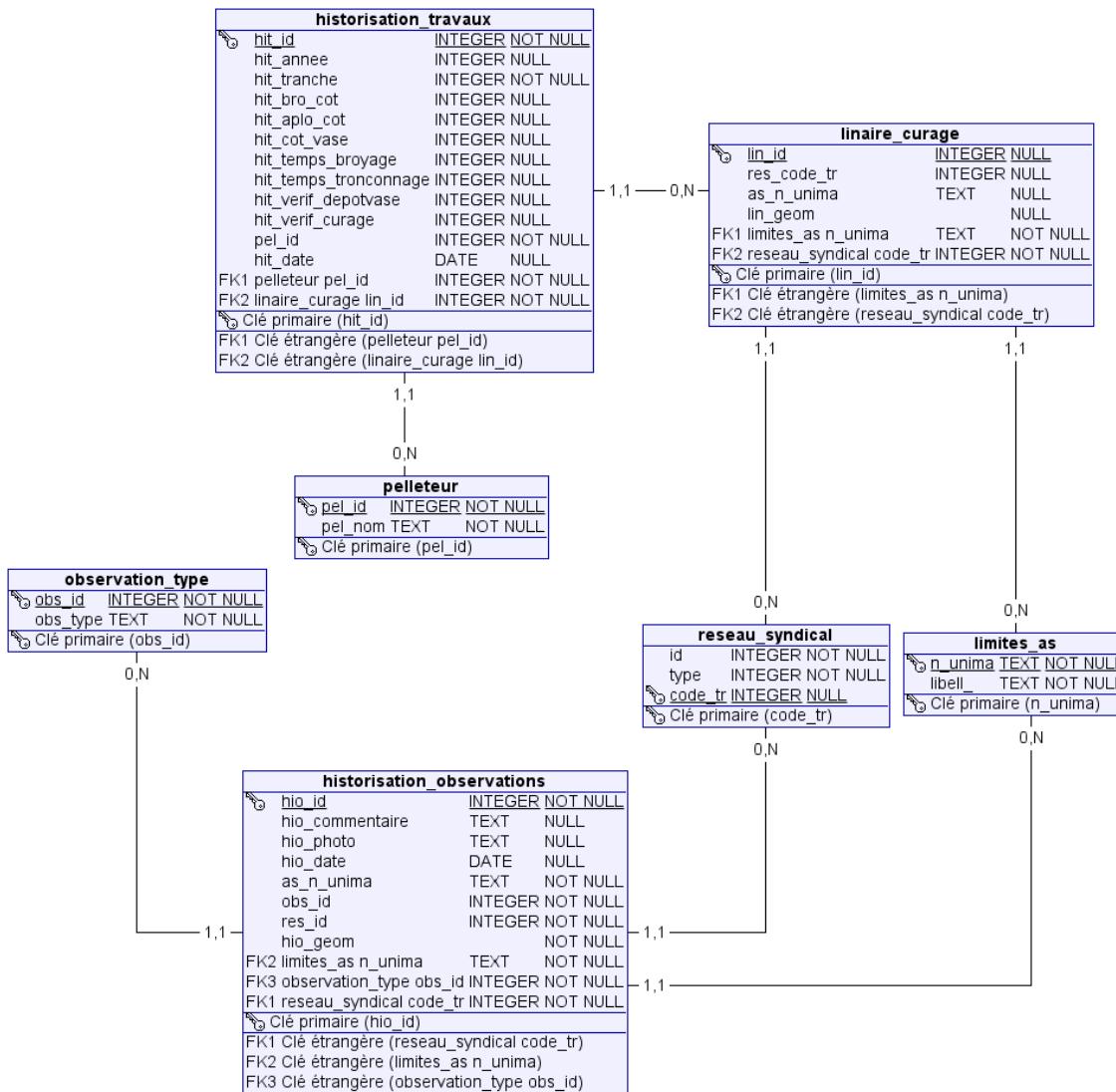


Diagramme relationnel de la base de données test

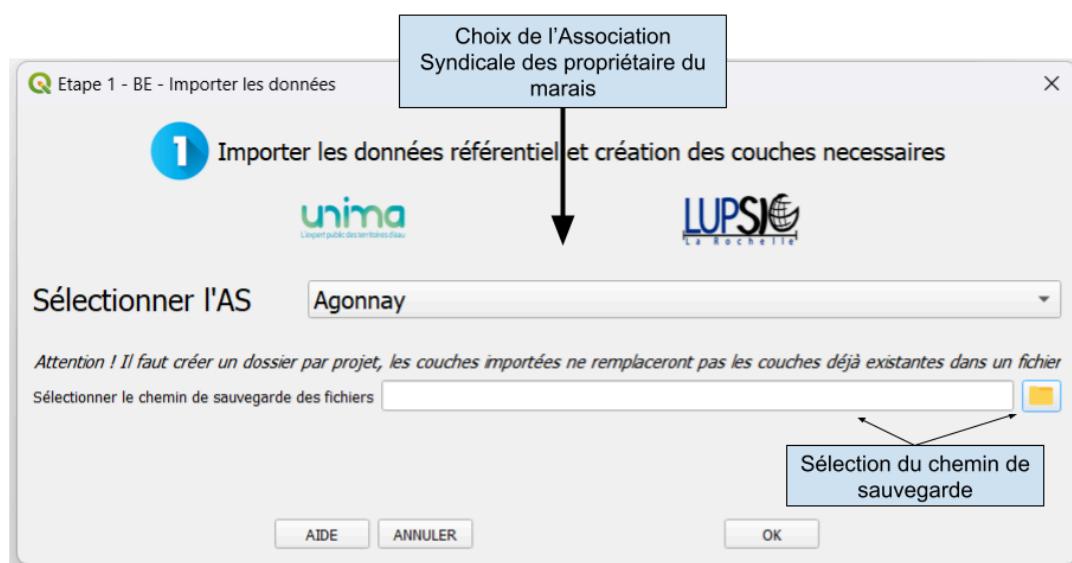
Un script SQL a été créé pour faciliter la créations des différentes tables à trouver en annexe (**VOIR ANNEXE**)

2.5 Le Plugin QGIS

Le Plugin réalisé a été découpé en 2 parties et 4 sous-parties distinctes, chacune répondant à une problématique propre du Bureau d'Études ou de la régie et ayant une interface utilisateur dédiée. Il est important de noter que pour fonctionner, il nécessite l'installation et la modification du plugin QFieldSync.

2.5.1 Bureau d'Études : Crédation du projet

A. L'interface utilisateur



Interface utilisateur de l'étape 1

Cette première partie du plugin sert donc à l'utilisateur, dans ce cas un agent du Bureau d'Études, à importer les couches adéquates lors de la création d'un Plan Pluriannuel de Travaux (PPT). L'agent choisit le nom de l'Association Syndicale des propriétaires de marais (AS) sur lequel porte son PPT ainsi qu'un dossier d'export.

Note

Pour promouvoir les bonnes pratiques de gestion des données et de projets, le plugin requiert un dossier vide pour son fonctionnement.
Cela empêche le stockage aléatoire des projets et des données dans l'arborescence des fichiers de l'utilisateur.

Si le plugin détecte qu'un dossier choisi n'est pas vide, il en informera l'utilisateur et interrompra le processus en cours.

B. Les problématiques

a. L'utilisation d'un référentiel

La première problématique à résoudre est l'utilisation d'un référentiel existant pour la création des nouveaux Plans Pluriannuels de Travaux. Ce référentiel est déjà disponible dans la couche *reseau_syndical* de la base de données UNIMA.

Cependant, certains tronçons (primaires et secondaires) peuvent être très longs, nécessitant des travaux sur plusieurs années ou sur une seule portion du réseau. Il est donc nécessaire de pouvoir modifier la géométrie de ces tronçons au besoin. Le Bureau d'Études n'a cependant pas les droits pour effectuer de telles modifications sur le réseau référentiel directement.

Note

Chaque tronçon du réseau marécageux est classé en trois types : primaire, secondaire et tertiaire. Les réseaux primaires et secondaires bénéficient d'une surveillance accrue en raison de leur rôle crucial dans la gestion de l'eau du marais.

En raison de leurs caractéristiques physiques (largeur, profondeur), ils font également l'objet davantage de subventions pour leur entretien.

Leur classification est déterminée en amont des Plans Pluriannuels de Travaux par les membres des Associations Syndicales des propriétaires du marais (AS).

Pour résoudre cette problématique, la méthode la plus simple consiste à copier les informations de la table *reseau_syndical* dans un nouveau fichier sur le disque local. Cela permet à l'utilisateur de modifier la géométrie et les attributs du réseau tout en conservant un référentiel existant intact. Le plugin automatise ces étapes. Il demandera à l'utilisateur sur quelle Association Syndicale des propriétaires de marais (AS) il souhaite travailler. À partir de cette information et d'une requête SQL, les réseaux hydriques correspondants pourront être facilement récupérés.

Note

Il est possible que certains fossé se comblent avec le temps ou que la géométrie du réseau évolue. Avant de commencer la réalisation des Plans Pluriannuels de Travaux (PPT), les membres du Bureau d'Études doivent d'abord vérifier l'état du réseau.

Dans le tutoriel donné aux membres du Bureau d'Études (BE), il est explicitement demandé de solliciter l'intervention du pôle SIG pour effectuer les changements conformément aux bonnes pratiques et pour garantir le respect de la topologie du réseau.

[Lien vers le code](#)

Une fois la couche importée il faut l'exporter dans le dossier de sauvegarde sélectionné par l'utilisateur. La couche désormais stockée en local peut être importée au projet QGIS. Maintenant que la modification est possible⁶, le plugin supprime les attributs non nécessaires de cette couche.

Note

Les couches provenant d'un référentiel déjà existant, de nombreux champs ne sont pas utiles pour un récolement de l'information.

Ainsi 8 champs sont supprimés de la couche, ce qui améliore le confort utilisateur en réduisant la complexité de la table.

[Lien vers le code](#)

Ensuite, le plugin ajoute de nouveaux champs à la couche⁷, nécessaires pour assurer une bonne exportation dans la base de données à la fin du processus.

[Lien vers le code](#)

Les champs ajoutés vont permettre de transmettre et d'historiser au mieux les informations liées aux travaux de curage et annexe.

De plus, le plugin ajoute un style préenregistré à la couche, comprenant une symbologie et un formulaire d'attributs personnalisé.⁸ (voir 2.5.1.G)

[Lien vers le code](#)

Ces étapes sont réalisées à nouveau la couche *limites_as*.

⁶ Cette étape est nécessaire car une couche issue de la base de données UNIMA est en lecture seule. En la sauvegardant localement, elle devient modifiable, car elle est distincte de la couche référentielle stockée en base de données.

⁷ Vous trouverez un dictionnaire des champs en annexe de ce rapport.

⁸ Pour en savoir plus sur les style importé, ce reporter au [2.5.1.G](#)

[Lien vers le code](#)

b. Crédation de la couche d'observation

Les pelleteurs et les conducteurs de travaux ont exprimé leur souhait d'ajouter des informations préalables au chantier, comme les accès disponibles pour leurs engins entre les parcelles. Pour répondre à ce besoin, une couche de points (*observations_remarques*) est créée. Cette couche facilitera l'échange d'informations entre les pelleteurs et les conducteurs de travaux. Ils pourront ainsi, communiquer la position des pelleteuses à la fin de la journée, des travaux annexes à réaliser, ou toute autre information importante.

Cette nouvelle couche aura comme attributs le type d'information partagée, un commentaire, une photo facultative, une date, ainsi que des références au numéro de l'Association Syndicale (AS) concernée et au code du réseau hydrique le plus proche.

Note

À l'Unima, chaque tronçon du réseau hydrique référentiel possède un code unique. Ce code facilite la communication avec les différents partenaires. Il est composé du numéro d'identification de l'Association Syndicale (AS) et d'un préfixe indiquant le type de tronçon;

Une fonction intéressante du plugin est que les types d'observations sont stockés en base de données (dans la table *observation_type*) et peuvent évoluer avec le temps et les besoins. Le formulaire d'attribut du type d'observation est donc mis à jour à chaque création de projet. Pour ce faire, le plugin importe la table *observation_type*, en extrait les valeurs et les attribue à une liste de valeurs dans QGIS. Ainsi, si la Régie de Travaux ou le Bureau d'Études expriment le besoin d'ajouter des observations particulières pour des chantiers spécifiques (relevés de faune et de flore, travaux particuliers, etc.), le plugin mettra à jour ces informations automatiquement à la création du prochain projet..

[Lien vers le code](#)

c. L'ajout d'un fond de plan

Il a également été demandé de rajouter un fond de plan pour un meilleur confort utilisateur. Le pôle SIG de l'Unima a déjà produit un raster virtuel disponible sur le serveur local de l'Unima. Le plugin se charge juste de l'importer.

[Lien vers le code](#)

d. Autres options réalisées par le plugin

Pour plus de confort utilisateur on effectue également un zoom sur la zone d'étude importée.

Lien vers le code

Enfin, on enregistre le projet pour éviter tout problème ultérieur.

Lien vers le code

e. Les étapes de vérifications

Finalement, il est nécessaire d'effectuer des étapes de vérification pour s'assurer que les informations fournies par l'utilisateur sont correctes avant de lancer le programme.

Lien vers le code

f. Programme complet

Le programme complet concernant cette première partie est indiqué en annexe⁹

g. Les styles

Plusieurs styles ont été créés pour mieux lire, écrire et transmettre les différentes informations renseignées par le Bureau d'Études.

Ces styles sont pré-enregistrés dans les répertoires du plugin, sont appliqués et sauvegardés dans la couche, lors de la création du projet.

Le style de la couche de linéaire est comme suit :

Étiquette	Règle
✓ Principale	"res_type" = 1 AND "hit_tranche" IS NULL
✓ Secondaire	"res_type" = 2 AND "hit_tranche" IS NULL
✓ Tertiaire	"res_type" = 3 AND "hit_tranche" IS NULL
✓ Tranche 1	"hit_tranche" = 1
✓ Tranche 2	"hit_tranche" = 2
✓ Tranche 3	"hit_tranche" = 3
✓ Tranche 4	"hit_tranche" = 4
✓ Tranche 5	"hit_tranche" = 5
✓ Indication coté 1	"hit_bro_cot" IS NULL OR "hit_aplo_cot" IS NULL OR "hit_cot_vase" IS NULL
✓ Débroussaillage coté 1 prévu et non effectué	"hit_bro_cot" = 1 AND ("hit_temps_broyage" IS NULL AND "hit_temps_troncage" IS NULL)
✓ Débroussaillage coté 2 prévu et non effectué	"hit_bro_cot" = 2 AND ("hit_temps_broyage" IS NULL AND "hit_temps_troncage" IS NULL)
✓ Mise à l'aplomb coté 1 prévu et non effectué	"hit_aplo_cot" = 1 AND ("hit_temps_broyage" IS NULL AND "hit_temps_troncage" IS NULL)
✓ Mise à l'aplomb coté 2 prévu et non effectué	"hit_aplo_cot" = 2 AND ("hit_temps_broyage" IS NULL AND "hit_temps_troncage" IS NULL)
✓ Le dépôt de vase est prévu coté 1 et n'a pas été fait	"hit_cot_vase" = 1 AND "hit_verif_depotvase" IS NULL
✓ Le dépôt de vase est prévu coté 2 et n'a pas été fait	"hit_cot_vase" = 2 AND "hit_verif_depotvase" IS NULL
✓ Primaire & Secondaire	("res_type" = 2 OR "res_type" = 1) AND "hit_tranche" IS NOT NULL
✓ —	ELSE

Tout d'abord il existe 3 règles permettant d'afficher les types (primaires, secondaires, tertiaires) de tronçon à la création du projet. Les couleurs utilisées sont une demande des membres du Bureau d'Études, correspondant à une habitude d'utilisation.

⁹ [Lien vers l'annexe](#)

Ensuite vient un ensemble de règles concernant le choix d'une tranche de travaux. Une fois un réseau attribué à une tranche son type n'est plus l'information à faire ressortir à l'utilisateur en priorité, cette règle supplante la précédente.

Vient l'indication sur le canvas de la notion de côté. En effet, le linéaire hydrique se trouvant dans un marais, la notion d'amont et d'aval n'a pas de sens. Par conséquent, nous ne pouvons pas utiliser des notions de droite ou de gauche.

Pour aider les membres du Bureau d'Études à comprendre le côté 1 des tronçons, celui-ci est donc indiqué sur la carte.

Suivant cette logique de côté 1/côté 2, plusieurs règles de symbologie sont mises en place pour visualiser facilement les indications nécessaires aux travaux de débroussaillage, de mise à l'aplomb et à l'indication du côté de dépôt de vase. Cela permettra au pelleteur de plus facilement repérer les travaux à effectuer sur le réseau.

Note

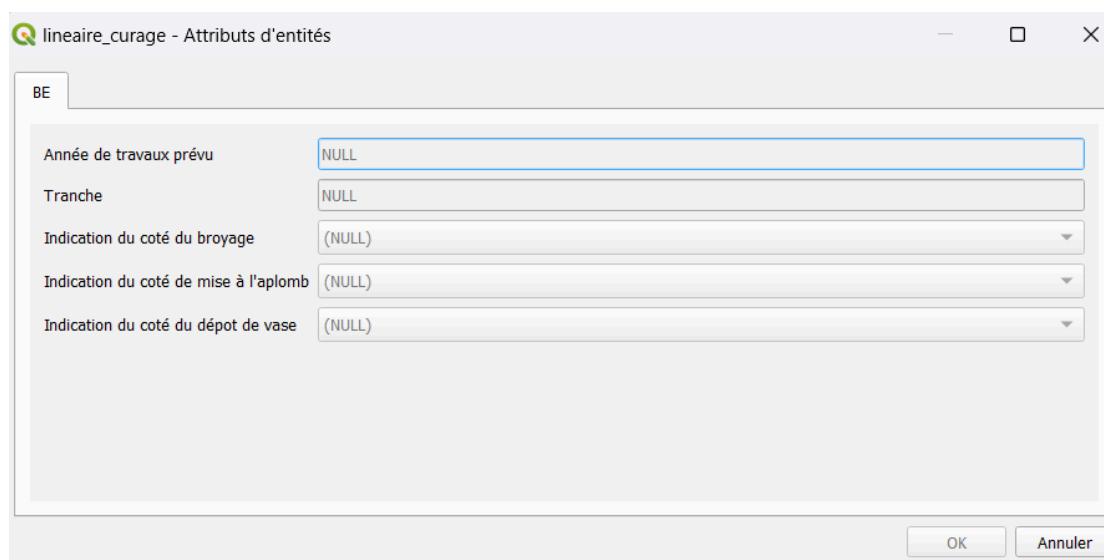
La mise à l'aplomb et le débroussaillage sont des travaux courants lors de l'entretien des canaux du marais. Ils sont nécessaires pour que la pelleteuse puisse avoir accès au fossé. Ces travaux consistent au broyage ou tronçonnage de la flore bordant un fossé. Elle est effectuée suivant des consignes strictes pour favoriser une repousse de cette dernière.

Le dépôt de vase est aussi une notion importante à prendre en compte lors de la réalisation des travaux d'entretiens. Il convient que la vase extraite de ces travaux soit déposée à proximité du tronçon en cours de curage. Le dépôt de vase est donc stocké sur les parcelles bordant les canaux. Généralement les propriétaires alternent le côté du dépôt de vase pour limiter son encombrement à chaque phase d'entretien (toutes les 5 à 10 années).

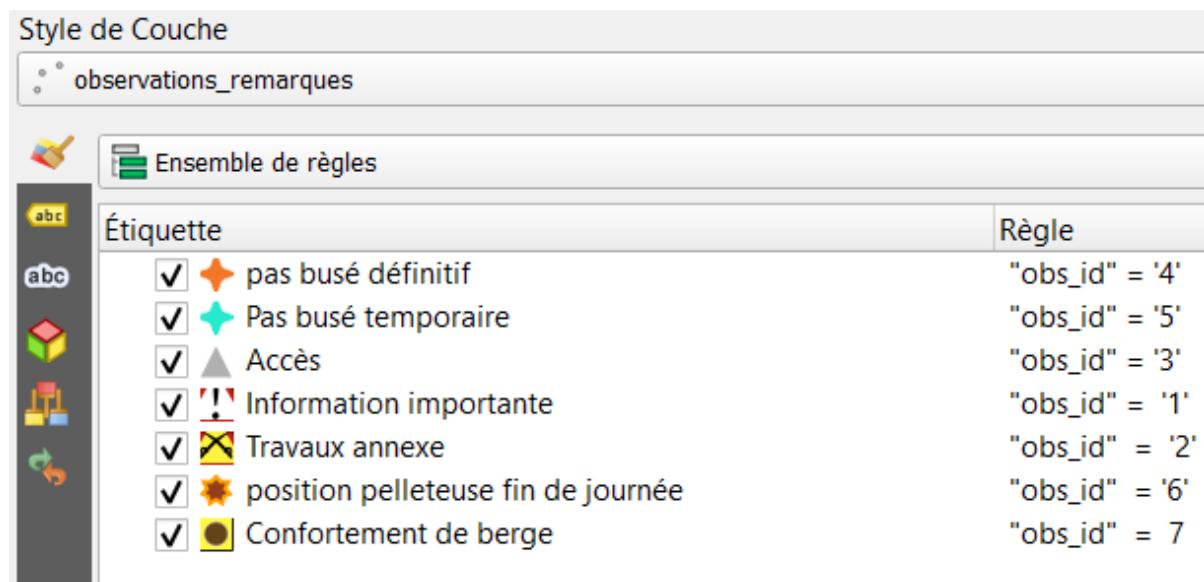
Finalement les extrémités d'un tronçon primaire et secondaire sont délimités par des flèches pour une meilleure lisibilité du tracé.

Les étiquettes du code des tronçons sont également affichées pour mieux se repérer lors d'échange entre les différentes parties en charge du projet.

Ce style permet aussi d'appliquer un formatage au formulaire d'attribut. Le formulaire de type glisser-déposer est retenu étant donné sa compatibilité avec l'application QField. Le formulaire d'attribut se présente de cette façon :

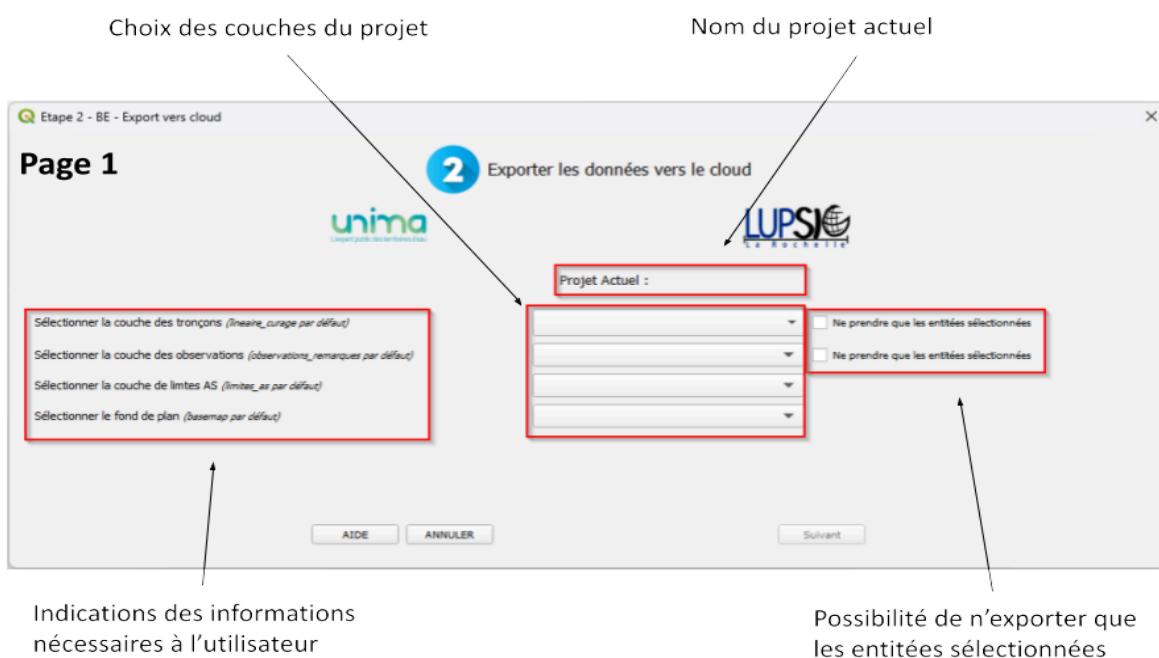


Une symbolologie est aussi mise en place pour la couche d'observation, consistant à créer des pictogrammes facilement reconnaissables pour bien différencier les types d'observations/remarques.

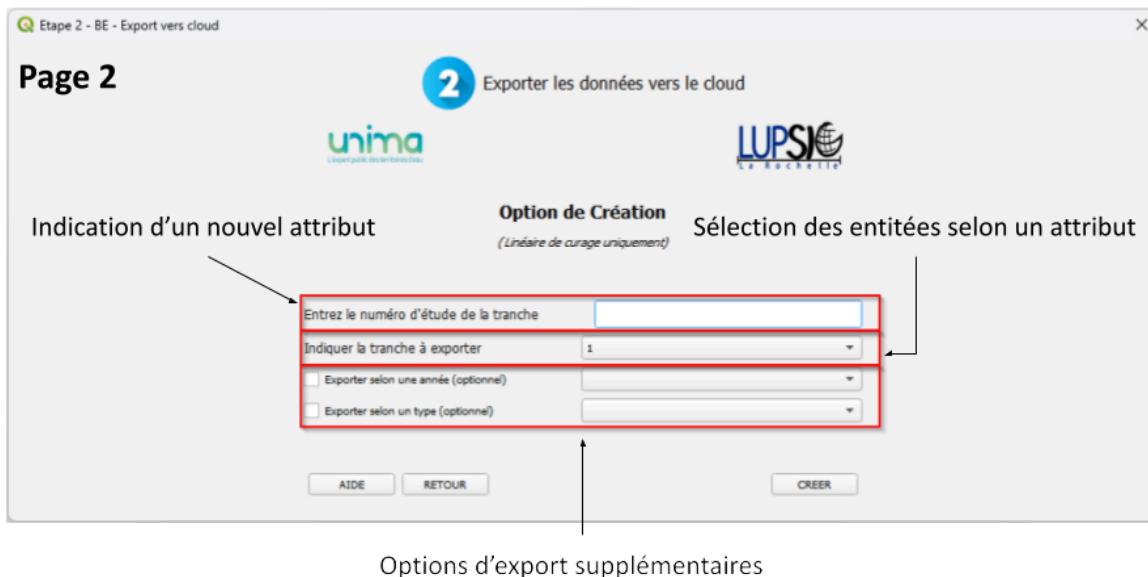


2.5.2 Bureau d'Études : Export du projet vers le Cloud

A. L'interface utilisateur



Interface utilisateur de l'étape 2 - Fenêtre 1



Interface utilisateur de l'étape 2 - Fenêtre 2

Cette interface permet à l'utilisateur de créer un projet QFieldCloud en sélectionnant les couches à inclure. L'utilisateur doit renseigner un numéro d'étude obligatoire et peut choisir d'autres options d'exportation, comme la sélection par année de travaux prévue ou par type de réseau.

B. Les problématiques

a. A quoi sert cette partie du plugin ?

Cette étape intervient à la fin de la création du Plan Pluriannuel de Travaux (PPT). Pour automatiser au maximum le processus de création de projet QFieldCloud, le plugin s'occupe d'extraire et de formater les données du projet en cours. L'utilisateur doit indiquer les couches nécessaires, qui incluent celles importées lors de l'étape 1 et enrichies de nouvelles informations. En fonction des options d'exportation choisies, le projet cloud est créé.

Un PPT en fin d'études peut comporter de nombreuses couches non nécessaires au récolelement de l'information souhaitée. Pour faciliter la création du projet QFieldCloud, le plugin crée un sous-dossier à l'emplacement du projet actuel. Ce nouveau dossier contiendra un projet QGIS et les différentes couches nécessaires. Ce projet servira de référence au plugin QFieldSync, qui se chargera de convertir ce projet en un projet QFieldCloud.

Note

Pour garantir le bon fonctionnement de l'extension QFieldSync et la création du projet cloud associé, de nombreux paramètres doivent être configurés manuellement pour chaque couche du

projet. Ce processus est fastidieux et susceptible d'erreurs, rendant le projet cloud incomplet ou inutilisable.

Deux solutions sont envisageables :

1. Automatisation du paramétrage en Python : Écrire un script Python pour automatiser la configuration des couches.
2. Utilisation d'un projet vierge : Créer un projet QGIS vierge de toutes couches parasites, éliminant ainsi la nécessité de configurer manuellement chaque couche.

La deuxième option est retenue en raison de sa simplicité. Elle évite un rétro-ingéniering trop complexe et minimise le risque d'erreurs humaines. Le plugin s'occupera donc de créer un nouveau projet QGIS ne contenant que les couches nécessaires, facilitant ainsi le processus de création du projet QFieldCloud.

b. Exporter les couches et les entités dans le nouveau projet

Le nom des couches est récupéré grâce aux différentes combobox de l'interface utilisateur.

[Lien vers le code](#)

Puis le plugin gère la gestion des options d'exports.

Lors du choix des exports, il est nécessaire de récupérer les informations depuis les attributs de la couche, afin de remplir les combobox associées, pour un meilleur confort utilisateur.

Ces valeurs uniques sont extraites des champs adéquats dans des listes, qui sont ensuite attribués aux différentes ComboBox.



[Lien vers le code](#)

Plusieurs options d'export sont disponibles. L'export de seulement les tronçons et/ou les observations sélectionner, mais aussi exporter des tronçons selon leur type ou leur année de travaux prévu. Ces options d'export facultatives offrent une meilleure adaptabilité pour la gestion des projets par le Bureau d'Études et la Régie de Travaux.

[Lien vers le code](#)

Note

Pour simplifier l'organisation et la gestion des projets, le plugin génère automatiquement les noms des nouveaux projets et dossiers. Ces noms sont composés des éléments suivants :

- Libellé de l'Association Syndicale (AS) : Indique l'association pour laquelle l'étude est réalisée.
- Tranche de travaux exportée : Spécifie la section des travaux concernés.
- Options supplémentaires : Ajoute des informations supplémentaires si nécessaire.

Cette méthode permet de :

- Faciliter l'identification des projets et des dossiers.
- Éviter les erreurs manuelles dans la dénomination.
- Maintenir une organisation cohérente et structurée des fichiers.

Exemple de Nom Automatique

- AS_Marais_Tranche_1
- AS_Riviere_Tranche_2_Type_3

Ce système de génération de noms automatiques assure une gestion efficace des projets et facilite leur accessibilité pour tous les utilisateurs impliqués.

c. Création d'un fond de plan pour le cloud

Pour que le projet cloud soit complet il faut impérativement qu'il possède un raster de fond de plan (communément appelé Basemap). Cette couche ne doit pas être trop lourde, au risque de saturer les espaces de stockage du cloud, mais suffisamment grande pour permettre de se repérer facilement dans l'espace.

Deux traitements natifs de QGIS sont utilisés pour automatiser ce processus, *MinimumBoudingGeometry* et *Buffer*. Grâce au premier l'emprise minimale des tronçons sélectionnées est délimitée, à laquelle s'ajoute un buffer de 500m pour mieux se repérer. Le résultat de ces opérations est une couche vectorielle temporaire.

Puis, l'utilisation de la bibliothèque GDAL et son traitement *ClipRasterByMaskLayer* vont permettre de découper le raster via une couche vectorielle, ici via la couche temporaire nouvellement créée résultante de *MinimumBoudingGeometry* et *Buffer*. Ce traitement permet aussi d'indiquer le type de compression voulu pour notre image résultante. Il a été rapidement conclu que le type de compression utilisé serait du

JPEG. En effet, la perte de données et de qualité n'est pas très importante car l'unique fonction de ce fond de plan est de permettre à l'utilisateur de se repérer.

Finalement, le raster en sortie de l'algorithme est exporté dans le nouveau dossier.

[Lien vers le code](#)

Note

Avant le début de ce stage, QFieldCloud dupliquait les fichiers raster à chaque import et export d'un projet cloud, ce qui entraînait une saturation rapide de l'espace de stockage disponible. Ce problème impactait significativement l'efficacité des projets nécessitant l'utilisation de données raster.

Un ticket a été ouvert à l'intention de la communauté de développeurs sur GitHub pour signaler cette problématique. Après quelques jours, un contributeur a identifié et remonté le problème. Une semaine après l'ouverture du ticket, un patch a été déployé pour réduire ce problème. Désormais, tout fichier de plus de 8 Go ne sera plus dupliqué en cas de mise à jour d'un projet cloud

<https://github.com/opengisch/QField/issues/5237#issuecomment-2111695728>

<https://github.com/opengisch/QFieldcloud/pull/946>

d. Créer un projet dans un sous dossier

Une fois les différentes couches traitées et créées, il faut les exporter dans le nouveau dossier et les importer dans un nouveau projet. Le plugin réalise en même temps la mise à jour des styles et des différents formulaires.

[Lien vers le code](#)

Note

Lors de la conversion d'un projet QGIS en projet QFieldCloud par le plugin QFieldSync, il est crucial de veiller à la bonne conversion des couches de données. Ces fichiers doivent impérativement se trouver dans le même répertoire que le fichier du projet à convertir.

En effet, l'emplacement des fichiers de données peut engendrer des problèmes de corruption ou d'échec de conversion, entraînant leurs absences

sur les tablettes.

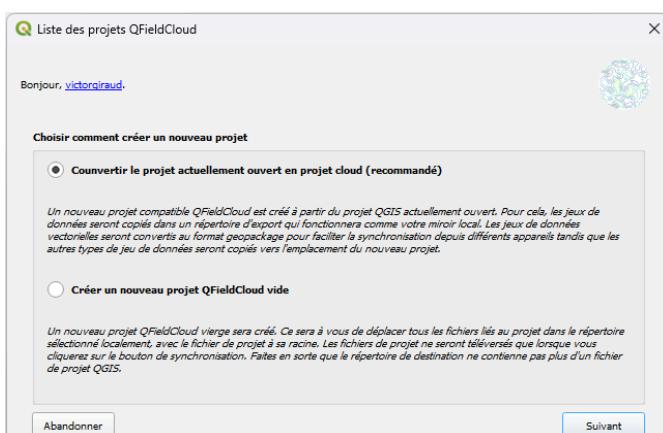
e. Les étapes de vérifications

Plusieurs étapes de vérification sont cruciales, notamment lors de la sélection des couches à exporter. Chaque couche doit comporter des champs spécifiques essentiels pour le bon déroulement des opérations ultérieures, notamment pour l'intégration des données collectées dans la base de données. Si des champs requis sont absents¹⁰, l'utilisateur est alerté et le processus d'exportation n'est pas démarré.

Par ailleurs, le plugin vérifie également la validité du numéro de tranche Unima¹¹ saisi par l'utilisateur. Ce numéro doit exclusivement contenir des chiffres et avoir une longueur de 4 ou 5 caractères pour être accepté. Cette vérification garantit la cohérence des données entrées avant leurs traitements ultérieurs dans le projet QFieldCloud.

[Lien vers le code](#)

f. Utilisation du plugin QFieldSync



Interface utilisateur du plugin QFieldSync
-Création d'un nouveau projet QFieldCloud-

sur le cloud.

Afin de convertir le nouveau projet créé en projet QFieldCloud, l'utilisation du plugin QFieldSync est indispensable. Dans une optique de laisser le moins de marge d'erreur à l'utilisateur, une fenêtre de ce plugin est appelée dès l'ouverture du nouveau projet.

Une fois l'interface lancée, l'utilisateur pourra garder les options par défaut et le projet sera correctement converti et envoyé

¹⁰ Vous trouverez la liste des champs obligatoires en annexe dans le dictionnaire des champs

¹¹ L'Unima référence les travaux réalisés avec un numéro pour ses besoins internes.

Note

Avant de commencer la conversion du projet QField vers QFieldCloud, une interface d'authentification s'ouvre. L'utilisateur doit se connecter pour pouvoir stocker ses projets.

Un compte personnel est utilisé pendant le stage, limitant considérablement la capacité de stockage et le nombre d'accès simultanés autorisés.

Pour un déploiement, l'Unima devra choisir entre souscrire à un abonnement QFieldCloud Pro ou déployer son propre serveur pour héberger un cloud privé, auquel QFieldCloud pourra se connecter.

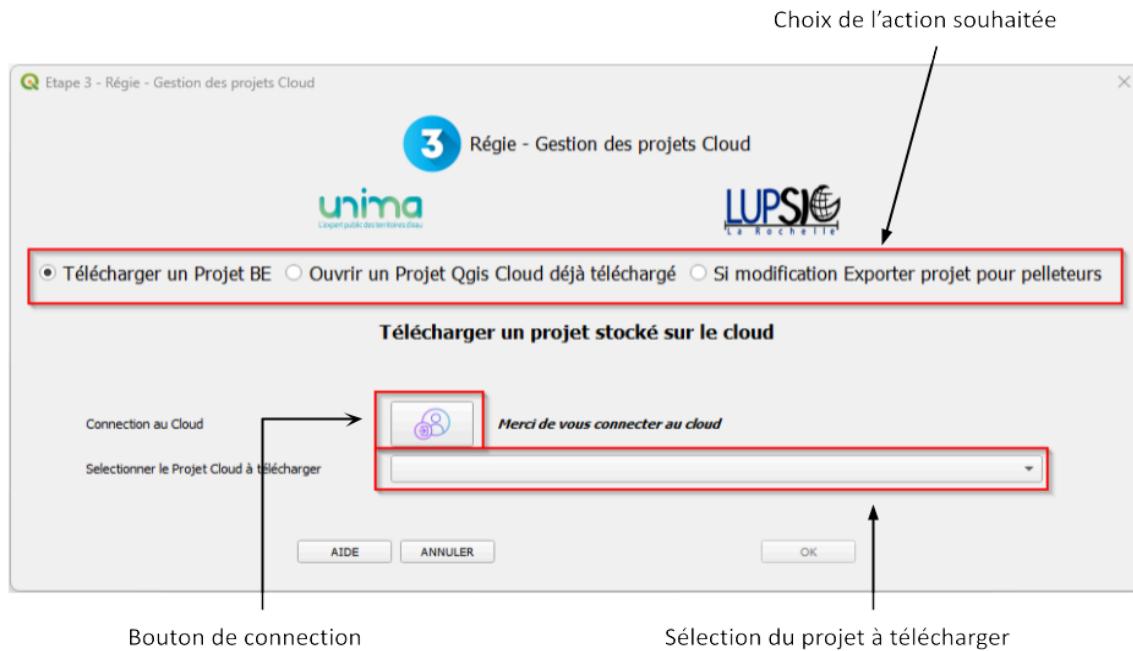
g. Le programme complet

Le programme complet concernant cette deuxième partie est disponible en annexe¹²

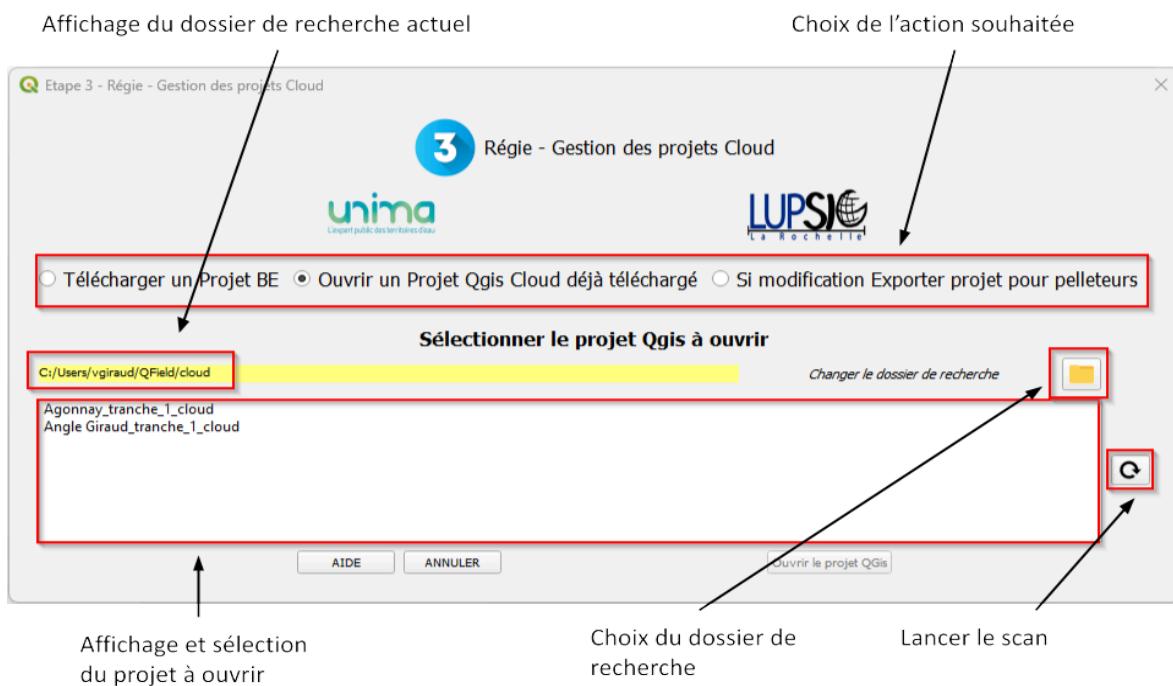
¹² [Lien vers l'annexe](#)

2.5.3 Conducteurs de travaux : Gestion de projet

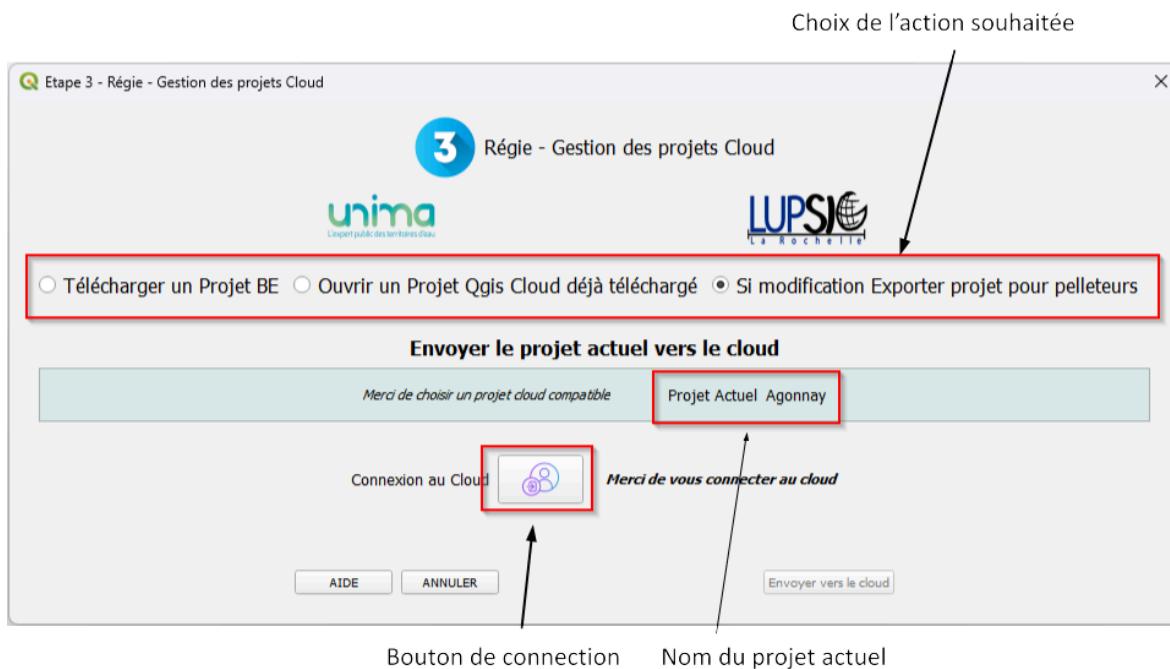
A. L'interface utilisateur



Interface utilisateur de l'étape 3 - Télécharger un projet depuis le cloud non présent sur la machine



Interface utilisateur de l'étape 3 - Ouvrir un Projet QFieldCloud déjà téléchargé



Interface utilisateur de l'étape 3 - Mettre à jour le projet sur le cloud

L'interface est découpée en trois actions distinctes, chacune répondant à une problématique précise:

- ❖ Télécharger un projet depuis le cloud
- ❖ Ouvrir un projet QFieldCloud déjà présent sur la machine
- ❖ Mettre à jour un projet QFieldCloud stocké sur le cloud

B. Les problématiques

a. Télécharger un projet depuis le cloud

Les conducteurs de travaux doivent examiner le plan de la tranche de travaux avant de commencer. Ils peuvent télécharger le projet stocké sur le cloud pour cette consultation.

Lorsqu'un projet n'est pas encore stocké localement, le plugin QFieldSync détecte cette absence et facilite le processus en initiant automatiquement le téléchargement du projet au bon endroit.

Note

Le plugin QFieldSync nécessite une arborescence spécifique pour fonctionner correctement. Il doit stocker les projets QGIS à un chemin d'accès précis et créer des sous-dossiers contenant des projets de sauvegarde. Cette architecture est complexe et demande des fichiers propres à QFieldSync. En utilisant les options de téléchargement natif de QFieldSync, nous garantissons que le projet stocké sur le cloud est téléchargé au bon endroit, avec toutes ses dépendances.

b. Se connecter à un compte QFieldCloud

Il est d'abord essentiel de se connecter à un compte QFieldCloud, où les projets sont stockés. Le plugin utilise une classe spécifique et appelle une méthode dédiée pour ouvrir l'interface de connexion nécessaire à cet accès.

[Lien vers le code](#)

Il est primordial de vérifier la réussite de la connexion une fois établie. Pour ce faire, la première modification apportée au plugin QFieldSync consiste à créer un signal lors de la fermeture de l'interface de connexion. Ce signal sera intercepté par le plugin recolement_unima pour tester la connexion. Du côté de recolement_unima, la réception du signal est gérée de la manière suivante

```
self.cloud_login_dialog.loginFinished.connect(self.login_signal_recu)
@pyqtSlot()
def login_signal_recu(self):
    QTimer.singleShot(3000, self.peupleComboBoxProjets)
    QTimer.singleShot(3000, self.connection_status)
```

Quant à la création du signal du côté de QFieldSync, elle est implémentée de la manière suivante.

```
class CloudLoginDialog(QDialog, CloudLoginDialogUi):
    instance = None
    loginFinished = pyqtSignal()
    # Reste du code #
```

```

def on_login_finished(self) -> None:
    # Reste du code #

    self.done(QDialog.Accepted)
QTimer.singleShot(5000, lambda: (print("Login finished signal emitted"), self.loginFinished.emit()))

```

En gras soulignées sont indiquée les lignes ajoutées pour émettre le signal.

Il convient de noter l'utilisation de `QTimer.singleShot()`, qui permet de retarder l'appel à certaines fonctions. Ce délai est nécessaire pour permettre au plugin QFieldSync de se connecter correctement au compte QFieldCloud de l'utilisateur. Sans cette attente, la suite du programme pourrait s'exécuter avant que le compte soit correctement instancié dans la session QGIS en cours, entraînant ainsi des dysfonctionnements du plugin.

c. Téléchargement d'un projet QFieldCloud pas encore présent sur la machine

Une fois la connexion établie, le conducteur de travaux sélectionne le projet stocké sur le cloud à partir d'une combobox dédiée, puis appuie sur "OK". Une fenêtre de QFieldSync s'ouvre alors avec le nom du projet passé en paramètre. QFieldSync détecte l'absence du projet localement et procède à son téléchargement complet et correct.

d. Ouvrir un projet QFieldCloud déjà présent sur la machine

Pour simplifier l'utilisation du plugin par les utilisateurs inexpérimentés, une fonctionnalité permettant d'ouvrir des projets QGIS directement depuis le plugin a été ajoutée. Cette fonctionnalité était nécessaire pour clarifier l'utilisation des projets QGIS, QFieldCloud compatibles(lié à l'architecture des dossiers de sauvegarde). L'ajout de cette fonctionnalité élimine toute ambiguïté et rend l'expérience des utilisateurs beaucoup plus facile.

[Lien vers le code](#)

Cette fonctionnalité est basée sur un autre plugin développé vu pendant les cours de pyQGIS de cette année. Bien qu'il comporte quelques modifications mineures, sa structure reste essentiellement la même.

Une particularité à noter est l'utilisation d'un slicing qui garantit l'exclusion correcte des répertoires lors du parcours avec `os.walk()`. Cette méthode est cruciale en raison du fonctionnement de QFieldSync, où plusieurs projets de sauvegarde peuvent résider

dans chaque dossier. Ces projets ne sont pas pris en compte lors des mises à jour vers le cloud¹³. Ainsi, le plugin s'assure que l'utilisateur ouvre le projet approprié.

e. Mettre à jour un projet QFieldCloud stocké sur le cloud

Afin de garantir la cohérence entre les modifications apportées par le conducteur de travaux sur le terrain et le projet stocké dans le cloud, le plugin QFieldSync assure la synchronisation des changements. Cependant, pour simplifier l'utilisation de cette fonctionnalité, le plugin Récolement Unima contourne les pages de configuration complexes de QFieldSync. Ce dernier propose en effet deux options de synchronisation : la mise à jour du projet cloud en fonction du projet sauvegardé sur la machine, ou inversement.

Pour éviter à l'utilisateur de faire un choix potentiellement erroné, le plugin Récolement Unima appelle automatiquement les fonctions adéquates de QFieldSync en utilisant des paramètres spécifiques. Ces paramètres, créés spécialement pour cette utilisation, permettent d'automatiser la synchronisation dans le sens le plus cohérent avec le contexte de travail. Il s'agit de la deuxième modification majeure apportée au plugin QFieldSync.

Lien vers le code

Appel de la méthode `show_transfer_dialog()` de la classe `CloudTransferDialog` avec le paramètre `preference_action='local'`, depuis le plugin Récolement Unima.

```
class CloudTransferDialog(QDialog, CloudTransferDialogUi):
    @staticmethod
    def show_transfer_dialog(
        network_manager: CloudNetworkAccessManager,
        cloud_project: CloudProject = None,
        accepted_cb: Callable = None,
        rejected_cb: Callable = None,
        parent: QWidget = None,
        preference_action = None,      # Ajout Victor
    ):
        #Reste du code#
        def __init__(self,
                     network_manager: CloudNetworkAccessManager,
                     cloud_project: CloudProject = None,
                     parent: QWidget = None,
                     preference_action = None,  # Ajout Victor
        )
```

¹³ Le plugin QFieldSync créer des projets de backup pour son utilisation personnelle. Ces projets ne doivent pas être modifiés.

```

self.filesPageOpen.connect(lambda: self._on_filesPageOpen(preference_action)) # Ajout Victor

def _on_filesPageOpen(self,preference_action): # Ajout Victor
    if preference_action == 'cloud':
        self._on_prefer_cloud_button_clicked()
        self._start_synchronization()
    elif preference_action == 'local':
        self._on_prefer_local_button_clicked()
        self._start_synchronization()
    else:
        return

```

En gras sont indiquées les modifications apportées à la classe `CloudTransferDialog` du plugin QFieldSync.

L'ajout d'un argument optionnel au constructeur de la classe, par défaut défini à `None`, permet de préserver les fonctionnalités de base du plugin QFieldSync. Lorsque cet argument est renseigné, comme c'est le cas dans le plugin Récolelement Unima, il fournit des informations supplémentaires qui simulent l'activation de boutons de configuration spécifiques. Ces informations permettent d'indiquer à QFieldSync le comportement attendu pour la synchronisation des données.

f. Le programme complet

Le programme complet concernant cette troisième partie du plugin Récolelement Unima est disponible en annexe¹⁴.

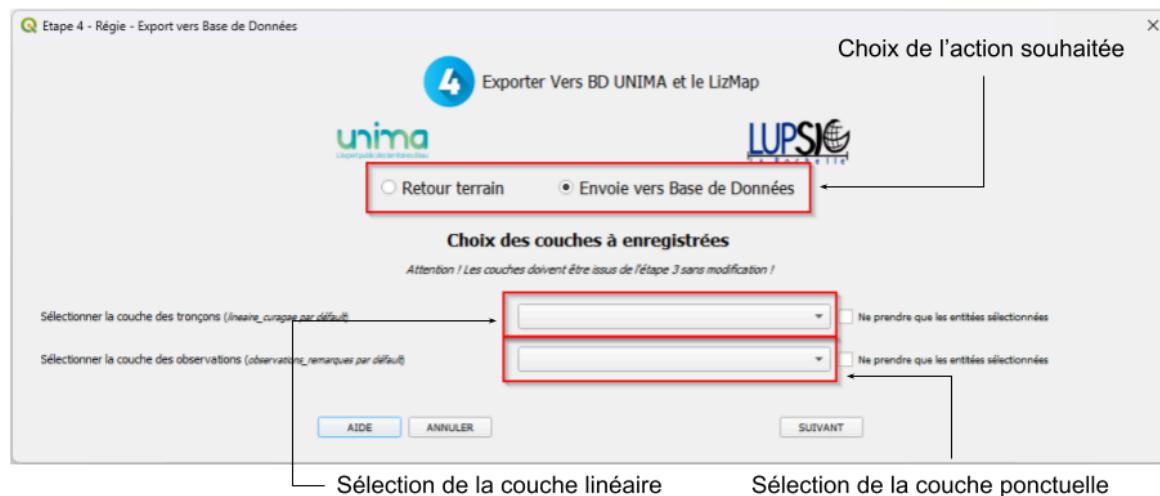
¹⁴ [Lien vers l'annexe](#)

2.5.4 Conducteurs de travaux : Gestions des données

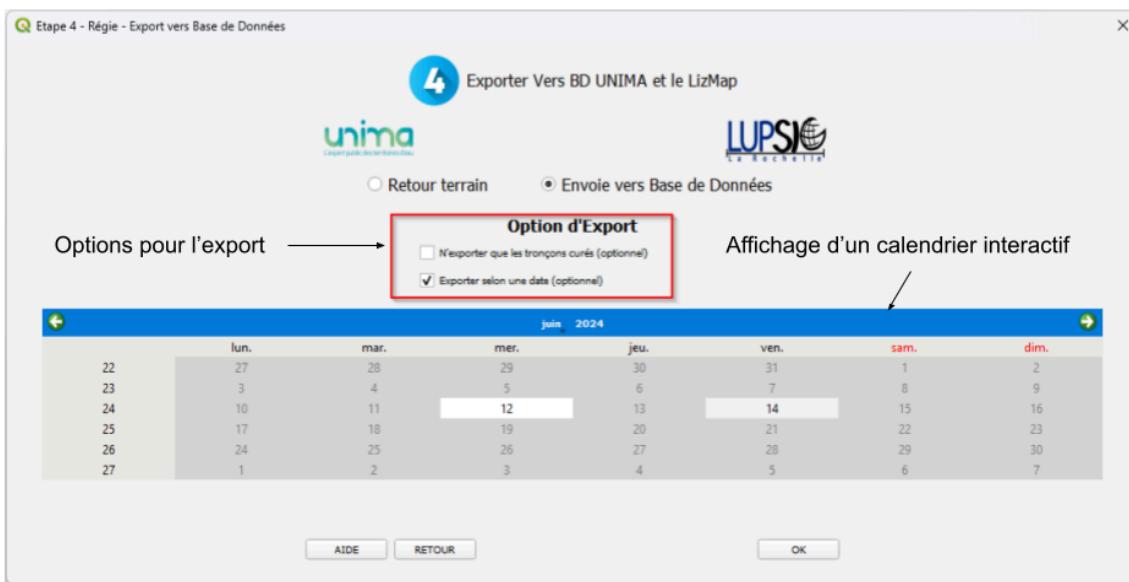
A. L'interface utilisateur



Interface utilisateur de l'étape 4 - Télécharger un projet depuis le cloud déjà présent sur la machine



Interface utilisateur de l'étape 4 - Envoyer vers la base de données - Page 1



Interface utilisateur de l'étape 4 - Envoyer vers la base de données - Page 2

B. Les problématiques

- Récupérer un projet cloud modifié par les pelleteurs

Avant d'envoyer les informations dans la base de données, le conducteur de travaux doit vérifier le contenu du projet renseigné par les pelleteurs. Il peut de cette manière

- Vérifier l'avancée des travaux
- Vérifier la qualité de la donnée
- Corriger les données reçues
- Couplet avec l'étape 3, informer les pelleteurs de modification ou de points de vigilance.

Pour se faire il doit mettre à jour son projet QGIS en fonction des modifications apportées par les pelleteurs.

Avec les mêmes problématiques que lors de la mise à jour du projets cloud ([2.5.3.a](#)) le plugin Récolelement Unima permet de passer outre un paramétrage hasardeux, menant à des erreurs, en forçant le plugin à appliquer les modifications du projet stockée dans le cloud

[Lien vers le code](#)

Pour ce faire, la méthode `show_transfer_dialog()` de la classe `CloudTransferDialog` est appelée avec le paramètre `preference_action='cloud'`, depuis le plugin Récolement Unima.

- b. Exporter les données linéaire vers le schéma `régie_travaux` de la BD UNIMA.

Une fois les données validées par le conducteur de travaux, il peut exporter les données vers le schéma Postgre dédié à la Régie travaux de l'Unima en utilisant la fonctionnalité d'exportation. Cette fonctionnalité offre plusieurs options :

- Exporter uniquement les données sélectionnées : Cette option permet de cibler l'exportation sur des données spécifiques, définies par l'utilisateur.
- Exporter uniquement les tronçons curés : Cette option permet de limiter l'exportation aux tronçons de travaux ayant fait l'objet d'un curage.
- Exporter les données d'une date ou entre deux dates : Cette option permet de filtrer les données à exporter en fonction d'une période temporelle définie. Pour faciliter la sélection de cette période, un calendrier est proposé. Ce calendrier dispose de fonctionnalités avancées :
 - Sélection d'une plage de dates : L'utilisateur peut sélectionner deux dates pour définir une plage d'exportation précise. Cette fonctionnalité n'est pas native du widget QCalendar standard.
 - Affichage des dates valides : Le calendrier affiche uniquement les dates réellement présentes dans les tables de données ponctuelles et linéaires. Cette restriction garantit la cohérence des données exportées et offre une meilleure expérience utilisateur

[Lien vers le code](#)

- c. Récupérations des informations de la table de linéaire

La table linéaire va être insérée dans la base de données dans deux tables différentes, une table pour la géométrie et l'id du réseau, et une pour les informations liées à l'historisation des travaux. Ce choix répond à plusieurs impératifs.

Priorité à l'historisation des travaux : En effet, l'outil de récolement d'information a pour objectif principal de retracer l'historique des interventions sur les réseaux. La

conservation de cet historique dans une table dédiée permet une meilleure gestion et une exploitation facilitée de ces données.

Adaptation à des tronçons partiellement curés : En cours de développement, il a été constaté que certains tronçons de réseau n'étaient pas curés sur leur totalité, nécessitant un découpage de la géométrie. Pour répondre à ce cas de figure et communiquer aux partenaires de l'Unima les zones exactes d'intervention, le découpage géométrique est également historisé.

Communication précise des zones curées : Cette approche permet de communiquer de manière précise les tronçons de réseaux effectivement curés, en dissociant les parties traitées de celles qui ne le sont pas. Cela s'avère particulièrement utile pour les réseaux de grande envergure, où le statut "curé" ne s'applique pas à l'intégralité du linéaire.

En résumé, la scission de la table linéaire en deux tables distinctes vise à privilégier l'historisation des travaux, à s'adapter à des géométries changeantes et à communiquer des informations précises aux partenaires.

Lien vers le code

Dans ce code les informations de la table linéaire sont récupérées et un INSERT ou un UPDATE sont réalisés dans le schéma régie_travaux de la BD UNIMA. Il est à notifier l'utilisation de la bibliothèque psycopg2 qui permet, entre autres, de mieux gérer les erreurs sql, ou encore d'éviter de potentielles injections SQL.

d. Vérification de l'existence d'une entité

Il faut cependant savoir s'il faut mener une opération UPDATE ou INSERT pour chaque entité. Pour se faire, une vérification est effectuée pour les tables concernées. Une pour la table reseau_travaux (où est stockée la géométrie et l'id du tronçon) et une pour la table historisation_travaux (où seront stockées les informations liées à l'historisation des travaux de curage). Ces vérifications effectuent des requêtes SQL pour vérifier si des entités sont déjà existantes dans les tables correspondantes.

Lien vers le code

Méthode pour la table reseau_travaux:

- Cette méthode vérifie si une entité avec le même `res_code_tr` et la même géométrie (`rt_geom`) existe déjà dans la table `reseau_travaux`.
- Si une entité correspondante est trouvée, son identifiant (`rt_id`) est renvoyé.
- Si aucune entité correspondante n'est trouvée, `None` est renvoyé.

Méthode pour la table historisation_travaux:

- Cette méthode vérifie si une entité avec le même `hit_num_tranche_be` et le même `rt_id` existe déjà dans la table `historisation_travaux`.
- Si une entité correspondante est trouvée, son identifiant (`hit_id`) est renvoyé.
- Si aucune entité correspondante n'est trouvée, `None` est renvoyé.

En fonction des résultats de ces méthodes, le code exécute les requêtes SQL appropriées :

- Table `reseau_travaux` :
 - Si une entité correspondante est trouvée (`existing_rt_id` non `None`), une requête `UPDATE` est exécutée pour mettre à jour les informations de l'entité existante.
 - Si aucune entité correspondante n'est trouvée (`existing_rt_id` est `None`), une requête `INSERT` est exécutée pour créer une nouvelle entité.
 - Table `historisation_travaux` :
 - Si une entité correspondante est trouvée (`existing_entity_id` non `None`), une requête `UPDATE` est exécutée pour mettre à jour les informations de l'entité existante.
 - Si aucune entité correspondante n'est trouvée (`existing_entity_id` est `None`), une requête `INSERT` est exécutée pour créer une nouvelle entité.
- e. Exporter les données ponctuelles vers le schéma `régie_travaux` de la BD UNIMA.

Les mêmes opérations sont effectuées pour l'exportation des données ponctuelles dans la BD UNIMA. L'existence ou non de l'entité dans la BD est d'abord vérifié (via sa géométrie et le code du tronçon associé) puis on effectue au choix un `INSERT` ou un `UPDATE` dans la BD.

[Lien vers le code](#)

- f. Export des potentielles photo vers le serveurs de stockage de l'Unima

Les pelleteurs ont la possibilité d'enregistrer des photos lors de la création de ponctuel de la couche `observations_remarques`. Ces photos doivent être enregistrées dans un serveur de l'Unima afin d'être accessibles à tout le monde pour une utilisation future, notamment lors de la création de rapport ou de compte rendu.

[Lien vers le code](#)

Dans ce code est utilisée la bibliothèque `shutil`. Elle permet de réaliser de nombreuses opérations sur les fichiers. Son utilisation ici permet de copier les fichiers photo

contenus dans l'architecture du projet QGIS, hérité de l'architecture de QField, dans un répertoire précis d'un serveur de l'Unima.

g. Le programme complet

Le programme complet concernant cette dernière partie du plugin est disponible en annexe¹⁵.

¹⁵ [Lien vers l'annexe](#)

2.6 Paramétrage des tablettes

Il est maintenant nécessaire de paramétrier les tablettes pour limiter leur utilisation à un cadre strictement professionnel. Étant attribuées individuellement aux pelleteurs, ceux-ci les auront en permanence avec eux. Afin d'éviter une dégradation du matériel, ou pour un usage personnel, un contrôle sera mis en place.

Il existe sur Android un moyen simple de créer des utilisateurs et de restreindre leurs profils sur des tablettes. Cependant, les tablettes acquises par l'Unima possèdent des puces 5G et sont donc considérées par le système Android comme des téléphones, et non des tablettes. L'alternative consiste à utiliser un contrôle parental. Android propose un contrôle parental complet et natif en partenariat avec Google, appelé "Family Link".

L'initialisation du Family Link nécessite l'utilisation d'un compte parental Google, qui peut être vu comme le compte superviseur. Un compte Google a donc été créé pour répondre à cette demande. Une fois le compte superviseur connecté, le paramétrage permet de restreindre les tablettes à toutes les applications sauf l'appareil photo, la géolocalisation et évidemment QField.

Il est à noter qu'un tuto a été écrit à destination des administrateurs de la solution afin de détailler toutes les manipulations. ([VOIR ANNEXE](#))

2.7 Initialisation des CronTable

Une fois les informations enregistrées en base de données, elles doivent être sécurisées puis envoyées dans la base de données *Lizmap* pour être utilisées dans le WebSIG de l'Unima. Ces transferts de données sont effectués périodiquement et de manière automatisée grâce à l'utilisation de cron-tables.

Note

Comme vu plus tôt, les données issues du récolement de l'information sont envoyées dans un schéma dont la Régie de Travaux possède tous les droits. Ces personnes, non formées à l'administration d'une base de données, peuvent commettre des erreurs et compromettre l'intégrité du schéma et des données.

C'est dans ce contexte que les données sont transférées dans un schéma spécifique, administré par le pôle SIG, pour y être sécurisées et archivées.

2.7.1 Cron Table et Fichier Bash

Les cron tables et les fichiers bash (.sh) sont des outils essentiels pour automatiser des tâches sur les systèmes Unix/Linux.

A. Fichier Bash

Un fichier bash est un script contenant une série de commandes à exécuter par l'interpréteur de commandes Bash. Ces fichiers sont utilisés pour automatiser des tâches répétitives.

Fonctionnement

- ➔ **Création du fichier :** Les fichiers bash sont créés avec une extension `.sh`.
- ➔ **Écriture des commandes :** Les commandes sont écrites dans le fichier avec une syntaxe Bash.
- ➔ **Rendre le fichier exécutable :** Le fichier doit être rendu exécutable avec la commande `chmod +x nom_du_fichier.sh`.
- ➔ **Exécution du fichier :** Le script est exécuté en utilisant `./nom_du_fichier.sh` ou en le planifiant dans une crontab.

B. Les CronTab

Cron est un service Unix/Linux utilisé pour exécuter des scripts ou des commandes à des moments spécifiques de manière récurrente. Les tâches à exécuter sont définies dans un fichier de configuration appelé crontab.

Fonctionnement

- ➔ **Édition de la crontab :** Avec la commande `crontab -e`.

- ➔ Syntaxe de la crontab : Chaque ligne de la crontab représente une tâche planifiée avec la syntaxe suivante :
- ```
sql
```

`minute heure jour_du_mois mois jour_de_la_semaine commande`

#### Note

minute : Minute de l'heure (0-59)  
 heure : Heure de la journée (0-23)  
 jour\_du\_mois : Jour du mois (1-31)  
 mois : Mois de l'année (1-12)  
 jour\_de\_la\_semaine : Jour de la semaine (0-7, où 0 et 7 représentent dimanche)  
 commande : La commande ou le script à exécuter

## 2.7.2 Transfert des données du schéma de la régie travaux vers le schéma d'historisation

Dans ce contexte de sécurisation des données, un transfert est effectué tous les jours à minuit des données du schéma *régie\_travaux* vers le schéma *historisation\_curage*. Ces deux schémas se trouvant dans la même base de données et sur le même serveur, le transfert de données se fera via une requête SQL.

De plus, la base de données est sauvegardée en .gz<sup>16</sup>. Les sauvegardes ainsi effectuées resteront 30 jours sur le serveur avant d'être automatiquement supprimées.

### A. Contenu du fichier Bash

Exemple de contenu du fichier bash :

```
#!/bin/bash

DB_USER="anonymisé"
DB_HOST="anonymisé"
PASSWORD="anonymisé"
DB_PORT="anonymisé"
DB_NAME="anonymisé"

BACKUP_DIR="/root/backup_db"
```

<sup>16</sup> L'extension .gz indique un fichier compressé à l'aide de l'algorithme de compression GNU Zip (gzip).

```

Nom du fichier de sauvegarde
BACKUP_FILE="regie_travaux_backup_$(date +'%Y%m%d_%H%M%S').sql.gz"

Sauvegarde du schéma regie_travaux
PGPASSWORD=$PASSWORD pg_dump -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME -n regie_travaux | gzip
> "$BACKUP_DIR/$BACKUP_FILE"

Requête SQL pour copier les données
SQL_QUERY="

DO $$

DECLARE
 rec RECORD;

BEGIN
 -- Copier la table 'historisation_observation'
 FOR rec IN (SELECT * FROM regie_travaux.historisation_observation) LOOP
 INSERT INTO historisation_curage.historisation_observation (col1, col2, col3)
 VALUES (rec.col1, rec.col2, rec.col3)
 ON CONFLICT (primary_key_column)
 DO UPDATE SET
 col1 = EXCLUDED.col1,
 col2 = EXCLUDED.col2,
 col3 = EXCLUDED.col3;
 END LOOP;
END $$;
répéter pour les autres tables
"

Exécuter la requête SQL
PGPASSWORD=$PASSWORD psql -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME -c "$SQL_QUERY"

Nettoyer anciens backups de plus de 60 jours
find $BACKUP_DIR/* -mtime +30 -exec rm {} \;

```

L'intégralité du code en annexe est disponible en annexe.([VOIR ANNEXE](#))

### B. Explication de la requête SQL

➔ Déclaration du Bloc PL/pgSQL:

DO \$\$

Déclaration d'une Variable de Type RECORD

```

DECLARE
 rec RECORD;

```

'DECLARE' Commence une section où les variables sont déclarées.

'rec RECORD;' Déclare une variable nommée rec de type RECORD, qui peut contenir une ligne de n'importe quelle table.

Début du Bloc de Code:

```
BEGIN
```

Boucle pour Chaque Enregistrement de la Table Source:

```
FOR rec IN (SELECT * FROM regie_travaux.historisation_observation) LOOP
```

'FOR rec IN (SELECT \* FROM regie\_travaux.historisation\_observation)' Itère sur chaque enregistrement de la table historisation\_observation dans le schéma regie\_travaux.

'LOOP' Commence la boucle qui va traiter chaque enregistrement individuellement.

➔ Insertion ou Mise à Jour des Enregistrements:

```
INSERT INTO historisation_curage.historisation_observation (col1, col2, col3)
VALUES (rec.col1, rec.col2, rec.col3)
ON CONFLICT (primary_key_column)
DO UPDATE SET
 col1 = EXCLUDED.col1,
 col2 = EXCLUDED.col2,
 col3 = EXCLUDED.col3;
```

**INSERT INTO** historisation\_curage.historisation\_observation (col1, col2, col3) indique les colonnes où doivent se faire l'insertion des valeurs

**VALUES** (rec.col1, rec.col2, rec.col3) indique les valeurs à insérer.

**ON CONFLICT** (primary\_key\_column) indique que si un conflit de clé primaire survient (par exemple, si un enregistrement avec la même clé primaire existe déjà), l'opération suivante sera exécutée.

**DO UPDATE SET** col1 = EXCLUDED.col1, col2 = EXCLUDED.col2, col3 = EXCLUDED.col3 indique que si un conflit se produit, l'enregistrement existant sera mis à jour avec les valeurs importées et non pas existentes. EXCLUDED fait référence aux valeurs qui ont causé le conflit.

Fin de la Boucle:

```
END LOOP;
```

Fin du Bloc PL/pgSQL:

END \$\$;

## Note

PL/pgSQL est un langage procédural spécifique à PostgreSQL, conçu pour écrire des fonctions et des procédures stockées à l'intérieur de la base de données. PL/pgSQL est basé sur le langage PL/SQL d'Oracle, mais avec quelques différences syntaxiques et fonctionnelles.

Les principales caractéristiques de PL/pgSQL sont les suivantes :

- Procedural Language : PL/pgSQL est un langage procédural, ce qui signifie qu'il permet d'écrire des instructions qui sont exécutées les unes après les autres, souvent pour automatiser des tâches ou mettre en œuvre des logiques complexes.
- Support des Variables et des Structures de Contrôle : PL/pgSQL prend en charge les variables, les structures de contrôle (comme les boucles et les conditions) et les fonctions.
- Interaction Directe avec la Base de Données : Les scripts PL/pgSQL peuvent effectuer des opérations directement sur la base de données, comme des requêtes SELECT, INSERT, UPDATE ou DELETE.
- Gestion des Transactions : Les transactions SQL peuvent être gérées à l'intérieur des scripts PL/pgSQL, permettant un contrôle plus fin sur les opérations effectuées dans la base de données.
- Traitement des Exceptions : PL/pgSQL permet de capturer et de gérer les exceptions, ce qui facilite la gestion des erreurs dans les scripts.

## C. Comportement de la Requête

La requête copie toutes les lignes de la table `historisation_observation` du schéma `regie_travaux` vers la table `historisation_observation` du schéma `historisation_curage`.

Si une ligne avec la même clé primaire existe déjà dans la table cible, elle est mise à jour avec les nouvelles valeurs de la ligne importée.

Cette approche permet de synchroniser les deux tables tout en évitant les conflits de clés primaires (réaliser un UPDATE lorsque qu'un enregistrement existe déjà).

## D. Contenu du CronTab

```
0 0 * * * root/export_regie_historisation.sh # tout les jour à 00h
```

### 2.7.3 Transfert des données du schéma d'historisation vers la BD LizMap

Il est maintenant nécessaire de transférer ces données vers la base de données *LizMap*, afin qu'elle puisse être utilisée dans la construction du projet QGIS qui servira pour le WebSig de type *Lizmap*.

Les données nécessitant d'être transférées d'une base de données à une autre, il est nécessaire de réaliser un dump du schéma *historisation\_curage* pour ensuite le restaurer dans la base de données *LizMap*.

Ce transfert est effectué toutes les semaines (les dimanches à minuit). De plus, la base de données est sauvegardée en .gz. Les sauvegardes ainsi effectuées resteront 60 jours sur le serveur avant d'être automatiquement supprimées.

#### E. Contenu du fichier Bash

Contenu du fichier bash :

```
#!/bin/bash

Chemin vers le fichier pgpass
PGPASSFILE="/root/.pgpass"

Informations de connexion pour la base de données UNIMA
DB_USER="anonymisé"
DB_HOST="anonymisé"
PASSWORD="anonymisé"
DB_PORT="anonymisé"
DB_NAME_UN="anonymisé"

Informations de connexion pour la base de données LIZMAP
LIZMAP_DB_NAME=$(grep "lizmap.com:5432" $PGPASSFILE | awk -F: '{print $3}')
LIZMAP_DB_HOST=$(grep "lizmap.com:5432" $PGPASSFILE | awk -F: '{print $1}')
LIZMAP_DB_PORT=$(grep "lizmap.com:5432" $PGPASSFILE | awk -F: '{print $2}')
LIZMAP_DB_USER=$(grep "lizmap.com:5432" $PGPASSFILE | awk -F: '{print $4}')
LIZMAP_DB_PASSWORD=$(grep "lizmap.com:5432" $PGPASSFILE | awk -F: '{print $5}')

BACKUP_DIR="/root/backup_db"

Nom du fichier de sauvegarde
BACKUP_FILE="historisation_curage_backup_$(date +'%Y%m%d_%H%M%S').sql.gz"

Sauvegarde du schéma historisation_curage de la base UNIMA avec suppression des objets existants lors de la restauration des données.
PGPASSWORD=$PASSWORD pg_dump -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME_UN -n historisation_curage --clean | gzip > "$BACKUP_DIR/$BACKUP_FILE"

Import du schéma dans la base de données LIZMAP
Utiliser la commande zcat pour décompresser et restaurer les données dans le schéma
```

```
historisation_curage de la base LIZMAP
zcat "$BACKUP_DIR/$BACKUP_FILE" | PGPASSWORD=$LIZMAP_DB_PASSWORD psql -h
$LIZMAP_DB_HOST -p $LIZMAP_DB_PORT -U $LIZMAP_DB_USER -d $LIZMAP_DB_NAME

Nettoyer anciens backups de plus de 30 jours
find $BACKUP_DIR/* -mtime +30 -exec rm {} \;
```

L'intégralité du code est disponible en annexe.([VOIR ANNEXE](#))

## F. Explication de la requête SQL

- ➔ Configuration des chemins et des informations de connexion :

`PGPASSFILE="/root/.pgpass"`

Définition du chemin vers le fichier `pgpass` contenant les mots de passe.

### Note - Le fichier Pgpass

Le fichier `.pgpass` stocke de manière sécurisée les informations de connexion aux bases de données PostgreSQL.

Chaque ligne du fichier contient les paramètres de connexion suivants, séparés par des deux-points : hôte, port, base de données, utilisateur et mot de passe. Lorsqu'un outil PostgreSQL, comme `psql` ou `pg_dump`, est exécuté, il utilise ces informations pour se connecter sans demander de mot de passe. Son avantage principal est de garder les mots de passe hors des scripts, réduisant le risque d'exposition des informations sensibles.

- ➔ Extraction des informations de connexion pour LizMap à partir de `pgpass` :

`LIZMAP_DB_HOST=$(grep "lizmap.com:5432" $PGPASSFILE | awk -F: '{print $1}'")`

`grep` et `awk` sont des commandes utilisées pour rechercher et traiter des données textuelles dans des fichiers.

- `grep` : Cette commande recherche des lignes contenant une chaîne de caractères spécifique. Par exemple, `grep "lizmap.com:5432" $PGPASSFILE`

recherche la ligne contenant "lizmap.com:5432" dans le fichier désigné par **\$PGPASSFILE**.

- awk : Cette commande est utilisée pour traiter et analyser des données textuelles. Dans le script, `awk -F: '{print $3}'` indique que les champs sont séparés par des deux-points (-F:) et extrait le troisième champ (`{print $3}`). Ainsi, `awk` extrait les informations spécifiques de la ligne trouvée par `grep`.

- Définition du répertoire de sauvegarde et du nom du fichier :

```
BACKUP_DIR="/root/backup_db"
BACKUP_FILE="historisation_curage_backup_$(date +'%Y%m%d_%H%M%S').sql.gz"
```

Création d'un fichier de sauvegarde nommé avec un timestamp.

- Sauvegarde et compression :

```
PGPASSWORD=$PASSWORD pg_dump -h $DB_HOST -p $DB_PORT -U $DB_USER -d $DB_NAME_UN -n historisation_curage --clean | gzip > "$BACKUP_DIR/$BACKUP_FILE"
```

Exécution de `pg_dump` pour sauvegarder le schéma `historisation_curage` de la base UNIMA, en le compressant avec `gzip`.

L'option `--clean` dans `pg_dump` ajoute des commandes pour supprimer les objets existants avant de restaurer les données. Cela garantit que la base de données de destination est mise à jour avec une copie propre des données exportées. Ainsi, les anciennes données ou structures sont supprimées pour éviter les conflits.

- Importation dans la base LizMap :

Utilisation de `zcat` pour décompresser le fichier et `psql` pour restaurer les données dans la base LizMap.

- Nettoyage des anciens backups :

Suppression des fichiers de sauvegarde dans le répertoire de sauvegarde datant de plus de 30 jours.

## G. Comportement de la Requête

Ce script automatise la sauvegarde du schéma `historisation_curage` depuis la base de données UNIMA vers la base de données LIZMAP. Il extrait les informations de connexion pour LIZMAP à partir d'un fichier `.pgpass` sécurisé. Ensuite, il utilise

`pg_dump` avec l'option `--clean` pour sauvegarder le schéma, en supprimant les objets existants lors de la restauration. La sauvegarde est compressée en utilisant `gzip` et stockée dans un répertoire défini par `BACKUP_DIR`. Ensuite, la sauvegarde est restaurée dans LIZMAP en utilisant `zcat` pour décompresser et `psql` pour importer les données. Enfin, le script nettoie les anciennes sauvegardes après 30 jours.

#### H. Contenu du CronTab

```
0 0 * * 0 root/export_historisation_lizmap.sh # toute les semaines (dimanche) à 00h
```

### 2.8 Mise en place du LizMap

Il ne reste maintenant qu'à mettre à disposition des partenaires de l'Unima, en consultation et en téléchargement, les données récoltées et archivées. L'Unima a un partenariat avec le groupe 3liz pour l'hébergement et la maintenance d'un serveur LizMap. Un projet QGIS est créé puis, à l'aide de l'extension LizMap, est converti en projet LizMap. Une fois le transfert des fichiers sur le serveur LizMap effectué, le projet sera terminé et opérationnel.

#### 2.8.1 Contenu attendu du WebSig

Une réflexion sur les types de données à afficher dans le WebSiga d'abord été mené, car toutes les données collectées ne sont pas pertinentes pour les partenaires. Les données à transmettre ont été rapidement identifiées.

Pour les données ponctuelles, les travaux annexes, les travaux de confortement de berge<sup>17</sup>, et l'installation de pas busés<sup>18</sup> définitifs furent retenus.

Pour les données linéaires, il a été décidé de diviser les travaux en deux catégories. La première regroupe les travaux strictement liés au curage et les données qui en découlent, telles que les tronçons curés, la date de curage, les mètres linéaires, la hauteur de vase extraite<sup>19</sup>, la largeur du fossé historique<sup>20</sup> et la géométrie des tronçons,

<sup>17</sup> Le confortement des berges est une intervention de renforcement indispensable lorsque des berges présentent des faiblesses sous l'effet de l'érosion.

<sup>18</sup> Installation permettant le passage au dessus d'un canal pour les engins agricole et de chantier

<sup>19</sup> Exprimé en m<sup>3</sup>/m, désigne la quantité de vase extraite du fossé

<sup>20</sup> Les berges d'un canal s'agrandissent sous l'effet de l'érosion et de la biodiversité. La notion de fossé historique ne prends pas en compte cette agrandissement

mises en valeur et classées par date de réalisation. La seconde catégorie, qui comprend les travaux de broyage et de débroussaillage, sera également accessible aux utilisateurs, dans une section spécifique.

De plus, différentes options devront être implémentées dans le WebSig. Cela inclut l'initialisation de pop-ups sur les différentes couches vectorielles, l'impression d'une zone du canvas, et le téléchargement des données.

Il est important de noter que le WebSig devra également contenir les informations des travaux étant effectué depuis 2014, et devra accueillir les données récoltées jusqu'en 2027, date de fin de la majorité des Plan Pluriannuel de Travaux.

### 2.8.2 Importation des données

Les données importées dans le projet QGIS seront extraites du serveur Postgres *LizMap*, hébergé chez 3liz, d'où la nécessité de transférer régulièrement les données, comme mentionné précédemment.

Pour importer ces données, l'utilisation du plugin natif de QGIS, "Gestionnaire BD", sera essentielle. Ce plugin, grâce à sa "Fenêtre SQL", permet d'exécuter des requêtes SQL et d'importer le résultat de ces requêtes en tant que couche dans le projet.

Les données importées sont également découpées en 3 plages temporelles, de 2014 à 2019, de 2021 à 2023, et enfin post 2024.

Les requêtes pour les données produites entre 2014 et 2019, et celles de 2021 à 2023 sont extrait de l'ancien WebSig de l'Unima et ne seront pas traités

#### A. Requêtes des ponctuelles

La couche des données ponctuelles est simple à importer. Ces données, enregistrées en base de données avec leurs géométries, nécessitent seulement une clause WHERE pour filtrer les informations pertinentes. Par exemple, pour extraire les travaux réalisés en 2024, la requête SQL suivante est utilisée :

```
SELECT * FROM historisation_curage.historisation_observation AS hio WHERE hio_date >= '2024-01-01' AND hio_date <= '2024-12-31' AND obs_id IN (4, 5, 7);
```

Cette requête est facile à comprendre : elle extrait uniquement les données datées de 2024 pour les types de travaux spécifiés par les **obs\_id** 4, 5, et 7<sup>21</sup>.

---

<sup>21</sup> Identifiants des travaux annexes, des travaux de confortement de berge, et des pas busés

## B. Requêtes pour le linéaire liées aux curage

La requête pour extraire les données linéaires est plus complexe mais reste accessible. Elle nécessite une jointure entre les couches de géométrie et les données d'historisation pour associer les informations de curage à un linéaire. Ensuite, seuls les champs pertinents sont sélectionnés pour les partenaires. La longueur des tronçons est calculée automatiquement avec la fonction `ST_Length` de PostGIS. Enfin, dans cet exemple, on filtre les travaux pour l'année 2024.

Voici la requête SQL correspondante :

```

SELECT
res.rt_id,
res.res_code_tr,
ROUND(CAST(ST_Length(res.rt_geom) AS numeric), 2) AS longueur,
res.as_n_unima,
res.rt_geom,
hit.hit_id,
hit.hit_tranche,
hit.hit_verif_curage,
hit.hit_date,
hit.hit_hauteur_vase,
hit.hit_largeur_fosse
FROM
historisation_curage.reseau_travaux AS res
INNER JOIN historisation_curage.historisation_travaux AS hit ON res.rt_id = hit.rt_id
WHERE
hit.hit_date >= '2024-01-01' AND hit.hit_date <= '2024-12-31';

```

Il est important de noter l'utilisation d'une jointure de type `INNER JOIN`. En effet, cette jointure permet de renvoyer uniquement les lignes où il y a une correspondance dans les deux tables. Dans ce contexte, cela signifie que seuls les tronçons (de la table `reseau_travaux`) ayant des données d'historisation de travaux (dans la table `historisation_travaux`) seront inclus. Comme chaque tronçon doit obligatoirement avoir des informations de travaux pour être pertinent, l'`INNER JOIN` est approprié.

## C. Requêtes pour le linéaires des travaux annexes

Finalement la même logique est utilisée pour importer les données pour les travaux annexes.

```

SELECT
res.rt_id,
res.res_code_tr,
res.as_n_unima,
res.rt_geom,

```

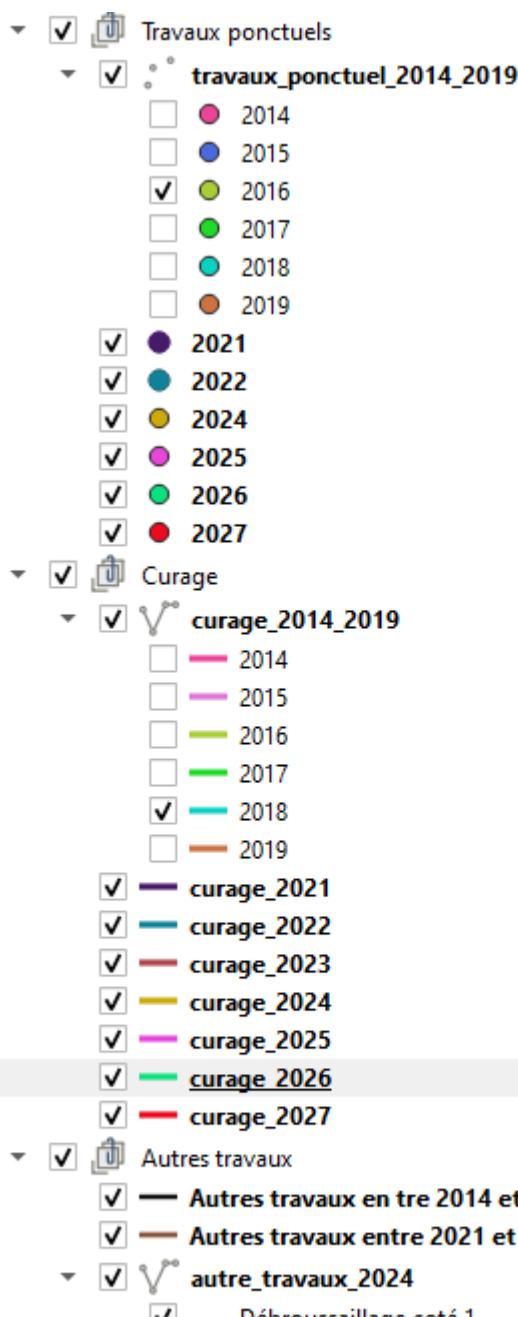
```

hit.hit_id,
hit.hit_tranche,
hit.hit_verif_curage,
hit.hit_date,
hit.hit_num_devis,
hit.hit_aplo_cot,
hit.hit_bro_cot,
hit.hit_temps_broyage,
hit.hit_temps_troncottage
FROM
historisation_curage.reseau_travaux AS res
INNER JOIN historisation_curage.historisation_travaux AS hit ON res.rt_id = hit.rt_id
WHERE
hit_date >= '2024-01-01' AND hit_date <= '2024-12-31'
AND hit_aplo_cot IS NOT NULL
AND hit_bro_cot IS NOT NULL

```

## 2.8.3 Mise en forme des données

### A. La symbologie



Une symbologie a été mise en place pour améliorer la lisibilité de la carte et des informations. Cette symbologie par catégorisation met en valeur les informations selon les dates de réalisation. Les couches ponctuelles et linéaires utilisent une symbologie unique par année, avec des couleurs distinctes pour chaque année.

La couche des "autres travaux" linéaires, en revanche, utilise une symbologie uniforme indépendamment de la date de réalisation, à partir de 2024 et l'implémentation de la solution. Cette couche sert principalement à indiquer les travaux de débroussaillage ou de mise à l'aplomb effectués.

Le travail de symbologie pour ce projet reste simple, car la catégorisation par année des différentes couches est réalisée dès leur importation. L'information est davantage transmise via les pop-ups des couches plutôt que par leur représentation graphique seule.

### B. Les infobulles QGIS

En anticipation du paramétrage du projet LizMap et pour une meilleure cohérence des données, les infobulles QGIS affichent les informations de la table attributaire de manière organisée et agréable.

Ces infobulles sont construites en HTML. Bien que QGIS ne permette pas d'associer un fichier CSS, ce qui rend le travail plus fastidieux, les infobulles peuvent récupérer dynamiquement les valeurs des champs de la table attributaire. De plus, l'utilisation de SQL au sein du HTML permet des opérations logiques pour une personnalisation accrue.

La charte graphique des infobulles de l'Unima a été reprise pour une meilleure cohérence au sein des différentes cartes proposées par l'Unima. Les informations pertinentes seront donc affichées dans un tableau structuré, avec le nom de l'information et sa valeur par lignes du tableau.

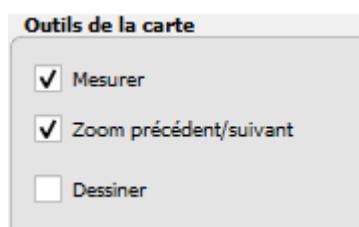
| Curage                   |                                               |
|--------------------------|-----------------------------------------------|
| Type                     | Seconadaire                                   |
| Date de curage           | 06-2024                                       |
| Longueur                 | 1148.24 m                                     |
| Hauteur de vase          | Pas de hauteur renseignée                     |
| Largeur fossé historique | Pas de largeur de fossé historique renseignée |

Le code HTML complet est disponible en Annexe ([VOIR ANNEXE](#))

### 2.8.3 Configuration du projet LizMap

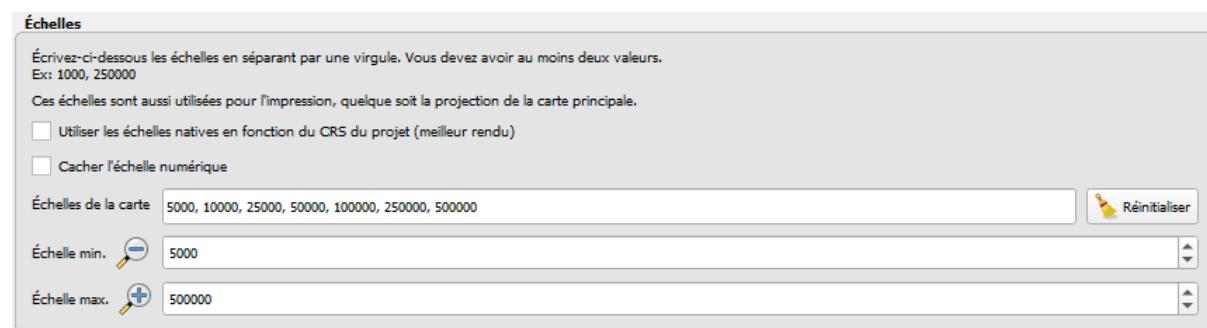
Une fois les couches correctement importées et leurs symbologies terminées, il faut créer le projet LizMap. Pour cela l'extension de 3liz permet de grandement faciliter la création d'un projet.

## A. Les options de la cartes



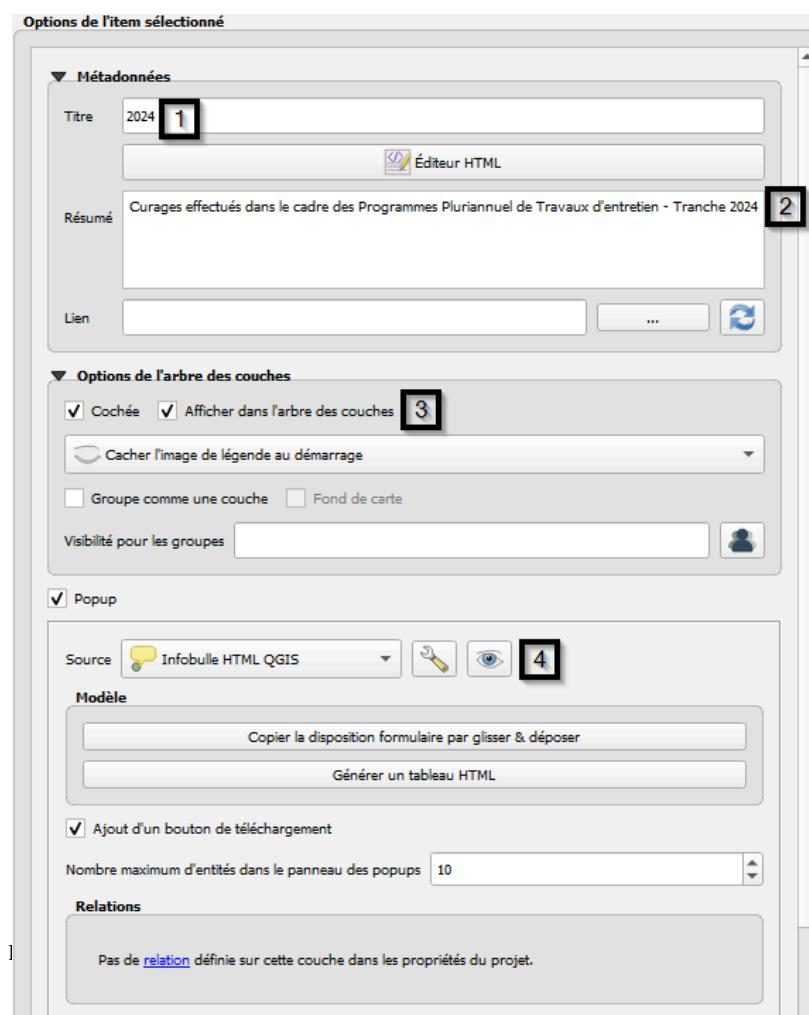
L'outils de mesure et de zoom est activé dans cette fenêtre.

Les différentes échelles possibles de la carte sont également renseignées.



Les échelles choisies correspondent à celles utilisées dans les autres productions de l'Unima.

## B. Couches



Cet onglet permet de gérer divers paramètres pour chaque couche du projet.

Le titre de la couche, tel qu'affiché dans le WebSig, est renseigné. [1]

Un contexte supplémentaire pour l'utilisateur peut être ajouté pour mieux préciser la donnée. [2]

La couche de l'année en cours est pré-cochée pour être affichée dès l'ouverture du WebSig. [3]

L'affichage des popups est activé pour chaque entité de la couche, et les infobulles HTML QGIS sont définies comme source de ces popups, comme expliqué précédemment. [4]

#### C. Fond de cartes

Les fonds de cartes choisis sont également conformes à la charte graphique de l'Unima. Le fond de plan "Voyager" est utilisé. En complément, une couche d'orthophoto de 2021, produite par le Département de la Charente-Maritime, est ajoutée.

#### D. Tables attributaires

Dans l'onglet 'Table attributaire' du plugin Lizmap, les clés primaires et les différents champs à masquer à l'utilisateur, lors de la consultation des tables attributaires des données, sont renseignés.

|    | Couche                               | Clé primaire | Champs à masquer                                  |
|----|--------------------------------------|--------------|---------------------------------------------------|
| 1  | ° ° 2021                             | 123 id       | id,id_as,date,login,adherent,num                  |
| 2  | ° ° 2022                             | 123 id       | id,id_as,login,adherent,num                       |
| 3  | ° ° 2024                             | 123 hio_id   | hio_id,hio_photo,res_code_tr                      |
| 4  | ° ° 2025                             | 123 hio_id   | hio_id,hio_photo,res_code_tr                      |
| 5  | ° ° 2026                             | 123 hio_id   | hio_id,hio_photo,res_code_tr                      |
| 6  | ° ° 2027                             | 123 hio_id   | hio_id,hio_photo,res_code_tr                      |
| 7  | V° Autres travaux entre 2014 et 2019 | 123 _uid_    | _uid_id,code_tr,num,login,ancien_codetr           |
| 8  | V° Autres travaux entre 2021 et ...  | 123 _uid_    | _uid_id,fid,code_tr,num,login,ancien_codetr       |
| 9  | V° curage_2014_20...                 | 123 _uid_    | _uid_id,code_tr,num,login,ancien_codetr           |
| 10 | V° curage_2021                       | 123 _uid_    | _uid_id,fid,id_as,code_tr,num,login,ancien_codetr |
| 11 | V° curage_2022                       | 123 _uid_    | _uid_id,fid,id_as,code_tr,num,login,ancien_codetr |
| 12 | V° curage_2023                       | 123 id       | id,fid,id_as,code_tr,num,login,ancien_codetr      |
| 13 | V° curage_2024                       | 123 _uid_    | _uid_rt_id,res_code_tr,hit_id                     |
| 14 | V° curage_2025                       | 123 _uid_    | _uid_rt_id,res_code_tr,hit_id                     |
| 15 | V° curage_2026                       | 123 _uid_    | _uid_rt_id,res_code_tr,hit_id                     |
| 16 | V° curage_2027                       | 123 _uid_    | _uid_rt_id,res_code_tr,hit_id                     |

## E. Mise en page

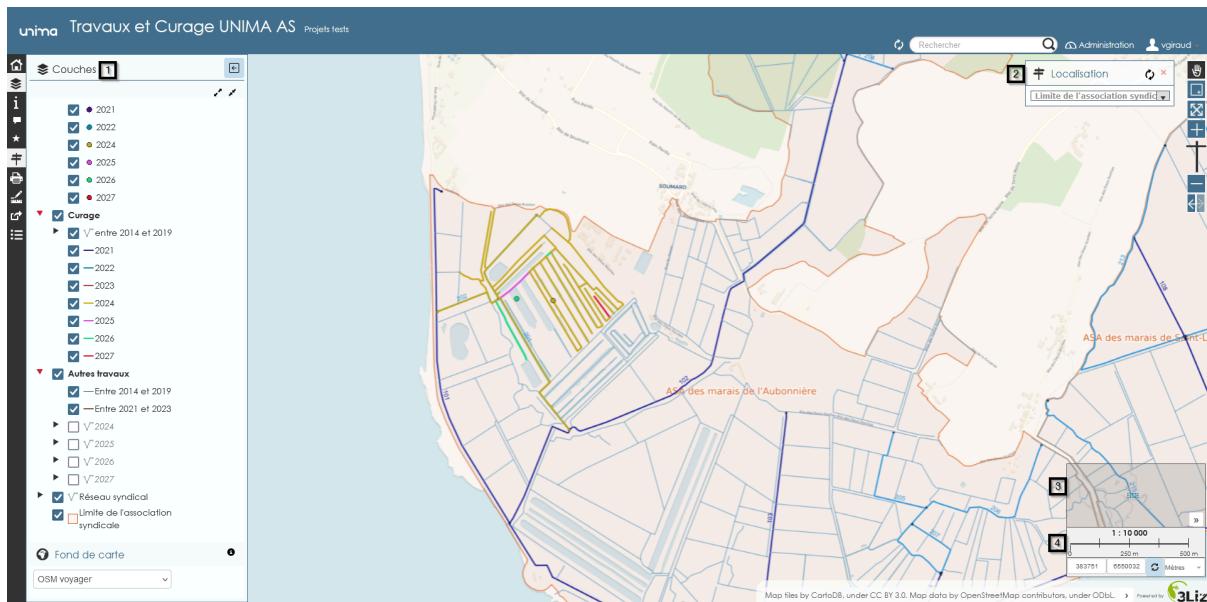
L'outils d'impression est également ajouté à la carte. Il est couplé à la mise en page QGIS du projet, et le DPI de l'impression est réglable par l'utilisateur (100, 200 ou 300 dpi)

La mise en page QGIS suit la charte graphique de l'Unima. La légende est adaptative, affichant uniquement la symbologie des entités présentes dans le canvas de la carte au moment de l'impression.

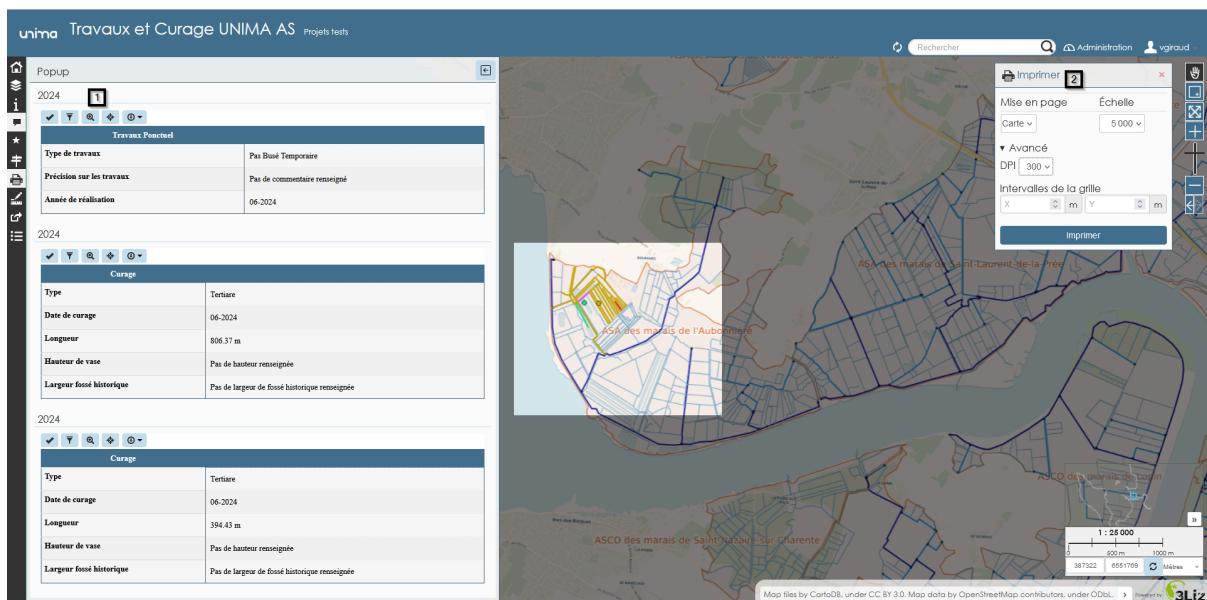
## F. Localiser par couche

Une fonction de localisation par couche a également été ajoutée. Cette dernière permet de centrer la carte sur une Association syndicale des propriétaires du marais (AS) en fonction de leur nom.

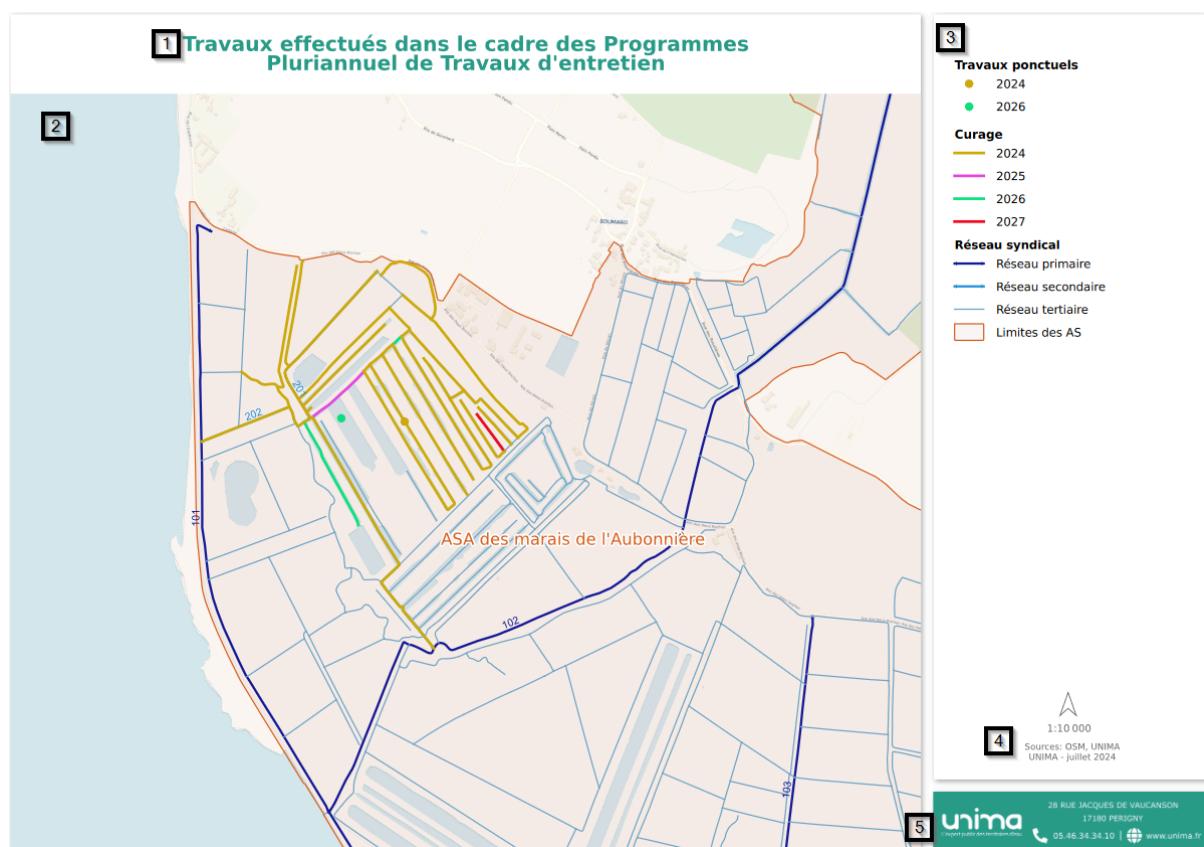
### 2.8.4 Présentation du LizMap



Dans le WebSig se trouve une légende [1], avec les couches regroupé en groupe, en fonction de l'information, un outil de localisation [2], mais aussi une carte contextuelle [3] et une échelle [4]



Ici, l'affichage des popups [1] et les options d'impression [2]



Lors de l'impression de la carte le document généré est composé d'un titre [1], du canvas de la carte (tel que sur l'illustration précédente) [2], d'une légende dynamique [3], de la flèche nord, de l'échelle et des sources [4] et enfin du logo et des coordonnées de l'Unima [5].

Les utilisateurs ont également la possibilité de télécharger directement les données et leurs géométries depuis le WebSig, ou seulement leurs tables attributaires dans plusieurs formats.

---

### 3. SUIVI ADMINISTRATIF DES ASSOCIATIONS SYNDICALE DE MARAIS

---

#### 3.1 Problématique

##### 3.1.1 Présentation du problème

L'Unima dispose d'un service administratif chargé de superviser les Associations Syndicales qui le souhaitent. Ce service assure le suivi des parcelles et des propriétaires, l'émission de facturations et de taxes, l'organisation des assemblées générales et la gestion des questionnements des propriétaires.

La base de données cadastrale et des propriétaires est mise à jour annuellement avec les nouvelles données cadastrales. Cependant, étant en contact constant avec les présidents des Associations Syndicales et les propriétaires, ce service dispose généralement d'une à deux années d'avance sur les données cadastrales officielles, ce qui peut poser des problèmes lors des mises à jour.

Ces mises à jour ne se font pas en une seule fois, mais au cas par cas, en fonction des travaux prévus, impliquant ainsi de nombreuses mises à jour tout au long de l'année.

##### 3.1.2 Etat des lieux

L'UNIMA paye actuellement un abonnement à une solution cloud de l'entreprise JVS-Mairistem pour la gestion administrative des Associations Syndicales qui ont confié leur secrétariat à l'UNIMA. Cette solution se présente sous la forme d'une interface permettant la consultation et la saisie d'informations cadastrales concernant les propriétaires de parcelles situées dans le périmètre des Associations Syndicales. Cette solution propriétaire à un coût significatif pour l'UNIMA et rend difficile la mise à jour des données, qui est facturée par le prestataire.

### 3.1.3 Étapes de fonctionnement de la solution actuelle

1. Mise à jour d'un Plan d'ensemble : La cellule SIG met à jour le plan d'ensemble<sup>22</sup> des parcelles.
2. Création d'un formulaire Excel : Les informations des propriétaires extraites du plan d'ensemble sont compilées dans un formulaire Excel.
3. Envoi à JVS : Le formulaire est envoyé à JVS pour traitement.
4. Retour des conflits par JVS : JVS renvoie les éventuels conflits, indiquant les discordances entre le cadastre et la base de données.
5. Résolution des conflits : Le service administratif résout les conflits.
6. Mise à jour de la base de données : Une fois les conflits résolus, les informations sont de nouveaux transmises à JVS puis la base de données est mise à jour
7. Intégration des tarifications : Les tarifications sont intégrées dans la base de données mise à jour.
8. Publipostage : Les informations mises à jour sont utilisées pour générer et envoyer des documents aux propriétaires.

## 3.2 Creations d'un cahier des charges

### 3.2.1 Critères de la solutions

Pour mieux comprendre les besoins réels du service administratif, plusieurs réunions ont été organisées. Il en est ressorti des demandes formelles, des axes d'amélioration du service existant ("ASAPérimètre" de JVS-Mairistem), et de nouvelles fonctionnalités souhaitées. Ces demandes concernent les fonctionnalités pratiques du service, plutôt que des aspects techniques.

Axes d'amélioration et nouvelles fonctionnalités souhaitées :

- ⇒ Meilleure gestion des conflits lors de l'import de données
- ⇒ Meilleure ergonomie
- ⇒ Fluidité de la solution développée
- ⇒ Meilleure gestion des propriétaires et des parcelles

Fonctionnalités existantes à conserver :

- ⇒ Calcul automatique des tarifications
- ⇒ Publipostage des factures
- ⇒ Publipostage des convocations aux assemblées générales

---

<sup>22</sup> Mise à jour des limites et parcelles composant un Association Syndicale donnée, en vue d'une gestion administratives et financière

## ⇒ Émission de documents d'émargement lors des assemblées générales

Il est vite apparu que la compréhension du périmètre technique est primordiale avant de continuer la réflexion, notamment sur les besoins réels des tables et des champs à implémenter dans la future base de données.

Ces notions techniques n'étant pas maîtrisées ou conscientisées par le service administratif, leurs demandes concernant les informations nécessaires au bon fonctionnement du support n'ont pas été émises.

Plusieurs jours ont été consacrés à comprendre les différents champs et attributs proposés par la solution de JVS-Mairistem. L'objectif était d'évaluer leur pertinence et de déterminer s'il était nécessaire de les conserver ou de les modifier pour répondre aux besoins spécifiques du service administratif.

Le résultat des recherches effectué sur la Solution de JVS-Mairistem est consultable [ici](#)

### 3.2.1 Solutions envisagées

Plusieurs solutions ont été envisagées pour répondre aux besoins du service administratif. La première proposition consistait à développer un logiciel stand-alone utilisant QGIS et Python. Cette solution permettrait de créer un logiciel desktop capable de traiter efficacement toutes les tables et leurs informations grâce à la puissance de Python couplée aux modules natifs de QGIS. L'intégration directe avec QGIS apparaissait comme un avantage notable pour le traitement de la base de données. Cependant, cette approche a été rapidement écartée suite à la demande de mon maître de stage, qui a suggéré de reconsidérer cette solution en raison de sa complexité et du besoin de recul sur la solution globale.

Après une réflexion approfondie, il est devenu évident que la solution devait prendre la forme d'un site web connecté à une base de données. Cette approche offrirait plusieurs avantages, notamment l'accessibilité depuis n'importe quel appareil, la centralisation des données et une mise à jour simplifiée. En optant pour une solution web, l'UNIMA pourrait gérer de manière plus efficace et cohérente les informations administratives et cadastrales tout en facilitant l'accès et la manipulation des données pour les utilisateurs.

Pour créer un site web connecté à une base de données dynamique, l'utilisation de PHP pourrait sembler une solution évidente. Cependant, en raison de la complexité du projet, qui implique un grand nombre de fonctionnalités et de pages, ainsi que d'une maîtrise limitée de PHP, il a été nécessaire de rechercher d'autres solutions.

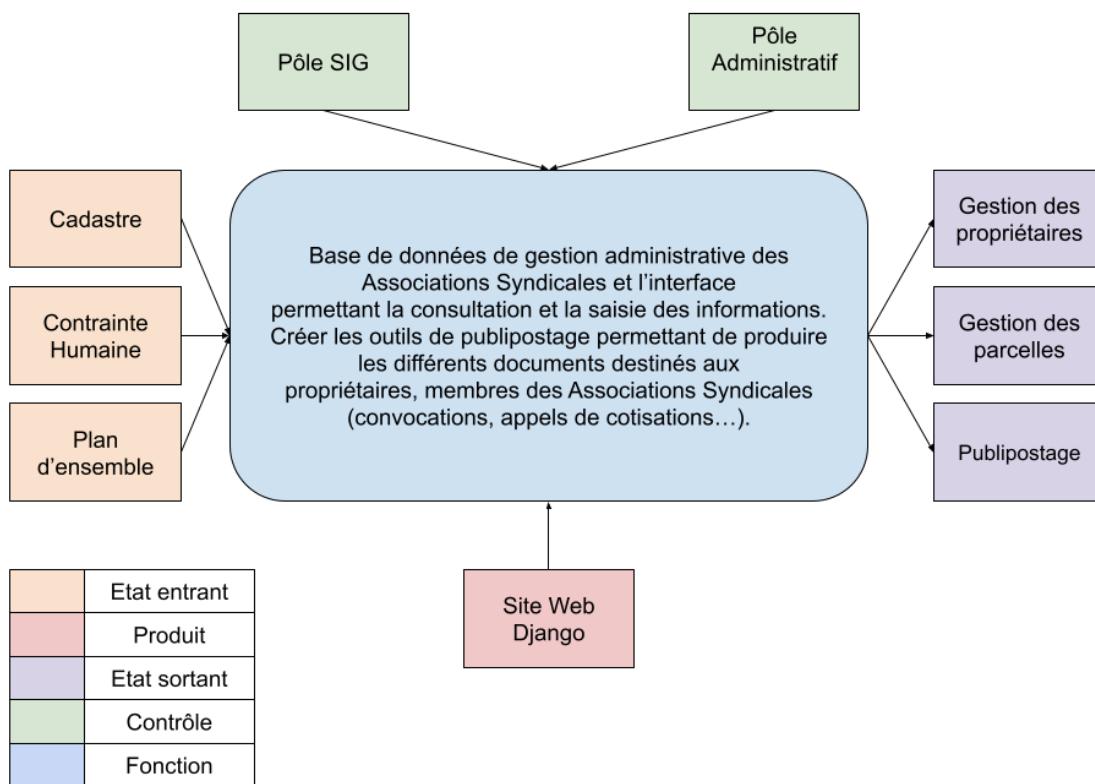
Django a été évoqué à plusieurs reprises comme une solution potentielle. Ce framework Python est idéal pour répondre aux besoins identifiés. Il offre de nombreux avantages, notamment la simplification de la création de sites web, la gestion de bases de données, et l'organisation de projets. De plus, Django bénéficie d'une communauté de développeurs très active et impliquée, ce qui facilite le support et l'amélioration continue de la solution.

Après discussion avec mon maître de stage, nous avons convenu que l'utilisation de Django, couplée à une base de données, serait la solution optimale pour réaliser ce projet. Django permettrait de centraliser les données administratives et cadastrales de manière efficace, tout en offrant une interface web accessible et facile à utiliser pour les utilisateurs du service administratif.

### 3.3 La solution développée

#### 3.3.1 Elaboration du processus de réflexion

Le processus de réflexion pour mettre en place la solution a été une succession d'étapes avant d'arriver à un schéma cohérent de penser. Dans un premier temps un SADT de niveau 0 a permis de remettre en contexte ce que devait apporter la solution :



SADT de niveau 0 pour la solution sur l'administration d'une base de données cadastrale

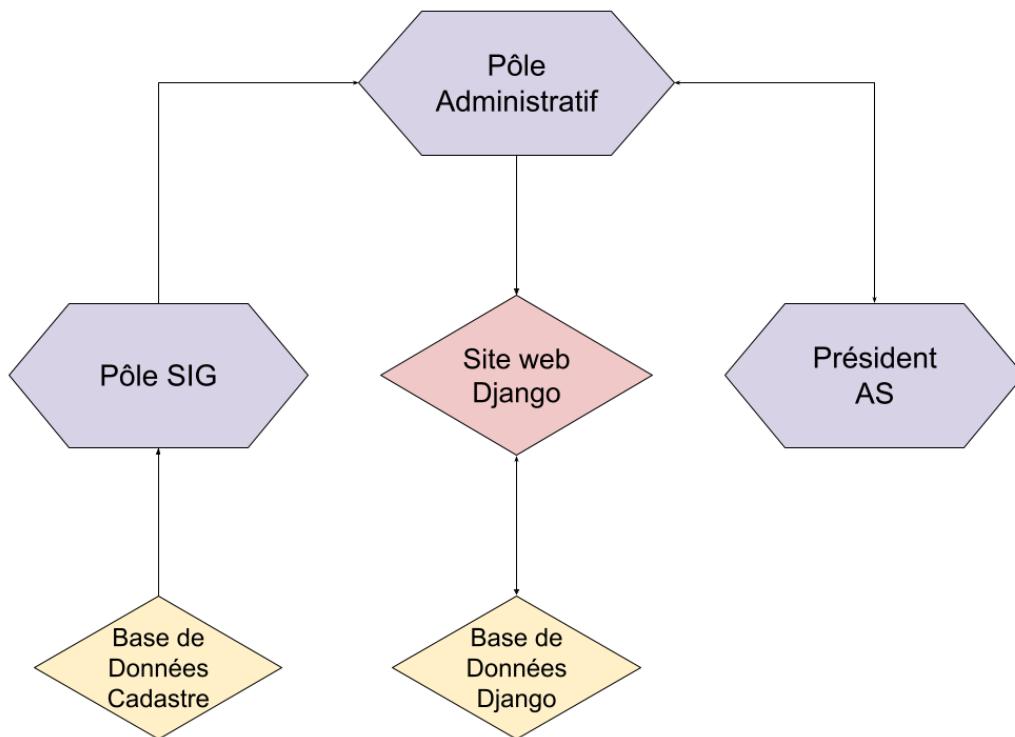
### 3.3.2 Schématisation du fonctionnement de la solution

La solution fera intervenir 3 acteurs, 2 bases de données et un site de gestion de base de données

Le fonctionnement de la solution restera sensiblement identique :

1. Mise à jour d'un Plan d'ensemble : La cellule SIG met à jour le plan d'ensemble des parcelles.
2. Crédit d'un formulaire Excel : Les informations des propriétaires extraites du plan d'ensemble sont compilées dans un formulaire Excel.
3. Envoi du formulaire dans le site web : Le formulaire est envoyé au pôle administratif pour vérification puis importation dans le site web.

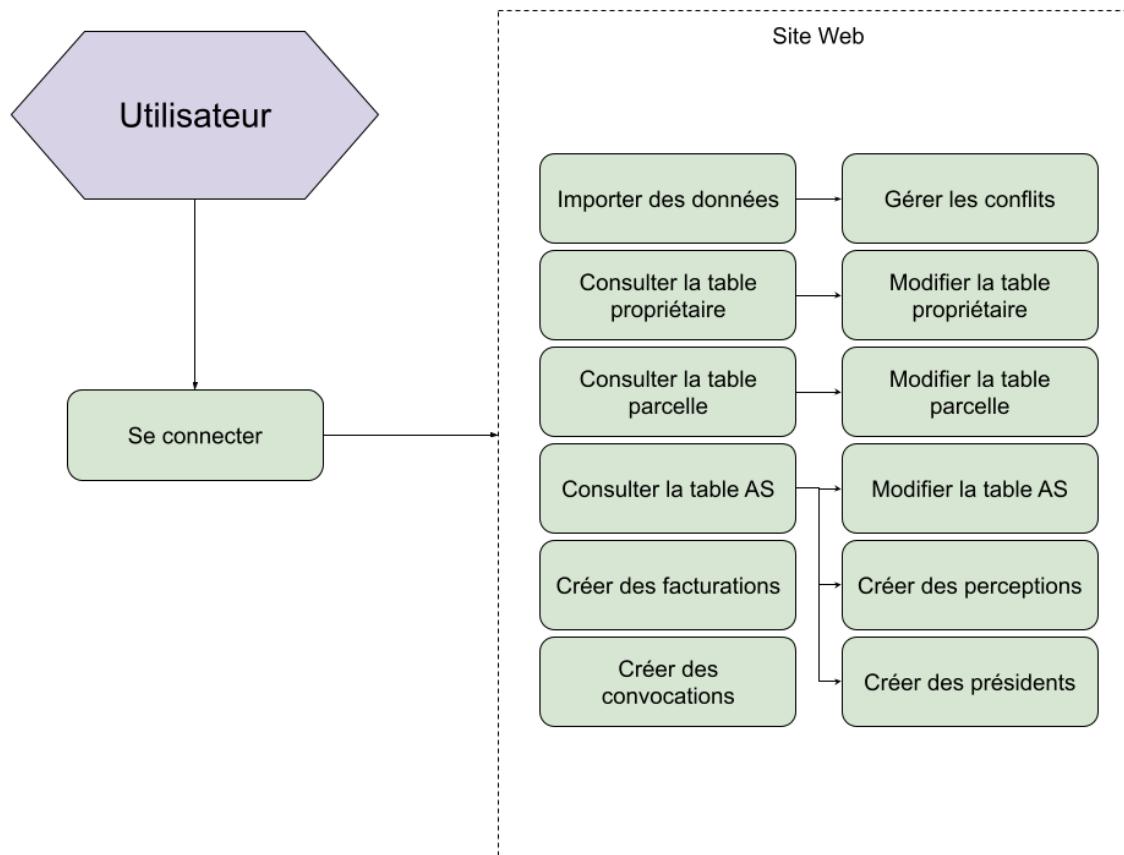
4. Retour des conflits par le site web : le site web renvoie les éventuels conflits, indiquant les discordances entre le cadastre et la base de données.
5. Résolution des conflits : Le service administratif résout les conflits avec les informations transmises par le site web.
6. Mise à jour de la base de données : Une fois les conflits résolus, la base de données est mise à jour
7. Intégration des tarifications : Les tarifications sont intégrées dans la base de données mise à jour.
8. Publipostage : Les informations mises à jour sont utilisées pour générer et envoyer des documents aux propriétaires.



Schématisation de l'interaction des acteurs pour le publipostage de la facturation et des convocations

Dans le cadre du stage, nous nous concentrons uniquement sur le développement du site web. Après délibérations avec mon maître de stage, nous avons identifié 13 fonctionnalités essentielles, y compris la gestion de la connexion de l'utilisateur à

l'application. Ces fonctionnalités visent à améliorer l'ergonomie, la gestion des conflits de données, et à fournir des outils automatisés pour les tâches administratives courantes.



Schématisation de l'utilisation du site web par le pôle administration

### 3.3.3 Elaboration d'un planning de travail

Enfin, un diagramme de Gantt a été produit pour une meilleure organisation du temps de travail

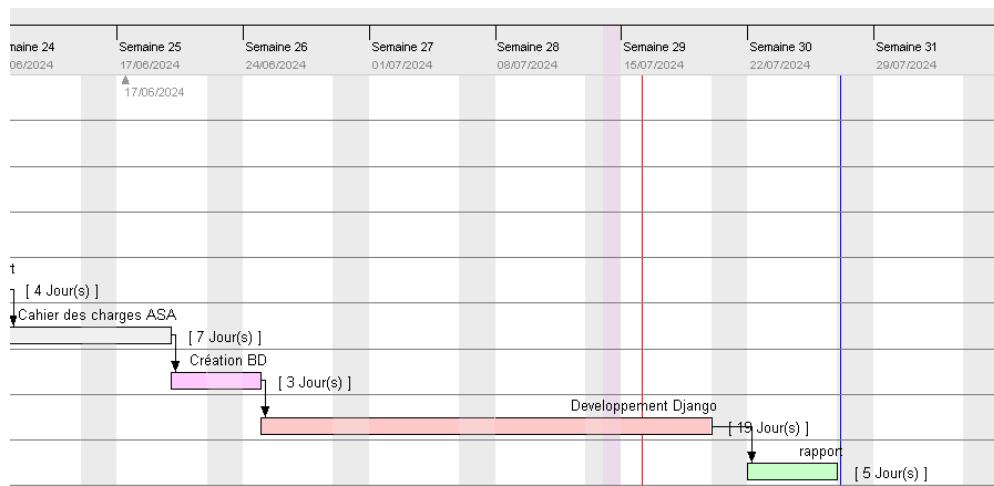


Diagramme de GANTT prévisionnel du projet

Nous remarquons que le travail est divisé en trois grande étapes:

1. La réalisation du cahier des charges vu précédemment
2. La création d'une base de données pour stockées les informations
3. La partie développement du site web en python, HTML, CSS, Javascript

Enfin du temps est alloué à la rédaction du rapport en fin de projet pour une meilleure gestion du temps de ce dernier.

### 3.4 Le Framework Django

Django est un framework<sup>23</sup> web open source en Python. Il a pour but de rendre le développement d'applications web simple et basé sur la réutilisation de code. Développé en 2003 pour le journal local de Lawrence (État du Kansas, aux États-Unis), Django a été publié sous licence BSD à partir de juillet 2005.

Depuis juin 2008, la Django Software Foundation s'occupe du développement et de la promotion du framework. En plus de cette promotion régulière, des conférences entre développeurs et utilisateurs de Django sont organisées deux fois par an depuis 2008. Nommées DjangoCon, une se déroule en Europe et l'autre aux États-Unis.

<sup>23</sup> Un framework est constitué de plusieurs bibliothèques, chacune spécialisée dans un domaine.

Django est un framework qui s'inspire du principe MVC<sup>24</sup> composé de trois parties distinctes :

- ⇒ Un langage flexible qui permet de générer du HTML, XML ou tout autre format texte ;
- ⇒ Un contrôleur fourni sous la forme d'un « remapping » d'URL à base d'expressions rationnelles ;
- ⇒ Une API<sup>25</sup> d'accès aux données est automatiquement générée par le framework compatible CRUD<sup>26</sup>. Inutile d'écrire des requêtes SQL associées à des formulaires, elles sont générées automatiquement par l'ORM.<sup>27</sup>

En plus de l'API d'accès aux données, une interface d'administration fonctionnelle est générée depuis le modèle de données. Un système de validation des données entrées par l'utilisateur est également disponible et permet d'afficher des messages d'erreur automatiques.

Sont également inclus :

- ⇒ un serveur web léger permettant de développer et tester ses applications en temps réel sans déploiement ;
- ⇒ un système élaboré de traitement des formulaires muni de widgets permettant d'interagir entre du HTML et une base de données. De nombreuses possibilités de contrôles et de traitements sont fournies ;
- ⇒ un framework de cache web pouvant utiliser différentes méthodes (MemCached, système de fichier, base de données, personnalisé) ;
- ⇒ le support de classes intermédiaires (intergiciel) qui peuvent être placées à des stades variés du traitement des requêtes pour intégrer des traitements particuliers (cache, internationalisation, accès...) ;
- ⇒ une prise en charge complète d'Unicode.

Django peut être considéré comme une boîte à outils où chaque module peut fonctionner de façon indépendante.

### 3.4.1 Le principe de base

Une application Django fonctionne de la manière suivante :

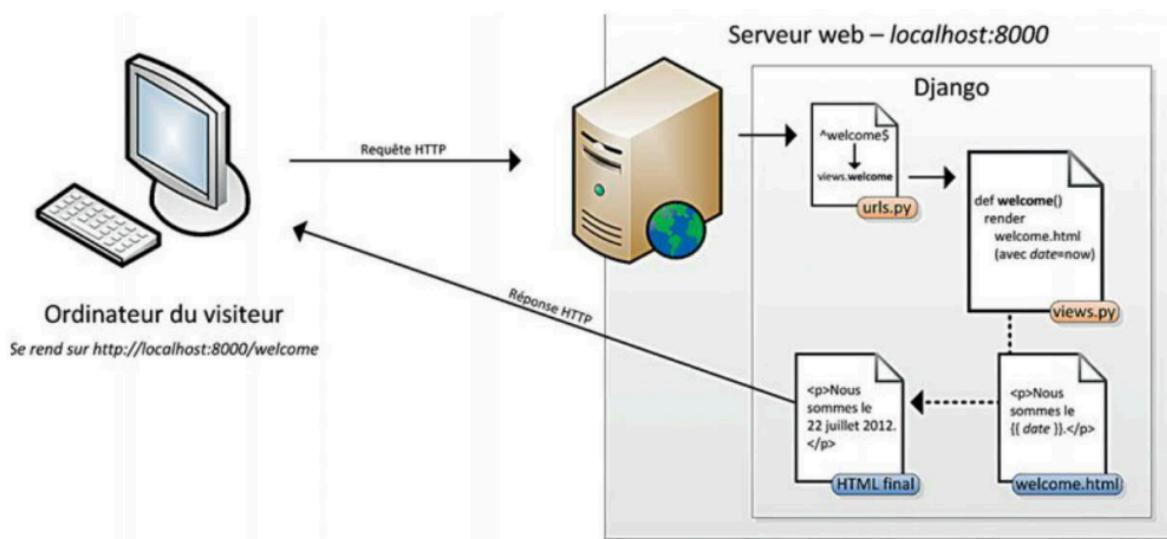
---

<sup>24</sup> Modèle-vue-contrôleur ou MVC est un motif d'architecture logicielle destiné aux interfaces graphiques, lancé en 1978 et très populaire pour les applications web.

<sup>25</sup> Une API est un ensemble normalisé de classes, de méthodes, de fonctions et de constantes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

<sup>26</sup> **Create, Read, Update, Delete**

<sup>27</sup> L'ORM est un type de programme informatique qui se place en interface entre un programme applicatif et une base de données relationnelle pour simuler une base de données orientée objet.



Il est important de comprendre qu'une application Django ne peut fonctionner qu'avec divers fichiers python communiquant entre eux. Ainsi pour afficher une page Web ayant n'importe quel contenu il faut renseigner diverses informations dans des fichiers et des classes précises. Même si le principe paraît fastidieux, avec de l'entraînement cela permet d'économiser un temps important.

### 3.4.2 Les fichiers primordiaux

Voici une liste des fichiers python primordiaux utilisés dans le cadre de l'application django réalisée durant le stage, mais se retrouvant dans toutes les applications Django.

- ❖ `settings.py` : Le fichier `settings.py` contient toutes les configurations de votre projet Django. Voici quelques-unes de ses principales fonctions
  - Configuration de la base de données : Définition des paramètres de connexion à la base de données.
  - Applications installées : Liste des applications Django et des applications tierces utilisées dans le projet.
  - Middleware : Définition des composants middleware qui traitent les requêtes et réponses.
  - Configurations de sécurité : Paramètres comme `SECRET_KEY`, `DEBUG`, et les configurations pour les hôtes autorisés (`ALLOWED_HOSTS`).
  - Paramètres internationaux : Configuration de la langue (`LANGUAGE_CODE`) et du fuseau horaire (`TIME_ZONE`).
  - Gestion des fichiers statiques et médias : Chemins et URL pour les fichiers statiques et les fichiers médias.

- ❖ urls.py : Le fichier `urls.py` est responsable de la gestion du routage des URL vers les vues correspondantes. Il définit les correspondances entre les URL et les vues de l'application.
- ❖ views.py : Le fichier `views.py` dans un projet Django joue un rôle central en définissant la logique métier de l'application. Les vues gèrent les requêtes HTTP et renvoient les réponses appropriées. Elles peuvent être basées sur des fonctions ou des classes, chacune ayant ses propres avantages.
  - Vues basées sur les fonctions (FBV) : Vues définies sous forme de fonctions. [C'est sur ce type de vue que l'application a été développé](#)
  - Vues basées sur les classes (CBV) : Vues définies sous forme de classes, permettant la réutilisation et la modularité du code.
- ❖ models.py : Le fichier `models.py` contient les définitions des modèles de données. Chaque modèle correspond à une table de base de données et définit les champs et comportements de cette table.
- ❖ forms.py : Le fichier `forms.py` est utilisé pour définir les formulaires de l'application. Django fournit des outils pour générer et gérer les formulaires, y compris la validation et le rendu HTML.
  - Formulaires basés sur les modèles : Formulaires générés automatiquement à partir des modèles.
  - Formulaires personnalisés : Formulaires définis manuellement avec des champs spécifiques.
- ❖ admin.py : Le fichier `admin.py` est utilisé pour enregistrer les modèles afin qu'ils soient gérés via l'interface d'administration de Django. Il permet également de personnaliser cette interface.

### 3.4.3 Les fichiers d'application externe

De plus, comme mentionné dans la description de `settings.py`, des applications externes peuvent être utilisées pour personnaliser davantage notre application Django. Dans le contexte de cette application nous utilisons la bibliothèque 'Django-Tables2'. Elle facilite la création de tables HTML à partir de modèles Django, avec des fonctionnalités avancées comme le tri, la pagination et la personnalisation des colonnes. Deux fichiers couramment utilisés avec cette bibliothèque sont `tables.py` et `filters.py`.

- ❖ tables.py : Le fichier `tables.py` est utilisé pour définir les tables de données en utilisant les classes fournies par `django-tables2`. Voici comment ce fichier est structuré et utilisé :
  - Définir des Tables : Il permet de créer des classes qui représentent des tables de données.

- Personnalisation des Colonnes : On peut personnaliser les colonnes, ajouter des attributs, formater les données, etc.
  - Intégration avec les Modèles : Les tables peuvent être directement liées aux modèles Django, facilitant ainsi l'affichage des données.
- ❖ filter.py : Le fichier `filters.py` est utilisé pour définir des filtres de données en ajoutant des capacités de filtrage aux tables.
- Définir des Filtres : Il permet de créer des classes de filtres qui peuvent être appliquées aux requêtes de modèles.
  - Intégration avec les Vues : Facilite l'ajout de formulaires de filtrage dans les vues pour affiner les résultats affichés dans les tables.
  - Personnalisation des Champs de Filtre : On peut personnaliser les champs de filtre, ajouter des validations et des widgets.

### 3.4.3 La page d'administration

Django offre une page d'administration nativement et automatiquement. Cette fonctionnalité intégrée permet aux administrateurs de gérer facilement les données et les utilisateurs de l'application web.

Un administrateur, ou superutilisateur, peut être créé via la ligne de commande en utilisant la commande suivante :

```
python manage.py createsuperuser
```

Une fois le superutilisateur créé, il peut se connecter à la page d'administration en se rendant à l'adresse web suivante :

`http://MonSiteWeb/admin/`

Après identification, le superutilisateur accède à une interface listant toutes les tables et les utilisateurs de l'application web. Cette page permet de :

- Ajouter ou supprimer des enregistrements de tables.
- Gérer les droits et les informations des utilisateurs.

La page d'administration se crée et se met à jour automatiquement, offrant un contrôle total sur l'application web. Cette automatisation accélère considérablement le processus de création et de déploiement des applications web, rendant Django particulièrement efficace pour les développeurs.

## 3.5 Les bases de données

### 3.4.1 La base de données

La base de données devra s'adapter aux besoins spécifiques de l'utilisateur, du format de la donnée en entrée (fichier excel) mais aussi répondre aux spécificités du framework Django. En effet, Django est conçu pour ne pas manipuler des requêtes ou des tables SQL mais des classes et des objets python. Même si le framework est très puissant, ne maîtrisant pas tous ses aspects, ce manque de connaissance se ressent sur la modélisation de la base de données. La base de données à donc été déployé de cette manière : [Lien vers diagramme](#)

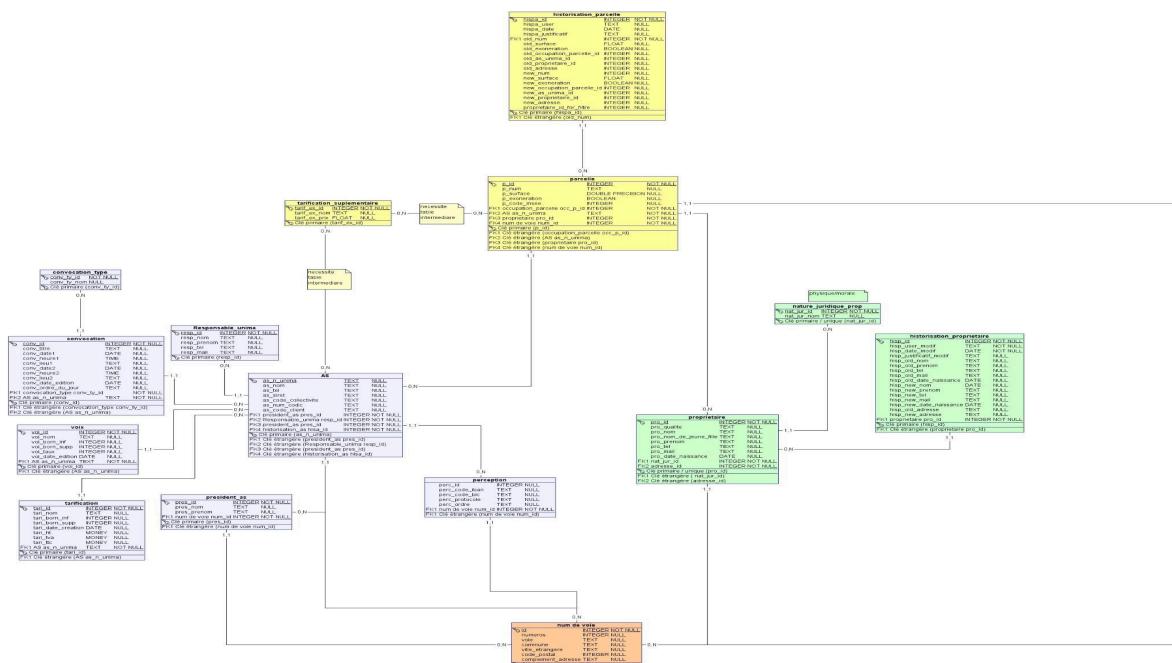


Diagramme relationnel de la base de données test

Elle est découpée en 4 parties.

- La partie adresse (en orange)
- ↳ Numéros + indice de répétition

- ⇒ Nom de voie
- ⇒ Nom de la commune
- ⇒ Code postal de la commune
- ⇒ Complément d'adresse
- ⇒ ville étrangère (permet de renseigner une adresse à l'étranger)
- L'adresse se décompose conformément à une adresse de type BAN
  - Liée aux formulaires "adresse" lors de l'ajout ou la modification d'une adresse. (**VOIR FORMULAIRE**)
- La partie propriétaire (en vert)
  - ⇒ Qualité
  - ⇒ Nom
  - ⇒ Nom de jeune fille
  - ⇒ Prénom
  - ⇒ Téléphone
  - ⇒ Email
  - ⇒ Date de naissance
  - ⇒ Sa nature juridique
  - ⇒ Son adresse
  - ⇒ L'historisation des propriétaires
- La partie parcelle (en jaune)
  - ⇒ Numéros de parcelle
  - ⇒ Surface
  - ⇒ Sa possible exonération
  - ⇒ Code Insee de la parcelle
  - ⇒ Tarification supplémentaire possible
  - ⇒ L'Association Syndicale d'appartenance
  - ⇒ L'historisation des parcelles
  - ⇒ Son propriétaire
  - Il a été exprimé le fait que l'adresse et la géométrie des parcelles n'était pas important pour le pôle administratif, ils n'ont donc pas étaient implémentés
- La partie association syndicale (en bleue)
  - ⇒ Nom
  - ⇒ Téléphone
  - ⇒ Numéro de SIRET
  - ⇒ Code collectivité
  - ⇒ Numéro CODIC
  - ⇒ Code Client
  - ⇒ Le taux de facturation
    - ⇒ Nom
    - ⇒ Borne inférieur

- ⇒ Borne supérieur
- ⇒ Date de création
- ⇒ Taux
- ⇒ Les convocations aux assemblées générales
  - ⇒ Titre
  - ⇒ Date 1 et 2
  - ⇒ Heure 1 et 2
  - ⇒ Lieu 1 et 2
  - ⇒ Date d'édition
  - ⇒ Ordre du jour
- ⇒ Les informations du président
  - ⇒ Nom
  - ⇒ Prénom
  - ⇒ Téléphone
  - ⇒ Email
  - ⇒ Adresse
- ⇒ Responsable Unima
  - ⇒ Nom
  - ⇒ Prénom
  - ⇒ Téléphone
  - ⇒ Email
- ⇒ Chaque AS se voit attribuer un responsable Unima du pôle administratif

### 3.4.2 Les models Django

Cette base à ensuite dû être traduire en models django. Les modèles sont la représentation centrale des données dans une application Django. Chaque modèle correspond à une table de base de données et est défini comme une classe Python. Les attributs de la classe représentent les champs de la table. Voici un exemple simple d'un modèle :

```
from django.db import models

class Article(models.Model):
 title = models.CharField(max_length=200)
 content = models.TextField()
 published_date = models.DateTimeField(auto_now=True)
```

Django fournit une variété de champs pour différents types de données, tels que **CharField** pour les chaînes de caractères, **IntegerField** pour les nombres entiers, **DateTimeField** pour les dates et heures, etc. Chaque champ peut avoir des options de validation et des contraintes.

Vous trouverez le fichier models.py, résultant en grande partie de la traduction du modèle de données, [ici](#).

### 3.4.3 La requête SQL

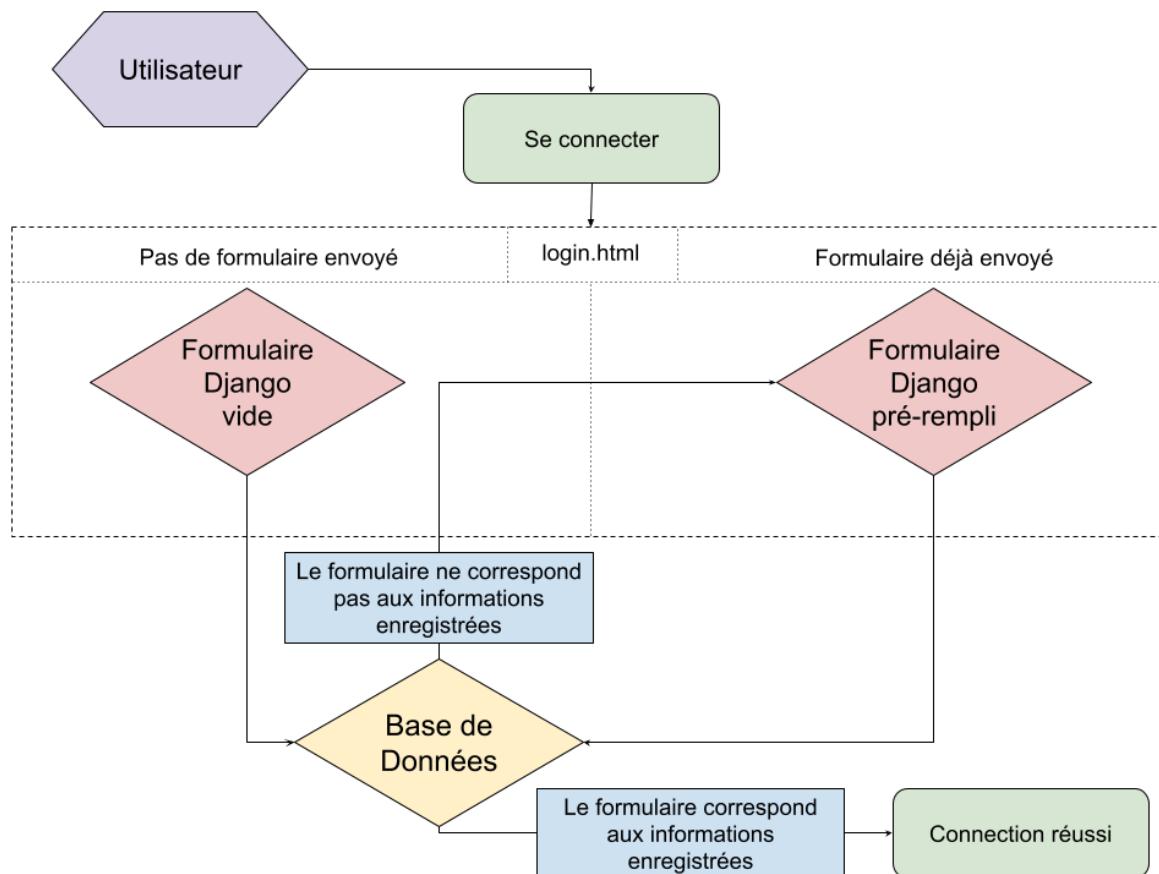
## 3.5 L'application Django

### 3.5.1 L'identification

La première étape à réaliser est la création d'un système d'authentification.

Cette première page permettra de mieux appréhender les connexions entre les différents fichiers et méthodes Python afin de réaliser un page web dynamique et ses actions.

#### A. Logique



Logique de la page de connexion

Pour se connecter l'utilisateur doit soumettre un formulaire avec son nom d'utilisateur et son mot de passe dans la page web dédiée<sup>28</sup>. Une fois le formulaire validé, les

<sup>28</sup> login.html

données sont vérifiées avec les données stockées en base de données. Si les données sont validées, la connexion est réussie et l'utilisateur est redirigé vers la page d'accueil de l'application. Sinon l'utilisateur est redirigé vers la même web, avec l'affichage d'un message d'erreur, et le nom d'utilisateur renseigné précédemment déjà dans le formulaire.

#### B. L'interface utilisateur

##### C. La page Web

###### a. Fonctionnement de Django

Les pages web des projets django peuvent être écrites comme un page web classique, c'est-à-dire avec du HTML pour les éléments composants la page web, du CSS pour le style de ces éléments, et éventuellement du JavaScript pour l'interaction avec l'utilisateur.

Cependant plusieurs fonctionnalités majeures supplémentaires sont présentes dans la construction des pages web Django.

#### Logique Algorithmique

Django permet l'intégration facile de logiques algorithmiques dans les pages web via des balises spécifiques. Cela inclut des opérations comme `if`, `for...in`, et `while`, qui peuvent être directement utilisées dans la syntaxe de la page web pour créer des comportements dynamiques et conditionnels.

#### Contexte

Les pages web Django disposent d'un contexte supplémentaire qui permet la réalisation de ces logiques algorithmiques. Ce contexte est transmis depuis la vue, ce qui permet d'intégrer des variables et des données dynamiques directement dans les templates<sup>29</sup>.

#### Template Extending

Django est conçu pour maximiser la réutilisation du code. Dans cette optique, des fichiers HTML peuvent être écrits et servir de base pour d'autres pages. Cela permet de réutiliser des éléments HTML courants comme le `<head>`, le `<header>` et le `<footer>`. En utilisant la balise `{% extends %}`, les développeurs peuvent définir une structure de base qui est partagée par plusieurs pages.

#### Blocks

---

<sup>29</sup> Les templates sont des modèles de page HTML utilisés pour afficher des données dynamiques en Django.

Cette fonctionnalité découle directement du principe d'extending. Avec des balises dédiées, il est possible de créer des "blocks". Ces blocks permettent d'ajouter ou de modifier des éléments du code. Grâce à ce principe, des classes ou des ids peuvent être changés en fonction de la page web. De plus, ces blocks peuvent avoir des valeurs par défaut pour plus de personnalisation.

Le block le plus fréquemment utilisé est le block **content**. Ce block est généralement employé pour intégrer le contenu spécifique d'une page web, offrant ainsi une flexibilité et une modularité accrues dans la conception des pages.

#### b. Dans l'application

Dans l'application, une page HTML est utilisée comme socle à toutes les autres pages de l'application. Cette page est composée d'un **<head>** composé des références aux fichiers de styles CSS, de script JavaScript commun à toutes les pages de l'application. Dans ce fichier se trouve aussi un **<header>**, lui aussi commun à toutes les pages, prenant la forme d'une bannière permettant de naviguer dans l'application.

Le code de cette page HTML se trouve [ici](#)

Cette méthode permet d'avoir des fichiers de codes HTML beaucoup plus simples, en témoigne la page HTML de la connexion composée de très peu de lignes de codes.

Liens vers la page 'login.html'

#### D. *Forms.py*

Il est aussi à noter que la longueur du code de cette page de login est aussi influencée par l'utilisation d'une nouvelle balise Django '`{{ form.as_p }}`'. Cette balise permet de générer automatiquement le code HTML nécessaire pour les formulaires, réduisant ainsi la longueur du code et simplifiant la gestion des formulaires. En utilisant `{{ form.as_p }}`, les développeurs peuvent créer des formulaires sans écrire manuellement le code HTML, rendant le processus de création et de gestion des formulaires plus efficace et moins sujet aux erreurs.

Pour se faire il faut préalablement créer des formulaire Django. Ces formulaires prennent la forme de classes héritant de `forms`, une classe native de Django.

Par exemple, le formulaire de login `LoginForm` est composé de deux champs : un champ utilisateur et un champ mot de passe. Une fois créé, ce formulaire vérifie si les informations entrées correspondent à un utilisateur existant dans la base de données et renvoie l'information appropriée.

Le formulaire de login est consultable [ici](#)

## E. Views.py

Fichier pivot de toutes applications Django, la définition d'une vue se fait avec comme paramètre obligatoire '**request**' qui correspond à la requête HTML reçue par la page web.

Cette vue va d'abord vérifier si le formulaire à déjà été envoyé (présence de données de la méthode HTTP POST<sup>30</sup>). Si non, la page est affichée avec un formulaire vide. Si oui, la page récupère les informations contenues dans le POST et vérifie si une correspondance existe dans la base de données. Si aucune correspondance n'est trouvée, l'utilisateur reste sur la même page, avec l'affichage d'un message d'erreur (géré automatiquement par Django). A l'inverse, si une correspondance est trouvée, l'utilisateur voit son identifiant sauvegardé dans la session Django, puis redirigé vers la page d'accueil de l'application.

### Note

Les sessions Django sont le nom des cookies utilisés par l'application.  
 Comme pour les cookies, elles permettent de stocker les informations de l'utilisateur sur le disque dur de l'utilisateur, afin de réutiliser les données ultérieurement.  
 Dans ce contexte, seul l'id de l'utilisateur connecté et sauvegardé. En effet Django sauvegarde les informations de l'utilisateur dans une base de données dédiée, créer automatiquement.

La méthode liée au login est consultable [ici](#)

## F.Urls.py

Finalement il faut indiquer à Django quelle méthode utilisé lorsque l'utilisateur se trouve sur la page web du login. Pour ce faire, une ligne de code dans le fichier urls.py suffit. Dans le cadre de l'application web développé elle se compose de cette manière :

```
path('NomDeLaPageDansUrl',views.NomDeLaView, name='NomAppel')
```

avec :

- ⇒ **path** une méthode native de Django
- ⇒ **NomDeLaView**, importer du fichier views.py
- ⇒ **'NomAppel'**, permettant d'utiliser ce nom au lieu de **'NomDeLaPageDansUrl'** dans le code python et HTML

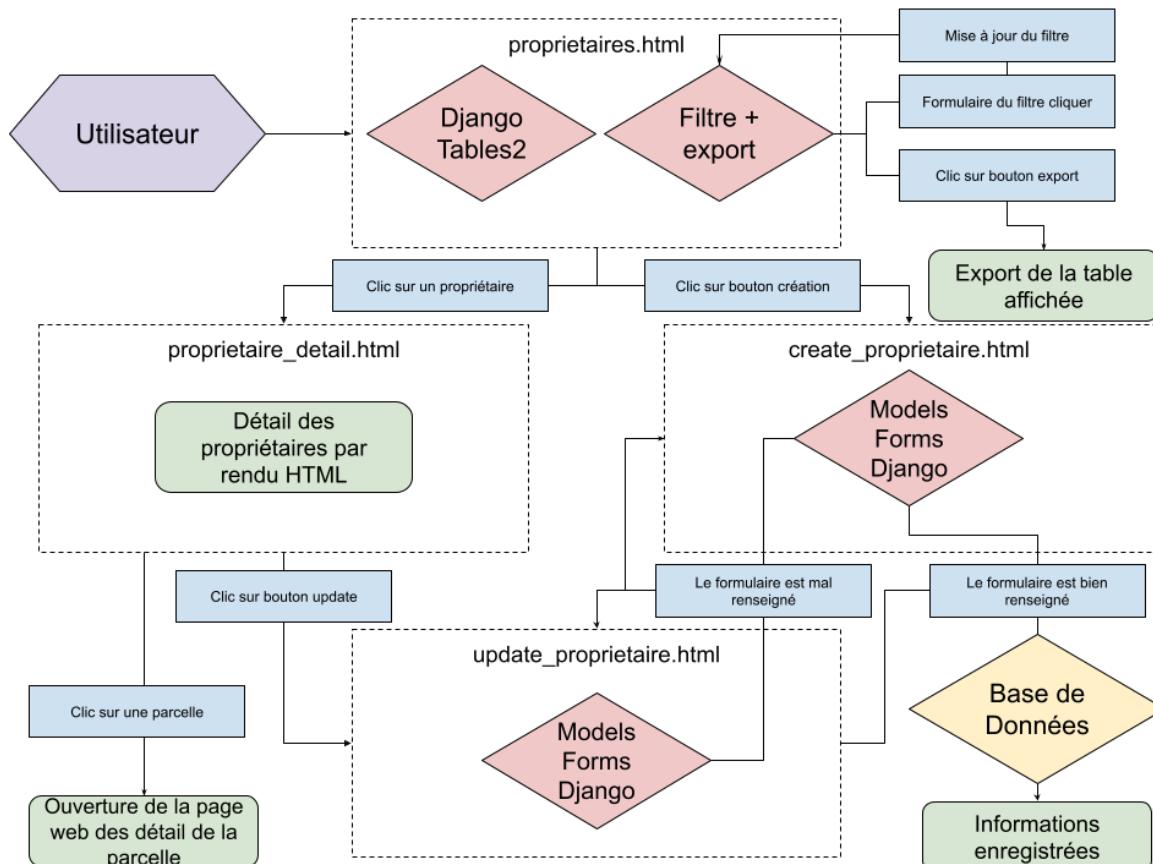
---

<sup>30</sup> La méthode POST, utilisée pour les mises à jour, envoie les données en toute sécurité dans le corps de la demande, ce qui est parfait pour les formulaires.



### 3.5.2 La gestion des propriétaires

#### A. Logique



Logique de la gestion des propriétaires

La gestion et l'administration des propriétaires constituent l'une des fonctionnalités principales de l'application. Les utilisateurs doivent pouvoir rechercher facilement un propriétaire dans la base de données, consulter et modifier ses informations.

Pour répondre à ces besoins, trois pages HTML ont été créées. La première page permet de visualiser l'ensemble des propriétaires dans un tableau. Ce tableau intègre une fonctionnalité de filtrage pour faciliter la recherche d'un propriétaire spécifique et d'une possibilité d'export au format CSV et XLSX. Des informations sommaires sur chaque propriétaire sont affichées ici.

En cliquant sur une ligne du tableau, l'utilisateur accède à une nouvelle page web qui récapitule toutes les informations détaillées du propriétaire sélectionné. Cette page permet également de modifier les informations du propriétaire.

Une troisième page permet la création d'un nouveau propriétaire. Cela permet de gérer facilement l'ajout de nouveaux propriétaires dans la base de données.

Il est important de noter qu'à la demande de mon maître de stage, aucun propriétaire ne peut être supprimé, afin de conserver une trace de tous les propriétaires ayant été renseigné, pour une meilleure historisation.

- [B. L'interface utilisateur](#)
- [C. Les pages Web](#)

Le code des pages web sont disponible [ici](#)

Un point est à relever sur la page proprietaires.html, l'utilisation d'une nouvelle balise, liée à l'application externe 'Django-Table2'.

Les applications externes, semblables à de nouvelles bibliothèques Django, possèdent leurs propres fonctionnement basé sur les fonctionnalités de Django. Elles permettent de simplifier davantage la création de site web.

Ici la simple ligne de code

```
{% load render_table from django_tables2 %}
```

permet d'afficher un tableau avec des colonnes personnalisées, un tri par colonnes et une pagination automatique

#### [D. Models.py](#)

Dans notre projet, l'affichage de la table via Django-Table2 n'est pas un simple affichage d'une table de la base de données, mais d'une vue SQL. Django ne prend pas en charge la création de vues SQL nativement, mais il peut les gérer sans problème avec quelques astuces.

##### a. Makemigrations & Migrate

Les commandes `makemigrations` et `migrate` sont essentielles pour gérer les modifications de la base de données :

- ⇒ `Makemigrations` : Cette commande demande à Django d'analyser le fichier `models.py` et de le comparer à ses versions précédentes. Elle crée des fichiers de migration nécessaires pour mettre à jour la base de données.
- ⇒ `Migrate` : Cette commande applique les fichiers de migration créés à la base de données existante. Elle garantit que la base de données est à jour avec les modèles définis.

L'utilisation de ces commandes présente de nombreux avantages, notamment pour le suivi des mises à jour, la maintenance et l'unicité de la base de données.

#### b. Création d'un fichier migrations vide

Pour créer une vue SQL, une migration vide est nécessaire. Voici la procédure :

Il faut d'abord créer un fichier de migration vide, avec cette ligne de commande :

```
python manage.py makemigrations --empty NomDeLAplication --name NomDeLaMigration
```

Une fois le fichier créé il faut ouvrir ce fichier et écrire la requête SQL pour créer la vue.

Enfin il faut appliquer les migrations avec la commande `migrate`. La vue SQL sera alors créée dans la base de données.

**Lien vers le code du fichier de migration**

#### c. Mise à jour du fichier Models.py

Après la création de la vue SQL, la dernière étape consiste en la mise à jour du fichier `models.py` :

- ⇒ Définir la Vue : Il faut créer une nouvelle classe dans `models.py` héritant de `models.Model`. Les attributs de cette classe correspondent aux colonnes de la vue.
- ⇒ Spécifier le Nom de la Vue : Dans la classe `Meta` de la nouvelle classe, il faut spécifier le nom de la vue SQL comme table de référence.

Cette méthode permet à Django de gérer et d'interagir avec la vue SQL comme avec une table normale, facilitant ainsi l'affichage et la manipulation des données via Django-Table2.

#### d. Tables utilisées

Pour fonctionner correctement, cette partie gestion des propriétaires fait appel à plusieurs tables, instancier dans `models.py` et héritant toutes de `models.Model`.

- nature\_juridique<sup>31</sup>
- proprietaire
- historisation\_proprietaire
- historisation\_proprietaire\_document
- adresse

#### E. *Forms.py*

Pour créer et mettre à jour un propriétaire, des formulaires sont nécessaires. Django offre d'autres types de formulaires que vue précédemment ([voir login](#)), dont le **ModelForm** qui simplifie considérablement le processus de création et de gestion des entités.

Les **ModelForm** de Django sont conçus pour automatiser la création de formulaires basés sur les modèles de la base de données. Ils génèrent automatiquement les champs du formulaire à partir des attributs du modèle spécifié. Voici les étapes pour créer un **ModelForm** :

Définir le Formulaire :

- Création d'une classe de formulaire héritant de `forms.ModelForm`.
- Dans la classe `Meta`, spécifiez le modèle associé via la variable `model`.

Personnalisation :

- Les **ModelForm** permettent une personnalisation avancée : renommer les champs, exclure certains champs, utiliser différents widgets, et limiter les choix de réponses.

Le formulaire pour les propriétaires est consultable [ici](#)

Un propriétaire possède également une adresse, stockée dans une table distincte. Pour gérer les adresses, un formulaire dédié est nécessaire. En séparant ce formulaire du formulaire des propriétaires, on permet sa réutilisation ultérieure et une meilleure organisation du code.

Pour améliorer le confort de l'utilisateur et éviter les erreurs de saisie liées à la construction de la partie adresse de la base de données, un script JavaScript communique avec l'API de la Base Adresse Nationale (BAN). Grâce à des requêtes successives à cette API, l'utilisateur peut rechercher facilement une adresse française et cliquer sur l'adresse souhaitée. Ensuite, le script assigne les valeurs de cette adresse aux champs du formulaire adresse.

---

<sup>31</sup> Permet de renseigner la nature juridique des propriétaires Personne Morale ou Personne Physique

Le formulaire pour les adresses est consultable [ici](#)

Le script pour l'autocomplétion de l'adresse est disponible [ici](#)

A la demande du service administration, une meilleure gestion de l'historisation a été demandée. Cette historisation prend également en compte la sauvegarde de justificatif. Lors de la mise à jour d'un propriétaire, l'utilisateur aura la possibilité de joindre des documents. Pour la sauvegarde de document Django nécessite un formulaire héritant de Forms et d'une table pour lier les documents sauvegardés à l'historisation.

Le formulaire pour la sauvegarde des documents est consultable [ici](#)

#### Note

Django gère la sauvegarde des fichiers en utilisant son système de gestion de fichiers intégré. Lorsqu'un fichier est soumis via un formulaire, il est traité par un champ `FileField` ou `ImageField` dans le modèle. Django stocke le fichier dans le répertoire spécifié par la variable `MEDIA_ROOT` du fichier de configuration et enregistre le chemin relatif du fichier dans la base de données. Le système de gestion des fichiers de Django permet ainsi une manipulation et une récupération faciles des fichiers téléchargés.

En utilisant les `ModelForm`, Django simplifie non seulement la création des formulaires mais aussi leur gestion et leur personnalisation, rendant le développement web plus rapide et plus efficace.

#### F. Views.py

Les vues Django sont essentielles pour gérer les opérations algorithmiques nécessaires à l'application. Chaque vue est responsable d'une page web spécifique et exécute des opérations précises. Voici un récapitulatif de leurs fonctionnalités et spécificités.

#### Vue propriétaire

Cette vue affiche un tableau des propriétaires en utilisant Django-Table2 et permet de filtrer les résultats.

- ⇒ Affichage du tableau : Utilisation de Django-Table2 pour afficher les propriétaires.
- ⇒ Filtrage : Récupération des paramètres via la requête dans l'URL pour appliquer des filtres.

#### Vue propriétaire\_detail

Cette vue affiche les détails d'un propriétaire spécifique, récupérant plusieurs types d'informations.

- ⇒ Récupération du propriétaire : Identification du propriétaire grâce à son ID.
- ⇒ Parcelles possédées : Récupération des parcelles détenues par le propriétaire.
- ⇒ Associations Syndicales (AS) : Récupération des AS dont le propriétaire est membre.
- ⇒ Historisation : Récupération des historiques du propriétaire et de ses parcelles.

#### Vue `create_proprietaire`

Cette vue permet la création d'un nouveau propriétaire et de son adresse associée.

- ⇒ Instanciation des formulaires : Formulaires pour le propriétaire et l'adresse.
- ⇒ Vérification des formulaires : Validation des données saisies.
- ⇒ Attribution et sauvegarde : Attribution de l'adresse au propriétaire et sauvegarde dans la base de données.
- ⇒ Message utilisateur : Affichage d'un message indiquant la réussite ou l'échec de la sauvegarde.

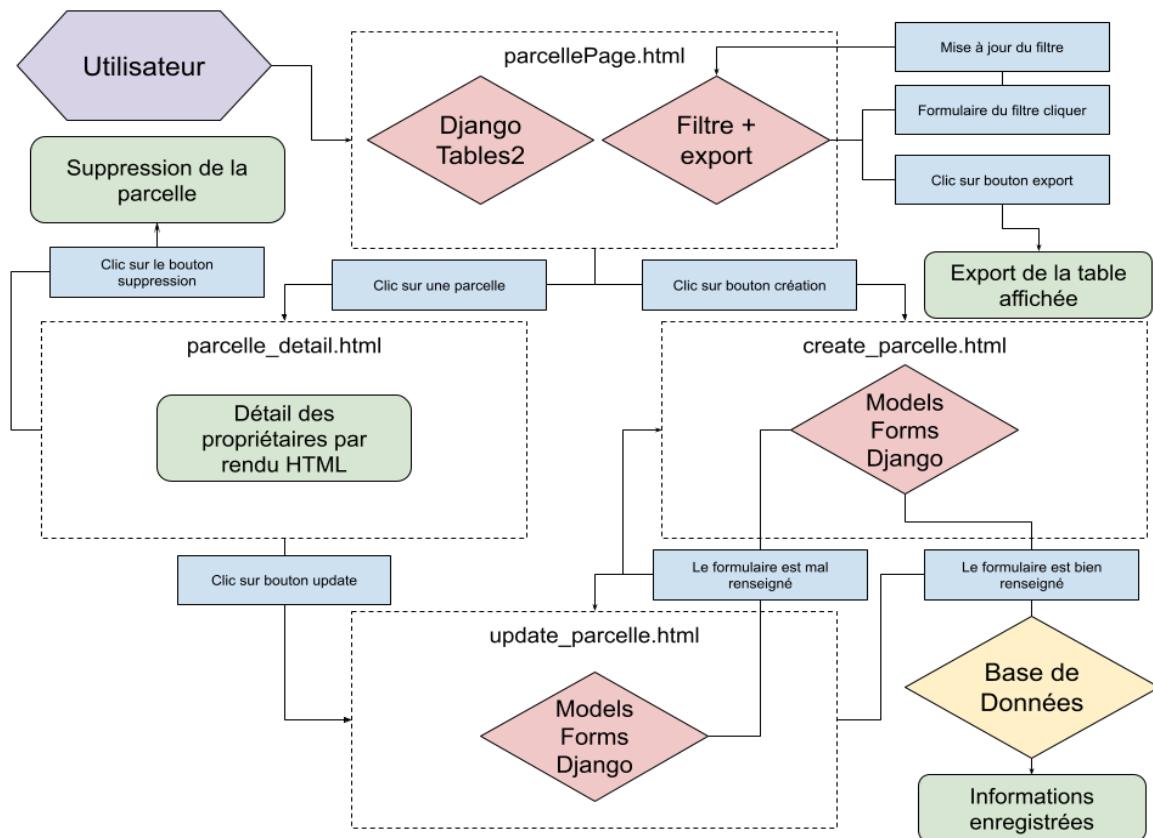
#### Vue `update_proprietaire`

Cette vue permet la mise à jour des informations d'un propriétaire, y compris la gestion des documents justificatifs et l'historisation.

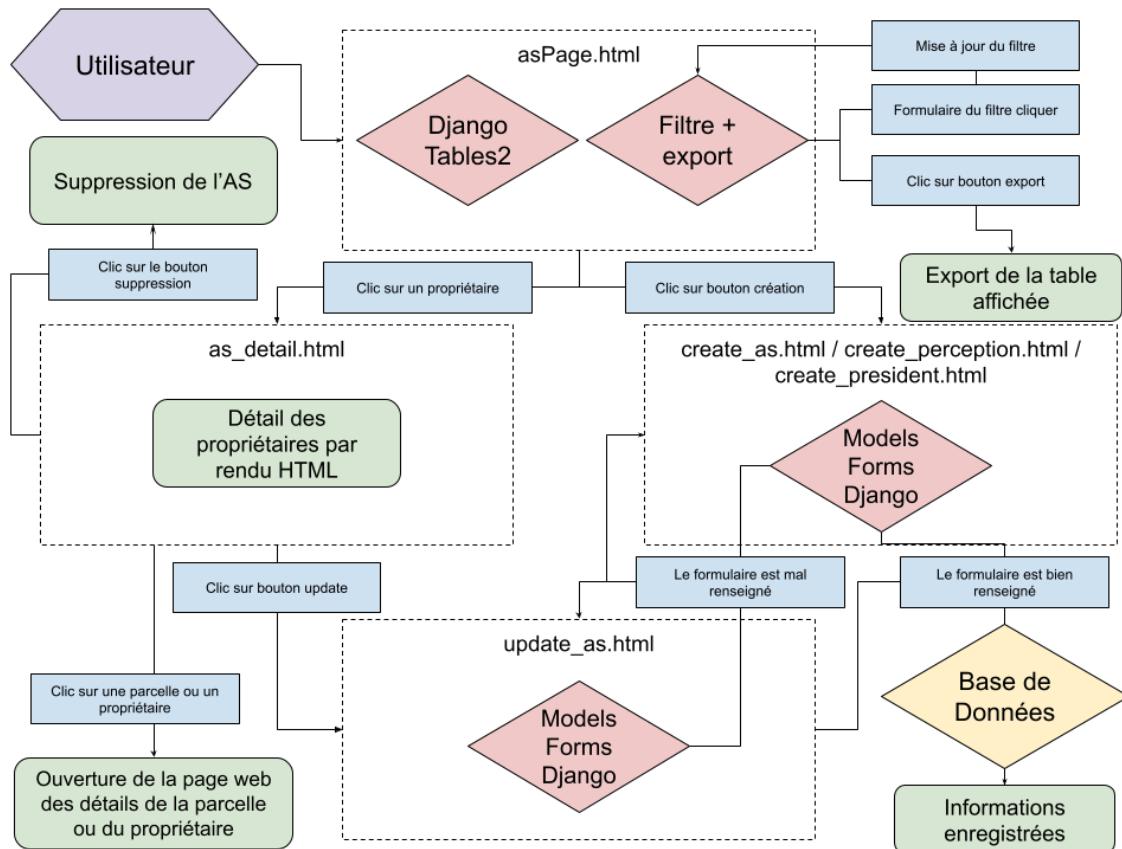
- ⇒ Instanciation des formulaires : Formulaires pour le propriétaire et l'adresse.
- ⇒ Récupération des documents : Utilisation de la méthode FILES pour gérer les documents.
- ⇒ Vérification et sauvegarde : Validation des formulaires et sauvegarde des anciennes et nouvelles valeurs pour l'historisation.
- ⇒ Message utilisateur : Affichage d'un message indiquant la réussite ou l'échec de la sauvegarde.

### 3.5.2 La gestion des parcelles et des Association Syndicales

La logique derrière ces deux fonctionnalités est identique à celle des propriétaires, leurs parties techniques ne seront donc pas abordées.



Logique de la gestion des parcelles



Logique de la gestion des AS

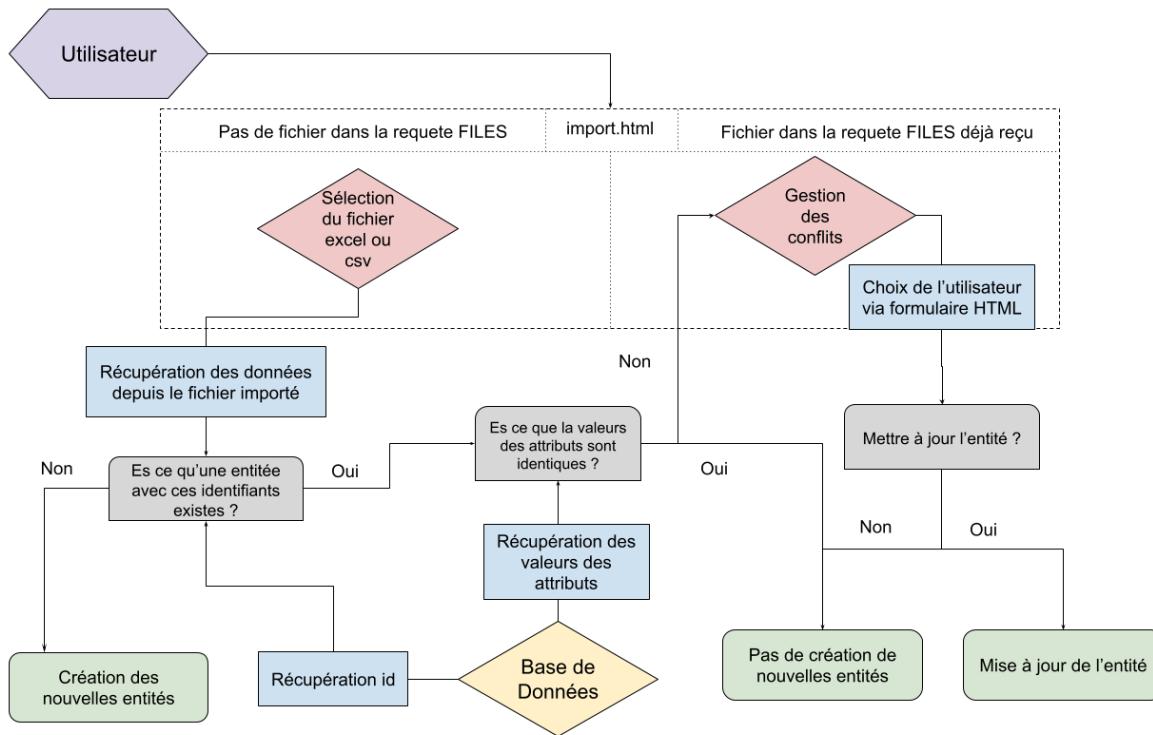
### 3.5.3 La gestion des imports et des conflits

Dernière grosse partie aborder pendant ce stage, la gestion des imports des données dans la base de données

Comme vu dans le cahier des charges, les données sont transmises via un fichier excel.

Une fonctionnalité à d'abord été testé avec un simple script python (consultable [ici](#)). Une fois la validité du script effectué, il a été converti pour fonctionner sous django (views, templates html, et css).

## A. Logique



Logique de la gestion des AS

La logique derrière est la suivante :

1. L'utilisateur importe le fichier excel ou csv issu de la requête SQL réalisé par le pôle SIG
2. Le programme vérifie si une entité avec des identifiant similaire existe déjà
3. Si non il va créer une nouvelle entité
4. Si oui l'application vérifie si les données sauvegardées en base de données et les données importées sont identiques
5. Si les données sont identiques, il n'y a pas de création d'entité et l'application passe à l'entité suivantes
6. Si les données sont différentes, les valeurs des attributs présent en base de données et dans le fichiers importé sont enregistrées et passé en contexte à la page HTML
7. L'utilisateur choisi alors l'action à réaliser grâce à un formulaire HTML
8. Si il préfère garder les informations sauvegardées en base de données, il n'y a pas de mise à jour de l'entité et passe à la suivante.

9. Si l'utilisateur préfère importer les nouvelles données, l'entité est mise à jour.

## B. Importation des données

### a. Import du fichier

Les données sont transmises au service administratif via un fichier Excel. Pour importer ces données dans la base de données de l'application, le service utilise un formulaire HTML [1] pour sélectionner le fichier. Une fois le fichier choisi, il est affiché [2], puis l'utilisateur clique sur "Importer" [3].

## Import Data

The screenshot shows a user interface for importing data. On the left, there is a text input field with placeholder text 'Sélectionnez un fichier :'. To its right is a blue button labeled 'Parcourir...'. Next to the button is a message 'Aucun fichier sélectionné.' In the top right corner of the input field, there is a small red square containing the number '1'. To the right of the message, there is a green button labeled 'Importer'. In the top right corner of the 'Importer' button, there is a small red square containing the number '3'. Between the message and the 'Importer' button, there is another small red square containing the number '2'.

L'application web va alors réaliser un script pour vérifier l'existence ou non de la donnée en BD, puis va alimenter un dictionnaire pour sauvegarder ses conflits, si besoin.

Pour se faire, la view 'import\_date' récupère d'abord le fichier transmis dans la méthode post du formulaire. Une fois le fichier récupéré, et grâce à la bibliothèque Pandas, il est lu, indépendamment du type de fichier, les données sont nettoyées puis le processus commence.

### Note

Pandas est une bibliothèque écrite en Python, permettant la manipulation et l'analyse de données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques.

Son nom est dérivé du terme Panel Data (en français "données de panel"). Son nom est également un jeu de mots sur l'expression "Python Data Analysis".

### b. Fonction de nettoyage des données

`trim_row_data` parcourt chaque champ de la ligne de données et supprime les espaces en trop autour des chaînes de caractères, ce qui est essentiel pour éviter les erreurs de correspondance et les incohérences dans les noms et adresses.

`clean_data` convertit les valeurs en chaînes de caractères et supprime les espaces superflus, même pour les types de données numériques, assurant ainsi que toutes les données sont uniformes avant traitement.

`convertir_date` est également cruciale pour le nettoyage, car elle convertit les dates de naissance au format français ('jour/mois/année') en format de date standard PostgreSQL ('année-mois-jour'), tout en gérant les erreurs potentielles et les valeurs nulles.

### c. Processus

Le processus va, ligne par ligne, vérifier si les informations existent déjà dans la base de données, s'il est nécessaire de les créer ou si un conflit apparaît.

Quatre notions d'informations sont transmises dans les fichiers excel. L'Association Syndicale (AS) de la parcelle, les informations de la parcelle, du propriétaire, et l'adresse du propriétaire.

`get_or_create_as_unima` récupère le numéro d'AS (Association Syndicale) de la parcelle depuis la colonne `as_num_unima`. Elle vérifie ensuite si ce numéro existe déjà dans la base de données. Si le numéro est trouvé, la méthode renvoie l'objet correspondant ; sinon, elle crée un nouvel objet avec les données fournies ainsi que des valeurs par défaut.

le code de cette fonction est disponible [ici](#)

`get_or_create_adresse` vérifie la présence d'une adresse existante ou créer une nouvelle adresse.

le code de cette fonction est disponible [ici](#)

`get_or_create_proprietaire` vérifie la présence d'un propriétaire avec le même identifiant ou non. Cette vérification se fait avec une combinaison de trois champs, le nom, le prénom et la date de naissance. Si le propriétaire n'existe pas, il est créé. En cas contraire, le processus vérifie les différences entre les données importées et les données sauvegardées. En cas de différence, un dictionnaire est ajouté à la liste des conflits. Ce dictionnaire contient l'objet sauvegardé en base de données, l'objet importé, et enfin le type de conflit, ici propriétaire. Dans tous les cas, la méthode renvoie un objet propriétaire, créé ou existant.

le code de cette fonction est disponible [ici](#)

`get_or_create_parcelle` fonctionne exactement comme '`get_or_create_proprietaire`' retournant un objet `parcelle`.

le code de cette fonction est disponible [ici](#)

### C. Affichage des conflits

Une fois l'intégralité des données traitées, la view `import_data` est rappelée avec les listes des conflits passées en contexte.

Si ces listes ne sont pas vides, le template HTML Django va afficher les différents conflits à l'utilisateur.

## Conflits détectés

### Propriétaires [1]

| Champs [2]                        | Données importées [3]           | Données existantes [4]                    | Action                                                                                                                                                                                                                                                 |
|-----------------------------------|---------------------------------|-------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Nom                               | COMMUNE D ANGOULINS             | COMMUNE D ANGOULINS                       |                                                                                                                                                                                                                                                        |
| Prénom                            |                                 |                                           |                                                                                                                                                                                                                                                        |
| Date de naissance                 |                                 |                                           |                                                                                                                                                                                                                                                        |
| Adresse                           | LE BOURG, 17690 ANGOULINS       | AV DU COMMANDANT LISIACK, 17690 ANGOULINS | <div style="border: 1px solid #ccc; padding: 5px; width: fit-content;"> <input checked="" type="checkbox"/> Garder l'existant [5] <br/> <input type="checkbox"/> Garder l'existant <br/> <input type="checkbox"/> Remplacer par importé         </div> |
| <a href="#">Tout importer</a> [6] | <a href="#">Tout garder</a> [7] | <a href="#">Valider les choix</a> [8]     |                                                                                                                                                                                                                                                        |

Le type de conflit est d'abord affiché [1], suivi de la présentation des données sous forme de tableau [2]. Ce tableau compare les données du fichier Excel [3] avec celles présentes dans la base de données [4]. L'utilisateur peut choisir l'action à entreprendre pour chaque conflit [5], ou opter pour importer toutes les données [6] ou ne rien importer [7]. Si l'utilisateur décide d'appliquer des actions spécifiques pour chaque conflit, il doit valider le formulaire [8].

### D. Gestion des conflits

Quatre options doivent être gérées en fonction du résultat du formulaire envoyé par l'utilisateur :

- ⇒ Tout importer
- ⇒ Ne rien importer
- ⇒ Importer au cas par cas
- ⇒ Ne pas importer au cas par cas

Si l'utilisateur choisit "Ne rien importer" (option "Tout garder" cliquée), le processus s'arrête sans modification des données. La même logique s'applique pour "Ne pas importer au cas par cas".

En revanche, si l'utilisateur opte pour "Tout importer" ou "Importer au cas par cas", la vue `import_data` récupère les informations transmises via la méthode POST, stockées dans des champs cachés dans le formulaire HTML. La vue attribue ensuite ces valeurs aux objets propriétaires ou parcelles existantes, selon le type de conflit à résoudre. Ces objets sont identifiés grâce aux identifiants fournis par la méthode POST.

#### E. *Code complet*

Le code complet de la view gérant l'importation et la gestion des conflits se trouve [ici](#)

Le code HTML de la page gérant l'importation et la gestion des conflits se trouve [ici](#)

---

## 4. POUR ALLER PLUS LOIN

---

### 4.1 La solution de Récolement de l'information

#### 4.1.1 État Actuel

La solution de récolelement développée est opérationnelle mais limitée par un compte gratuit QFieldCloud, permettant une connexion unique à la fois. Cependant, pour une utilisation simultanée par plusieurs utilisateurs sur des projets communs ou réalisés en parallèle, deux solutions de déploiement sont envisageables.

#### 4.1.2 Abonnement QFieldCloud Organization

Souscrire à un abonnement QFieldCloud Organization à hauteur de 20 €/mois/utilisateur. Cet abonnement offre :

- Un grand espace de stockage
- La gestion de plusieurs utilisateurs simultanés
- La possibilité de travailler sur plusieurs projets en parallèle
- Une assistance et une aide technique

Pour Unima, cela impliquerait un abonnement pour le Bureau d'étude, un pour les conducteurs de travaux et deux pour les pelleteurs, soit un coût total de 80 €/mois. Bien que cette solution soit simple et efficace, elle représente un coût non négligeable.

#### 4.1.3 Déploiement d'un Serveur QFieldCloud Personnel

Une alternative serait de déployer un serveur QFieldCloud personnel. Une documentation détaillée existe pour l'installation de ce type de serveur, nécessitant la location d'un serveur communiquant avec l'extérieur. Cette option offre plusieurs avantages :

- Gestion plus fine de l'espace de stockage
- Création et gestion des utilisateurs sans contraintes

Des offres de serveurs privés virtuels (VPS) sont disponibles à partir de 4,60 €/mois. Bien que la maintenance soit manuelle, cette solution est la plus efficace pour

pérenniser et étendre la solution de récolement à plus grande échelle, offrant une flexibilité maximale à l'Unima.

## 4.2 L'application Django

### 4.2.1 Phase Expérimentale

L'application est actuellement à une phase expérimentale de développement. Bien que fonctionnelle, elle nécessite plusieurs mises à jour importantes pour répondre pleinement aux exigences du cahier des charges. Trois fonctionnalités essentielles manquent encore : le calcul des factures, le calcul des voix et le publipostage. Avec un mois de développement supplémentaire, ces fonctionnalités pourraient être aisément implémentées.

### 4.2.2 Optimisation et Interface Utilisateur

En tant que première application Django conçue, certaines parties du code devront être optimisées. L'interface utilisateur, bien que fonctionnelle, reste sommaire et mérite d'être retravaillée. Bien que ce ne fût pas l'objectif principal du stage, une interface plus intuitive et esthétique serait bénéfique pour les utilisateurs finaux.

### 4.2.3 Fichier Excel

Se transmettre les informations via un fichier Excel n'est pas le meilleur moyen pour partager des données. Par exemple, les fichiers Excel peuvent être facilement corrompus, les modifications simultanées par plusieurs utilisateurs peuvent créer des conflits, et il y a un risque élevé d'erreurs humaines lors de la saisie des données. En effet, il serait beaucoup plus intéressant de récupérer les données à comparer directement depuis une base de données ou une vue. Cela limiterait les risques et permettrait une meilleure automatisation de l'application, comme par exemple via une mise à jour avec un simple bouton.

Ce point est cependant soumis à réserve, car même si le pôle SIG extrait actuellement les données depuis une requête SQL, ils peuvent apporter des modifications aux entités transmises, ce qui complique la situation.

### 4.2.4 Ressources Nécessaires pour Finalisation

Des moyens importants devront être déployés par Unima pour finaliser ce projet. Les fondations sont solides, mais des efforts supplémentaires sont nécessaires pour

atteindre les objectifs fixés. En particulier, la gestion des adresses nécessite une refonte pour correspondre à la vision initiale, avec une base d'adresses plus structurée.

#### 4.2.5 Expérience et Problèmes

Les problèmes rencontrés et les retards accumulés soulignent le manque d'expérience avec le framework Django avant le début de ce projet. Cette courbe d'apprentissage a impacté le développement, mais a également permis de poser des bases solides pour une application robuste et extensible.

---

## 5. CONCLUSION

---

---

## ANNEXE 1

### *Code du plugin Récolelement Unima*

---

etape1\_dialog.py programme complet

```
-*- coding: utf-8 -*-
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5 import uic

from QGIS.core import *
from QGIS.gui import *
import sys, os
import psycopg2
import processing
from db_manager.db_plugins import createDbPlugin

ui_path = os.path.dirname(os.path.abspath(__file__))
ui_path = os.path.join(ui_path, "forms")
form_action1, _ = uic.loadUiType(os.path.join(ui_path, "etape_1.ui"))

class Etape1_dialog(QDialog, form_action1):
 def __init__(self, interface):
 QWidget.__init__(self)
 self.setupUi(self)
 self.iface = interface
 self.nomBD = "unima"
 self.schema_reseau = "reseau"
 self.schema_limitesAS = 'administratif'
 self.schema_regieTravaux = "regie_travaux"
 self.as_dict = {}

 self.CB_choixAs.setStyleSheet("combobox-popup: 0;")
 self.ajoutCB_choixAs()
 self.PB_folder.clicked.connect(self.recupFichierSauvegarde)
 self.LE_choixPath.textChanged.connect(self.recupPathSauvegarde)
 self.PB_ok.clicked.connect(self.accept)
 self.PB_annuler.clicked.connect(self.reject)

 self.get_info_postgis(self.nomBD)

 def get_info_postgis(self, wnom):
 # renvoie l'hôte, le port, le nom de la DB, le user et le pwd sous forme de tuple
 resu = None
```

```

dbpluginclass = createDbPlugin("postgis")

for c in dbpluginclass.connections():
 if c.connectionName() == wnom:
 settings = QgsSettings()
 settings.beginGroup(u"/%s/%s" % ('PostgreSQL/connections', wnom))

 if settings.contains("database"):
 whost = settings.value("host")
 wport = settings.value("port")
 wdatabase = settings.value("database")
 wusername = settings.value("username")
 wpASSWORD = settings.value("password")

 resu = (whost, wport, wdatabase, wusername, wpASSWORD)
 return resu

def ajoutCB_choixAs(self):
 # Ajoute à la comboBox CB_choixAs le noms des AS depuis la table limites_as et peuplement du dictionnaire
 self.as_dict avec [nom]=numeros_as
 self.CB_choixAs.clear()

 try:
 whost, wport, wdatabase, wusername, wpASSWORD = self.get_info_postgis(self.nomBD)
 conn = psycopg2.connect(
 host=whost,
 port=wport,
 database=wdatabase,
 user=wusername,
 password=wpASSWORD
)
 schema = self.schema_limitesAS
 #schema = "administratif"
 cursor = conn.cursor()
 cursor.execute(f"SELECT nom, n_unima FROM {schema}.limites_as ORDER BY nom;")
 results = cursor.fetchall()
 conn.commit()
 conn.close()

 # Cr éation du dictionnaire self.as_dict avec result[0] comme clé et result[1] comme valeur
 for result in results:
 key = result[0]
 value = result[1]
 self.as_dict[key] = value

 # Ajout des clés du dictionnaire à la combobox
 for key in self.as_dict:
 self.CB_choixAs.addItem(key)

 except Exception as e:
 print("Erreur:", e)

```

```

def recuperFichierSauvegarde(self):
 # fonction pour enregistrer le chemin de sortie lors de l'utilisation du .getExistingDirectory (fenetre native)
 folder = QFileDialog.getExistingDirectory(self)
 folder = folder + "/"
 self.LE_choixPath.setText(folder)

def recuperPathSauvegarde(self):
 # fonction pour enregistrer lors de l'écriture dans le QLineEdit
 folder = self.LE_choixPath.text()
 folder = folder.replace("\\\\, \"/")
 folder = folder + '/' # sinon on se retrouve dans le dossier du dessus
 return folder

def recuperIdAschoisi(self):
 # Retourne les numéros Unia de l'AS choisi via le dictionnaire self.as_dict
 cleAs = self.CB_choixAs.currentText()
 numAs = self.as_dict[cleAs]

 return numAs

def ajoutReseauSyndical(self): #RETOUR AU RAPPORT
 # Ajout de la couche de référence
 self.drop_layer("reseau_syndical")
 try:
 # Déballage des informations de connexion
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 schema = self.schema_reseau
 numAS = self.recuperIdAschoisi()

 # Crédit de l'URI de la source de données
 uri = QgsDataSourceUri()
 uri.setConnection(whost, str(wport), wdatabase, wusername, wpassword)
 uri.setDataSource(schema, 'reseau_syndical', 'geom')

 # Crédit de la couche vectorielle à partir de l'URI
 sql_query = f"""
 "id_as" LIKE '{numAS}'
 """
 uri.setSql(sql_query) # Utilisation de setSql pour spécifier la requête SQL
 la_couche_resultat = QgsVectorLayer(uri.uri(), "reseau_syndical", "postgres")

 if la_couche_resultat.isValid():
 QgsProject.instance().addMapLayer(la_couche_resultat)
 print('Couche résultat ajoutée avec succès')
 else:
 print('Erreur: La couche résultat n\'est pas valide')
 return la_couche_resultat
 except Exception as e:
 message = "Erreur : %s" % str(e)
 self.warning(message)

```

```

def ajoutLimitesAs(self):
 # Ajout de la couche limites_as
 self.drop_layer("limites_as")
 try:
 # Déballage des informations de connexion
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 schema = self.schema_limitesAS
 numAS = self.recupIdAschoisi()

 # Création de l'URI de la source de données
 uri = QgsDataSourceUri()
 uri.setConnection(whost, str(wport), wdatabase, wusername, wpassword)
 uri.setDataSource(schema, 'limites_as', 'geom')

 # Création de la couche vectorielle à partir de l'URI
 sql_query = f"""
 "n_unima" LIKE '{numAS}'
 """
 uri.setSql(sql_query) # Utilisation de setSql pour spécifier la requête SQL
 la_couche_resultat = QgsVectorLayer(uri.uri(), "limites_as", "postgres")

 if la_couche_resultat.isValid():
 QgsProject.instance().addMapLayer(la_couche_resultat)
 print('Couche résultat ajoutée avec succès')
 else:
 print('Erreur: La couche résultat n\'est pas valide')
 return la_couche_resultat
 except Exception as e:
 message = "Erreur : %s" % str(e)
 self.warning(message)

def ajoutCoucheObservation(self):
 # Nom de la couche à créer
 nom_couche = "observations_remarques"

 # Définir le type de géométrie de la couche (dans ce cas, des points)
 type_geometrie = QgsWkbTypes.PointGeometry

 # Créer la couche de points vide avec le schéma défini
 couche_points_vide = QgsVectorLayer(
 "Point?crs=epsg:2154&field=fid:integer&index=yes", # Définir le type de géométrie et le CRS (EPSG:4326 pour WGS
 84)
 nom_couche,
 "memory" # Stocker la couche en mémoire
)

 # Ajouter les champs à la couche
 couche_points_vide.dataProvider().addAttributes([
 (QgsField("fid", QVariant.Int)),
 (QgsField("hio_commentaire", QVariant.String)),
 (QgsField("hio_photo", QVariant.String)),
 (QgsField("obs_id", QVariant.Int)),
 (QgsField("res_code_tr", QVariant.String)),
 (QgsField("hio_date", QVariant.Date)),
])

```

```

 (QgsField("as_n_unima", QVariant.String))]

Mettre à jour le cache des champs
couche_points_vide.updateFields()

Ajouter la couche à la liste des couches de QGIS
QgsProject.instance().addMapLayer(couche_points_vide)

return couche_points_vide

def exportVecto(self):
 # Export des couches récupéré depuis la BD Postgres
 crs = QgsCoordinateReferenceSystem(2154)
 folder = self.recupPathSauvegarde()
 # Export reseau_syndical
 layer_temp_export = self.ajoutReseauSyndical()
 filename = "lineaire_curage"
 filepath = os.path.join(folder, filename)

 error, message = QgsVectorFileWriter.writeAsVectorFormat(layer_temp_export, filepath, 'UTF-8', crs,
driverName='GPKG')
 if error == QgsVectorFileWriter.NoError:
 message += "Exportation Réussi"
 print(message)
 else:
 message += "Erreur lors de l'exportation"
 print(message)

 # Export limites_as
 layer_temp_export = self.ajoutLimitesAs()
 filename = "limites_as"
 filepath = os.path.join(folder, filename)

 error, message = QgsVectorFileWriter.writeAsVectorFormat(layer_temp_export, filepath, 'UTF-8', crs,
driverName='GPKG')
 if error == QgsVectorFileWriter.NoError:
 message += "Exportation Réussi"
 print(message)
 else:
 message += "Erreur lors de l'exportation"
 self.warning(message)

 # Export couche observation
 layer_temp_export = self.ajoutCoucheObservation()
 filename = "observations_remarques"
 filepath = os.path.join(folder, filename)

 error, message = QgsVectorFileWriter.writeAsVectorFormat(layer_temp_export, filepath, 'UTF-8', crs,
driverName='GPKG')
 if error == QgsVectorFileWriter.NoError:
 message += "Exportation Réussi"
 print(message)
 else:

```

```

message += " Erreur lors de l'exportation"
self.warning(message)

def ajoutCoucheExporter(self):
 # On enlève les couches issus de la BD et temporaire. Puis on ajoute les couches exportées au projet.
 self.drop_layer("reseau_syndical")
 self.drop_layer("limites_as")
 self.drop_layer("lineaire_curage")
 self.drop_layer("observations_remarques")

 folder = self.recupPathSauvegarde()
 folderLineaire = folder + 'lineaire_curage.gpkg'
 folderLimites = folder + 'limites_as.gpkg'
 folderObservaiton = folder + 'observations_remarques.gpkg'

 coucheLineaire = QgsVectorLayer(folderLineaire, 'lineaire_curage', "ogr")
 # Vérifier si la couche a été chargée avec succès
 if not coucheLineaire.isValid():
 print("Impossible de charger la couche.")
 else:
 # Ajouter la couche à la carte de QGIS, sans l'afficher (paramètre False)
 QgsProject.instance().addMapLayer(coucheLineaire, False)

 coucheLimites = QgsVectorLayer(folderLimites, 'limites_as', "ogr")
 # Vérifier si la couche a été chargée avec succès
 if not coucheLimites.isValid():
 print("Impossible de charger la couche.")
 else:
 # Ajouter la couche à la carte de QGIS, sans l'afficher (paramètre False)
 QgsProject.instance().addMapLayer(coucheLimites, False)

 coucheObservation = QgsVectorLayer(folderObservaiton, 'observations_remarques', "ogr")
 # Vérifier si la couche a été chargée avec succès
 if not coucheObservation.isValid():
 print("Impossible de charger la couche.")
 else:
 # Ajouter la couche à la carte de QGIS, sans l'afficher (paramètre False)
 QgsProject.instance().addMapLayer(coucheObservation, False)

def ajoutBaseMap(self): #RETOUR AU RAPPORT
 # On ajoute ici la couche raster ortho_2021.vrt. On ne fera pas encore de masque sur cette couche, on le fera
 uniquement lors de la préparation de l'exportation
 self.drop_layer("basemap")
 folderBaseMap = '//SRV-FILER/Stock/dao_sig/raster/Ortho/ortho_2021.vrt'
 baseMap = QgsRasterLayer(folderBaseMap, 'basemap', "gdal")
 # Vérifier si la couche a été chargée avec succès
 if not baseMap.isValid():
 print("Impossible de charger la couche.")
 else:
 # Ajouter la couche à la carte de QGIS, sans l'afficher (paramètre False)
 QgsProject.instance().addMapLayer(baseMap, False)

```

```

def styleCoucheObservation(self):
 # On formate la couche 'observations_remarques' avec des valeurs par défauts, l'application d'un style, et la mise à jour d'une ValueMap
 if len(QgsProject.instance().mapLayersByName("observations_remarques")) > 0:
 layer = QgsProject.instance().mapLayersByName("observations_remarques")[0]

 # Champ 'fid' autoincrémenté avec une valeur par défaut
 default_val = QgsDefaultValue(
 """
 CASE
 WHEN maximum("fid") is NULL THEN 1
 WHEN "fid" is NULL THEN maximum("fid")+1
 ELSE "fid"
 END
 """,
 applyOnUpdate=True)

 layer.setDefaultDefaultValueDefinition(layer.fields().lookupField('fid'), default_val)

 # Chargement du style
 style_path = os.path.dirname(os.path.abspath(__file__))
 style_path = os.path.join(style_path, "style")
 style_path = os.path.join(style_path, "style_ponctuel.qml")
 # Vérification si le fichier de style existe
 if os.path.exists(style_path):
 # Charger le style uniquement si le fichier existe
 layer.loadNamedStyle(style_path)
 layer.saveStyleToDatabase(name='style_ponctuel', description='Style par défaut de la couche observations_remarques', useAsDefault=True, uiFileContent="")
 print("Le fichier de style existe :", style_path)
 else:
 print("Le fichier de style n'existe pas :", style_path)

 # Mis en place de la value map pour le type d'observation #RETOUR AU RAPPORT
 # Importation de la table 'observation_type' pour récupérer les valeurs stockés dans cette table
 self.drop_layer("observation_type")
 try:
 # Déballage des informations de connexion
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 schema = self.schema_regieTravaux

 # Création de l'URI de la source de données
 uri = QgsDataSourceUri()
 uri.setConnection(whost, str(wport), wdatabase, wusername, wpassword)
 uri.setDataSource(schema, 'observation_type', '')

 la_couche_resultat = QgsVectorLayer(uri.uri(), "observation_type", "postgres")

 if la_couche_resultat.isValid():
 QgsProject.instance().addMapLayer(la_couche_resultat)
 print('Couche résultat ajoutée avec succès')

```

```

else:
 print('Erreur: La couche résultat n\'est pas valide')
except Exception as e:
 message = "Erreur : %s" % str(e)
 self.warning(message)

Récupération des donnée depuis la tables + formatage pour la ValueMap
layer = QgsProject.instance().mapLayersByName('observation_type')[0]

Initialisez une liste pour stocker les dictionnaires
liste_dictionnaires = []

Parcourez les entités de la couche
for feature in layer.getFeatures():
 # Récupérez les valeurs des champs 'obs_id' et 'obs_nom' pour chaque entité
 obs_id = feature['obs_id']
 obs_nom = feature['obs_nom']

 # Créez un dictionnaire avec 'obs_id' comme clé et 'obs_nom' comme valeur
 dictionnaire = {obs_nom:obs_id}

 # Ajoutez ce dictionnaire à la liste
 liste_dictionnaires.append(dictionnaire)

layerpoint = QgsProject.instance().mapLayersByName('observations_remarques')[0]

Récupéré l'index du champ 'obs_id'
idx = layerpoint.fields().lookupField('obs_id')

Définir la ValueMap
v_map = {'map': liste_dictionnaires}

Définir la configuration du widget de l'éditeur sur la couche, en passant par l'index du champ et la configuration du
widget.
layerpoint.setEditorWidgetSetup(idx, QgsEditorWidgetSetup('ValueMap', v_map))

Suppression de la table du projet
self.drop_layer("observation_type")
layerpoint.saveStyleToDatabase(name='style_ponctuel', description='Style par défaut de la couche
observations_remarques', useAsDefault=True, uiFileContent=")

def styleCoucheLineaire(self): #RETOUR AU RAPPORT
 # On formate la couche 'lineaire_curage' avec rename, ajouts et suppression de champs, l'application d'un style, et la
 mise à jour d'une ValueMap
 self.rename_champ("lineaire_curage","code_tr","res_code_tr")
 self.rename_champ("lineaire_curage","type","res_type")

layer = QgsProject.instance().mapLayersByName('lineaire_curage')[0]
Liste des chaps à supprimer
champs_a_supprimer = ['id','nom','adherent','loi_eau','num','valid_ddtm','login','ancien_codetr','connexion']
Enter Edit mode
with edit(layer):

```

```

Initialisation liste des indexs des champs à supprimer
index_champs = []
Itération sur les champs contenu dans la liste 'champs_a_supprimer'
for champ_a_supprimer in champs_a_supprimer:
 # récupération de l'index par le nom du champ:
 fieldindex_to_delete = layer.fields().indexFromName(champ_a_supprimer)
 # Vérification si le champ existe
 if fieldindex_to_delete == -1:
 # Si il n'existe pas on passe au prochain sans provoquer de crash
 continue
 index_champs.append(fieldindex_to_delete)

Supprime les champs par leurs index
layer.dataProvider().deleteAttributes(index_champs)
Update des changement effectué
layer.updateFields()

Ajouter les champs à la couche #RETOUR AU RAPPORT
Définir la limite de caractères pour les champs de type chaîne de caractères
max_length = 20
layer.dataProvider().addAttributes([(QgsField("hit_annee", QVariant.Int),
 (QgsField("hit_tranche", QVariant.Int)),
 (QgsField("hit_bro_cot", QVariant.Int)),
 (QgsField("hit_aplo_cot", QVariant.Int)),
 (QgsField("hit_cot_vase", QVariant.Int)),
 (QgsField("hit_temps_broyage", QVariant.Int)),
 (QgsField("hit_temps_troncottage", QVariant.Int)),
 (QgsField("hit_verif_depotvase", QVariant.Int)),
 (QgsField("hit_verif_curage", QVariant.Int)),
 (QgsField("hit_hauteur_vase", QVariant.Double)),
 (QgsField("hit_largeur_fosse", QVariant.Double)),
 (QgsField("pel_id", QVariant.Int)),
 (QgsField("hit_num_tranche_be", QVariant.String, len=max_length)),
 (QgsField("hit_num_devis", QVariant.String, len=max_length)),
 (QgsField("hit_date", QVariant.Date)))])

Mettre à jour le cache des champs
layer.updateFields()

Chargement du style #RETOUR AU RAPPORT
style_path = os.path.dirname(os.path.abspath(__file__))
style_path = os.path.join(style_path, "style")
style_path = os.path.join(style_path, "style_lineaire_be.qml")
Vérification si le fichier de style existe
if os.path.exists(style_path):
 # Charger le style uniquement si le fichier existe
 layer.loadNamedStyle(style_path)
 layer.setStyleToDatabase(name='style_lineaire_be', description='Style par défaut de la couche lineaire_curage
utilisé par le BE', useAsDefault=True, uiFileContent="")
 print("Le fichier de style existe :", style_path)
else:
 print("Le fichier de style n'existe pas :", style_path)

```

```

Mis en place de la value map pour les pelleteurs
Importation de la table pelleteur pour récupérer les valeurs stockés dans cette table
self.drop_layer("pelleteur")
try:
 # Déballage des informations de connexion
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 schema = self.schema_regieTravaux

 # Création de l'URI de la source de données
 uri = QgsDataSourceUri()
 uri.setConnection(whost, str(wport), wdatabase, wusername, wpassword)
 uri.setDataSource(schema, 'pelleteur', '')

 la_couche_resultat = QgsVectorLayer(uri.uri(), "pelleteur", "postgres")

 if la_couche_resultat.isValid():
 QgsProject.instance().addMapLayer(la_couche_resultat)
 print('Couche résultat ajoutée avec succès')
 else:
 print('Erreur: La couche résultat n\'est pas valide')
except Exception as e:
 message = "Erreur : %s" % str(e)
 self.warning(message)

Récupération des données depuis la table + formatage pour la ValueMap
layer = QgsProject.instance().mapLayersByName('pelleteur')[0]

Initialisez une liste pour stocker les dictionnaires
liste_dictionnaires = []

Parcourez les entités de la couche
for feature in layer.getFeatures():
 # Récupérez les valeurs des champs 'obs_id' et 'obs_nom' pour chaque entité
 pel_id = feature['pel_id']
 pel_nom = feature['pel_nom']

 # Créez un dictionnaire avec 'obs_id' comme clé et 'obs_nom' comme valeur
 dictionnaire = {pel_nom:pel_id}

 # Ajoutez ce dictionnaire à la liste
 liste_dictionnaires.append(dictionnaire)

liste_dictionnaires.append({"Non renseigné": 'null'})
layerLineaire = QgsProject.instance().mapLayersByName('lineaire_courage')[0]

Récupérez l'index du champ 'pel_id'
idx = layerLineaire.fields().lookupField('pel_id')
Définir la ValueMap
v_map = {'map': liste_dictionnaires}

Définir la configuration du widget de l'éditeur sur la couche, en passant par l'index du champ et la configuration du
widget.
layerLineaire.setEditorWidgetSetup(idx, QgsEditorWidgetSetup('ValueMap', v_map))

```

```

Suppression de la table du projet
self.drop_layer("pelleteur")

def styleLimitesAS(self): #RETOUR AU RAPPORT
 # On formate la couche 'Limites_as' avec l'application d'un style
 layer = QgsProject.instance().mapLayersByName('limites_as')[0]
 # Chargement du style
 style_path = os.path.dirname(os.path.abspath(__file__))
 style_path = os.path.join(style_path, "style")
 style_path = os.path.join(style_path, "style_limites_as.qml")
 # Vérification si le fichier de style existe
 if os.path.exists(style_path):
 # Charger le style uniquement si le fichier existe
 layer.loadNamedStyle(style_path)
 layer.saveStyleToDatabase(name='style_limites_as', description='Style par défaut de la couche limites_as',
useAsDefault=True, uiFileContent="")
 print("Le fichier de style existe :", style_path)
 else:
 print("Le fichier de style n'existe pas :", style_path)

def ordreCouches(self):
 #TODO
 #Changer l'ordre des couches pour convenir à une logique classique (point>ligne>polygone>basemap)
 self.coucheEnDernierIndex('observations_remarques')
 self.coucheEnDernierIndex('lineaire_curage')
 self.coucheEnDernierIndex('limites_as')
 self.coucheEnDernierIndex('basemap')
 done = "done"
 return done

def coucheEnDernierIndex(self,nom):
 # fonction pour ajouter une couche à la fin de l'arborescence de couche
 # nom(str) = nom de la couche à déplacer

 layer = QgsProject.instance().mapLayersByName(nom)[0]

 # Obtenez l'arborescence des couches de votre projet
 layer_tree_root = QgsProject.instance().layerTreeRoot()

 # Insérez la couche à la fin de l'arborescence
 layer_tree_root.insertChildNode(-1, QgsLayerTreeLayer(layer))

def snapping_configuration(self):
 #Mettre les options d'accrochage pour une bonne topologie des observations
 my_snap_config = QgsSnappingConfig()
 my_snap_config.setEnabled(True)
 my_snap_config.setType(QgsSnappingConfig.VertexAndSegment)
 my_snap_config.setUnits(QgsTolerance.ProjectUnits)
 my_snap_config.setTolerance(20)
 my_snap_config.setIntersectionSnapping(True)
 my_snap_config.setMode(QGIS.SnappingMode.AllLayers)

```

```

Appliquer la configuration de snapping au projet
QgsProject.instance().setSnappingConfig(my_snap_config)

def enregistreProjet (self): #RETOUR AU RAPPORT

 folder = self.recupPathSauvegarde()
 nomAS = self.CB_choixAs.currentText()
 file_path = os.path.join(folder, f"{nomAS}.qgz")
 project = QgsProject.instance()
 crs = QgsCoordinateReferenceSystem("EPSG:2154")
 project.setCrs(crs)
 project.write(file_path)

def zoomAS (self): #RETOUR AU RAPPORT
 canvas = self iface.mapCanvas()
 layer = QgsProject.instance().mapLayersByName("limites_as")[0] # Récupéré la couche polygonal la plus grande
 du projet

 extent = layer.extent() # Get the extent of the layer
 canvas.setExtent(extent) # Zoom sur l'emprise
 canvas.refresh() # Refresh the canvas to apply the changes

def rename_champ(self, nomCouche, ancienNom, nouveauNom):
 # Fonction pour renommer un champs avec nomCouche(str) = nom de la couche, ancienNom(str) = nom du champ à
 remplacer, nouveauNom(str) = nouveau nom du champ
 layer = QgsProject.instance().mapLayersByName(nomCouche)[0]
 index = layer.dataProvider().fieldNameIndex(ancienNom)
 if index != -1:
 layer.dataProvider().renameAttributes({index: nouveauNom})
 layer.updateFields()

def drop_layer(self,nom):
 # Fonction pour enlever une couche grâce à son nom
 if len(QgsProject.instance().mapLayersByName(nom)) > 0:
 layer = QgsProject.instance().mapLayersByName(nom)[0]
 QgsProject.instance().removeMapLayer(layer)

def verif(self) -> (bool): #RETOUR AU RAPPORT
 #fonction de vérification que les champs soient bien renseigné.

 path_sauvegarde = self.LE_choixPath.text()
 if self.CB_choixAs.currentText() == "":
 message = "Merci de sélectionner une AS"
 self.warning(message)
 return False
 if path_sauvegarde == "":
 message = "Merci de choisir un dossier de sauvegarde pour vos couches."
 self.warning(message)
 return False
 if len(os.listdir(path_sauvegarde)) > 0:

```

```

message = "Merci de choisir un nouveau dossier de sauvegarde vide pour vos couches.
 Des fichiers ont été
identifiés dans le dossier choisi"
 self.warning(message)
 return False

return True

def accept(self):
 if self.verif():
 self.exportVecto()
 self.ajoutCoucheExporter()
 self.ajoutBaseMap()
 self.styleCoucheObservation()
 self.styleCoucheLineaire()
 self.styleLimitesAS()
 done = self.ordreCouches()
 self.snaping_configuration()
 QTimer.singleShot(1000, lambda: self.zoomAS())
 self.enregistreProjet()
 if done == 'done':
 QDialog.accept(self)

def reject(self):
 QDialog.reject(self)

def warning(self, message):
 QMessageBox.warning(self, "Avertissement", message)

```

## etape2\_dialog.py programme complet

```
-*- coding: utf-8 -*-
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5 import uic

from QGIS.core import *
from QGIS.gui import *
from QFieldsync.gui.cloud_projects_dialog import CloudProjectsDialog
from QFieldsync.core.cloud_api import CloudNetworkAccessManager
from db_manager.db_plugins import createDbPlugin
from urllib.parse import urlparse

import sys, os, processing, time

ui_path = os.path.dirname(os.path.abspath(__file__))
ui_path = os.path.join(ui_path, "forms")
form_action1, _ = uic.loadUiType(os.path.join(ui_path, "etape_2.ui"))

class Etape2_dialog(QDialog, form_action1):
 def __init__(self, iface):
 QWidget.__init__(self)
 self.setupUi(self)
 self.nomBD = "unima"
 self.schema_reseau = "reseau"
 self.schema_limitesAS = 'administratif'
 self.schema_regieTravaux = "regie_travaux"

 self.iface = iface
 self.network_manager = CloudNetworkAccessManager(self.iface mainWindow())

 self.PB_suivant.setEnabled(False)

 self.nom_projet = QgsProject.instance().baseName()
 self.label_projetNom.setText(self.nom_projet)
 if self.nom_projet.endswith('_cloud'):
 self.label_projetInfo.setText('<html><head/><body><p>Merci de ne pas
 choisir un projet cloud (terminant par "_cloud")</p></body></html>')
 self.PB_suivant.setEnabled(False)
 self.progressBar.setRange(0, 0)
 self.PB_CloudConn.setVisible(False)

 self.PB_CloudConn.clicked.connect(self.open_cloud)
 self.CkB_selecReseau.stateChanged.connect(self.montrer_enti_select_troncon)
 self.CkB_selecObservation.stateChanged.connect(self.montrer_enti_select_observation)
 self.PB_suivant.clicked.connect(self.PB_suivant_clicked)
 self.PB_retour.clicked.connect(self.PB_retour_clicked)
 #self.CkB_selecTranche.stateChanged.connect(self.CkB_selecTrancheChanged)
 self.CkB_selecAnnee.stateChanged.connect(self.CkB_selecAnneeChanged)
```

```

self.CkB_selecType.stateChanged.connect(self.CkB_selecTypeChanged)
self.PB_creer.clicked.connect(self.accept)
self.PB_annuler.clicked.connect(self.reject)

self.remplirComboboxReseau()
self.remplirComboboxObservation()
self.remplirComboboxBaseMap()
self.remplirComboboxAs()
self.information('Pensez à bien sauvegarder vos couches, sauvegarder le projet puis le rafraîchir (F5) !')
self.suivantEnabled()

def remplirComboboxReseau(self):
 # Effacer tous les éléments existants de la combobox
 self.CB_choixReseau.clear()

 # Obtenir une liste de toutes les couches dans le projet
 layers = QgsProject.instance().mapLayers().values()

 # Filtrer les couches pour ne garder que celles qui sont des lignes ou des multilignes (geometryType() == 1)
 line_layers = [layer for layer in layers if layer.type() == QgsMapLayerType.VectorLayer and layer.geometryType() == 1]

 # Ajouter les noms des couches filtrées à la QComboBox
 for layer in line_layers:
 self.CB_choixReseau.addItem(layer.name())

def montrer_enti_select_troncon(self):

 self.label_troncon_select.setText("")
 if self.CkB_selecReseau.isChecked():
 layer = self.lineaire_selectionne()
 nb_enti = layer.featureCount()
 self.label_troncon_select.setText(f'{nb_enti}')

def remplirComboboxObservation(self):
 # Effacer tous les éléments existants de la combobox
 self.CB_choixObservation.clear()

 # Obtenir une liste de toutes les couches dans le projet
 layers = QgsProject.instance().mapLayers().values()

 # Filtrer les couches pour ne garder que celles qui sont des points ou des multipoints (geometryType() == 0)
 line_layers = [layer for layer in layers if layer.type() == QgsMapLayerType.VectorLayer and layer.geometryType() == 0]

 # Ajouter les noms des couches filtrées à la QComboBox
 for layer in line_layers:
 self.CB_choixObservation.addItem(layer.name())

def montrer_enti_select_observation(self):

 self.label_observation_select.setText("")
 if self.CkB_selecObservation.isChecked():

```

```

layer = self.observation_selectionne()
nb_enti = layer.featureCount()
self.label_observation_select.setText(f'{nb_enti}')

def remplirComboboxAs(self):
 # Effacer tous les éléments existants de la combobox
 self.CB_choixAs.clear()

 # Obtenir une liste de toutes les couches dans le projet
 layers = QgsProject.instance().mapLayers().values()

 # Filtrer les couches pour ne garder que celles qui sont des polygones ou des multipolygones (geometryType() == 2)
 line_layers = [layer for layer in layers if layer.type() == QgsMapLayerType.VectorLayer and layer.geometryType() == 2]

 # Ajouter les noms des couches filtrées à la QComboBox
 for layer in line_layers:
 self.CB_choixAs.addItem(layer.name())

def remplirComboboxBaseMap(self):
 # Effacer tous les éléments existants de la combobox
 self.CB_choixBasemap.clear()

 # Obtenir une liste de toutes les couches dans le projet
 layers = QgsProject.instance().mapLayers().values()

 # Filtrer les couches pour ne garder que celles qui sont des rasters
 line_layers = [layer for layer in layers if layer.type() != QgsMapLayerType.VectorLayer]

 # Ajouter les noms des couches filtrées à la QComboBox
 for layer in line_layers:
 self.CB_choixBasemap.addItem(layer.name())

def lineaire_selectionne(self): #RETOUR AU RAPPORT
 # Renvoie la couche correspondante aux tronçons (complète ou seulement les entités selectionnées)
 nom_layer = self.CB_choixReseau.currentText()
 layer = QgsProject.instance().mapLayersByName(nom_layer)[0]

 if not self.CkB_selecReseau.isChecked():
 return layer

 else:
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterFids(layer.selectedFeatureIds()))
 return layer_selection

def lineaire_selectionne_clip(self):
 # Renvoie la couche correspondante aux tronçons pour l'algorithme de découpe du clip raster
 nom_layer = self.CB_choixReseau.currentText()
 layer = QgsProject.instance().mapLayersByName(nom_layer)[0]

 return layer

```

```

def observation_selectionne(self):
Renvoie la couche correspondante aux observations (complète ou seulement les entités selectionnées)
 nom_layer = self.CB_choixObservation.currentText()
 layer = QgsProject.instance().mapLayersByName(nom_layer)[0]

 if not self.CkB_selecObservation.isChecked():
 return layer

 else:
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterFids(layer.selectedFeatureIds()))

 return layer_selection

def limites_as_selectionne(self):
Renvoie la couche correspondante aux limites_as et l'export dans le fichier temp
 nom_layer = self.CB_choixAs.currentText()
 layer = QgsProject.instance().mapLayersByName(nom_layer)[0]

 return layer

def basemap_selectionne(self):
Renvoie la couche correspondante à la basemap
 nom_layer = self.CB_choixBasemap.currentText()
 layer = QgsProject.instance().mapLayersByName(nom_layer)[0]
 return layer

def CkB_selecTrancheChanged(self):
Appel le remplissage de la comboBox en si la checkbox est cochée
 self.CB_choixTranche.clear()
 if self.CkB_selecTranche.isChecked():
 self.remplirComboboxChoixTranche()

def CkB_selecAnneeChanged(self):
Appel le remplissage de la comboBox en si la checkbox est cochée
 self.CB_choixAnnee.clear()
 if self.CkB_selecAnnee.isChecked():
 self.remplirComboboxChoixAnnee()

def CkB_selecTypeChanged(self):
Appel le remplissage de la comboBox en si la checkbox est cochée
 self.CB_choixType.clear()
 if self.CkB_selecType.isChecked():
 self.remplirComboboxChoixType()

def remplirComboboxChoixTranche(self): #RETOUR AU RAPPORT
Rempli la ComboBox CB_choixTranche avec les tranches renseignées dans 'lineaire_curage'
 self.CB_choixTranche.clear()
 layer = self.lineaire_selectionne()
 idx = layer.fields().indexOf('hit_tranche')
 values = layer.uniqueValues(idx)
 values = [v for v in values if v != 'NULL']
 strvalues = [str(value) for value in values]

```

```

for i in strvalues:
 self.CB_choixTranche.addItem(i)

def remplirComboboxChoixAnnee(self):
 # Rempli la ComboBox CB_choixAnnee avec les années renseignées dans 'lineaire_curage' si l'option est activée
 layer = self.lineaire_selectionne()
 idx = layer.fields().indexOf('hit_annee')
 values = layer.uniqueValues(idx)
 strvalues = [str(value) for value in values]
 for i in strvalues:
 self.CB_choixAnnee.addItem(i)

def remplirComboboxChoixType(self):
 # Rempli la ComboBox CB_choixType avec les types de tronçons renseignés dans 'lineaire_curage' si l'option est
 # activée
 layer = self.lineaire_selectionne()
 idx = layer.fields().indexOf('res_type')
 values = layer.uniqueValues(idx)
 strvalues = [str(value) for value in values]
 for i in strvalues:
 self.CB_choixType.addItem(i[0]) # On ne prends que le premier caractère car le type est en décimal

def lineaire_filtre(self):
 # Renvoie la couche lineaire selon les filtres appliqués par l'utilisateur
 project = QgsProject.instance()
 layer = self.lineaire_selectionne()
 expression = ""
 trancheSelect = self.CB_choixTranche.currentText()
 expression += f"hit_tranche" = {trancheSelect}
 if self.CkB_selecAnnee.isChecked():
 anneeSelect = self.CB_choixAnnee.currentText()
 if expression == "":
 expression += f"hit_annee" = {anneeSelect}"
 else:
 expression += f" AND "hit_annee" = {anneeSelect}"
 if self.CkB_selecType.isChecked():
 typeSelect = self.CB_choixType.currentText()
 if expression == "":
 expression += f"res_type" = {typeSelect}"
 else:
 expression += f" AND "res_type" = {typeSelect}"

 # Effacer la sélection précédente
 layer.removeSelection()

 # Créer une nouvelle couche avec les entités sélectionnées
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterExpression(expression))

 if expression == "":
 project.addMapLayer(layer)
 return layer
 else:
 project.addMapLayer(layer_selection)

```

```

 return layer_selection

def clipRaster(self): #RETOUR AU RAPPORT
 # Fonction permettant de découper l'orhto sur les limites de l'AS avec un buffer de 500m (pour une meilleure lisibilité et
 # une place diminué sur le cloud) et de l'exporter.
 project = QgsProject.instance()
 mask = self.limites_as_selectionne()
 input = self.basemap_selectionne()
 layer = self.lineaire_selectionne_clip()

 expression = ""
 trancheSelect = self.CB_choixTranche.currentText()
 expression += f"hit_tranche" = {trancheSelect}

 if self.CkB_selecAnnee.isChecked():
 anneeSelect = self.CB_choixAnnee.currentText()
 if expression == "":
 expression += f"hit_annee" = {anneeSelect}"
 else:
 expression += f" AND "hit_annee" = {anneeSelect}"
 if self.CkB_selecType.isChecked():
 typeSelect = self.CB_choixType.currentText()
 if expression == "":
 expression += f"res_type" = {typeSelect}"
 else:
 expression += f" AND "res_type" = {typeSelect}"

 layer.selectByExpression(expression)
 EmpriseMini = processing.run("QGIS:minimumboundinggeometry", {
 'INPUT': QgsProcessingFeatureSourceDefinition(layer.id(), True), #TRUE = que les entités sélectionnées (laissé
 vide pour FALSE)
 'TYPE': 3,
 'OUTPUT': 'memory:'})

 coucheEmpriseMini = QgsProject.instance().addMapLayer(EmpriseMini['OUTPUT'])

 layer.removeSelection()

 # Paramétrage du tampon
 buffer_params = {
 "INPUT": coucheEmpriseMini,
 "DISTANCE": 500,
 "OUTPUT": "memory:"
 }
 # Exécuter l'algorithme de tampon
 buffer_result = processing.run("native:buffer", buffer_params)
 self.buffer_layer_temp = buffer_result["OUTPUT"]

 # Paramétrage du clip raster par masque
 parameters = {
 'ALPHA_BAND': False,
 'CROP_TO_CUTLINE': True,

```

```

'DATA_TYPE': 0,
'INPUT': input,
'KEEP_RESOLUTION': True,
'MASK': self.buffer_layer_temp,
'MULTITHREADING': False,
'OUTPUT': 'memory',
'SOURCE_CRS': mask.crs(),
'TARGET_CRS': mask.crs(),
'NO_DATA': -9999,
'OUTPUT' : 'TEMPORARY_OUTPUT',
'OPTIONS': 'COMPRESS=JPEG',
}

clip_output = processing.run("gdal:cliprasterbymasklayer", parameters)

result = QgsRasterLayer(clip_output['OUTPUT'], "clipped", 'gdal')
project.addMapLayer(result, False)
layer_tree_root = QgsProject.instance().layerTreeRoot()

Insérez la couche à la fin de l'arborescence
layer_tree_root.insertChildNode(-1, QgsLayerTreeLayer(result))

Export
temp_folder = self.temp_folder
filename = "basemap_clipped.tif"
filepath = os.path.join(temp_folder, filename)

Créer un objet QgsRasterFileWriter
raster_writer = QgsRasterFileWriter(filepath)
opts = ["COMPRESS=JPEG", "JPEG_QUALITY=75"]
raster_writer.setCreateOptions(opts)
Utiliser l'objet pour écrire le raster
error = raster_writer.writeRaster(result.pipe(), result.width(), result.height(), result.extent(), result.crs())

Vérifier s'il y a une erreur
if error == QgsRasterFileWriter.NoError:
 message = "Exportation réussie"
 print(message)
 return filepath
else:
 # Récupérer le message d'erreur
 message = "Erreur lors de l'exportation Raster"
 print(message)

def exportLineaireFiltre(self): #RETOUR AU RAPPORT
 crs = QgsCoordinateReferenceSystem(2154)
 layer_temp_export = self.lineaire_filtre()

 expression = ""
 trancheSelect = self.CB_choixTranche.currentText()
 expression += f'tranche_{trancheSelect}'


```

```

if self.CkB_selecAnnee.isChecked():
 anneeSelect = self.CB_choixAnnee.currentText()
 if expression == "":
 expression += f'{anneeSelect}'
 else:
 expression += f'_{{anneeSelect}}'
if self.CkB_selecType.isChecked():
 typeSelect = self.CB_choixType.currentText()
 if expression == "":
 expression += f'type_{typeSelect}'
 else:
 expression += f'_type_{typeSelect}'

self.expression = expression
filename = f"lineaire_curage_{expression}.gpkg"
Nouveau dossier où enregistrer le projet
chemin_projet = QgsProject.instance().fileName()
repertoire_projet = os.path.dirname(chemin_projet)
self.temp_folder = os.path.join(repertoire_projet, f'{expression}')
temp_folder = self.temp_folder

Vérifier si le dossier temporaire n'existe pas déjà
if not os.path.exists(temp_folder):
 os.makedirs(temp_folder)
else:
 print(f"Le dossier temporaire '{temp_folder}' existe déjà.")
filepath = os.path.join(temp_folder, filename)

error, message = QgsVectorFileWriter.writeAsVectorFormat(layer_temp_export, filepath, 'UTF-8', crs,
driverName='GPKG')
if error == QgsVectorFileWriter.NoError:
 message += "Exportation Réussi de lineaire curage"
 print(message)
 return filepath
else:
 message += "Erreur lors de l'exportation de lineaire curage"
 print(message)

def exportLimites_as(self):

 layer = self.limites_as_selectionne()
 temp_folder = self.temp_folder

 filename = "limite_as.gpkg"
 filepath = os.path.join(temp_folder, filename)
 crs = QgsCoordinateReferenceSystem(2154)

 error, message = QgsVectorFileWriter.writeAsVectorFormat(layer, filepath, 'UTF-8', crs, driverName='GPKG')
 if error == QgsVectorFileWriter.NoError:
 message += "Exportation Réussi de limites_as"
 print(message)
 else:

```

```

message += "Erreur lors de l'exportation de limites_as"
print(message)

return filepath

def exportObservations(self):
 layer = self.observation_selectionne()
 couche_cible = self.buffer_layer_temp
 # Définir les paramètres de l'algorithme
 params = {
 'INPUT': layer,
 'PREDICATE': [0], # Utilisez 0 pour "intersecte"
 'INTERSECT': couche_cible,
 'OUTPUT': 'memory'
 # Vous pouvez spécifier un chemin de fichier si vous voulez enregistrer le résultat dans un fichier
 }

 # Exécuter l'algorithme
 resultat = processing.run("native:extractbylocation", params)

 # Accéder à la couche résultante
 couche_resultat = resultat['OUTPUT']

 temp_folder = self.temp_folder

 filename = "observation_remarque.gpkg"
 filepath = os.path.join(temp_folder, filename)
 crs = QgsCoordinateReferenceSystem(2154)

 error, message = QgsVectorFileWriter.writeAsVectorFormat(couche_resultat, filepath, 'UTF-8', crs, driverName='GPKG')
 if error == QgsVectorFileWriter.NoError:
 message += "Exportation Réussi de observations_remarques"
 print(message)
 else:
 message += "Erreur lors de l'exportation de observations_remarques"
 print(message)

 return filepath

def exportVersFolderTemp(self):
 # Nouveau dossier où enregistrer le projet

 # Récupérer le nom du champ 'nom' de la couche limites_as
 layer = self.limites_as_selectionne()
 nom_champ = 'nom'

 # Obtenir l'index du champ à partir de son nom
 index_champ = layer.fields().lookupField(nom_champ)

 # Vérifier si le champ existe dans la couche
 if index_champ != -1:
 # Boucler à travers les fonctionnalités de la couche
 for feature in layer.getFeatures():

```

```

Récupérer la valeur du champ pour chaque fonctionnalité
nom_champ_limites_as = feature.attribute(index_champ)
else:
 print(f"Le champ '{nom_champ_limites_as}' n'existe pas dans la couche.")

Récupérer le chemin absolu du fichier linéaire filtré
chemin_fichier_filtre = self.exportLineaireFiltre()

Récupérer le path du dossier de sauvegarde du projet
temp_folder = self.temp_folder
print(f'temp folder : {temp_folder}')
Extraire le nom du fichier sans le chemin et l'extension
nom_fichier_filtre = os.path.splitext(os.path.basename(chemin_fichier_filtre))[0]
nom_fichier_filtre = self.expression

Créer un nouveau nom de fichier en concaténant le champ 'nom' avec le nom du fichier filtré
nouveau_nom_fichier = f'{nom_champ_limites_as}_{nom_fichier_filtre}.qgz'

Chemin complet du nouveau projet
self.nouveau_projet_filepath = os.path.join(temp_folder, nouveau_nom_fichier)

self.filepathRaster = self.clipRaster()
print(f'chemin couche raster {self.filepathRaster}')
self.filepathLimites = self.exportLimites_as()
print(f'chemin couche limites_as {self.filepathLimites}')
self.filpathLineaire = self.exportLineaireFiltre()
print(f'chemin couche lineaire_curage {self.filpathLineaire}')
self.filepathObservation = self.exportObservations()
print(f'chemin couche observations_remarques {self.filepathObservation}')

self.supprimerCouchesTemporaires()
print('enregistrement du projet de base')
QgsProject().instance().write()

def creerProjetTemp(self): #RETOUR AU RAPPORT

Créer un nouveau projet QGIS
nouveau_projet = QgsProject().instance()
nouveau_projet.clear()
nouveau_projet.removeAllMapLayers()

nouveau_projet_filepath = self.nouveau_projet_filepath
Enregistrer le nouveau projet dans le nouveau dossier
nouveau_projet.write(nouveau_projet_filepath)
Définir le système de coordonnées de référence (CRS) du nouveau projet sur EPSG:2154
nouveau_projet.setCrs(QgsCoordinateReferenceSystem(2154))
importer couche clipRaster dans le nouveau projet

folderClipRaster = self.filepathRaster
coucheClipRaster = QgsRasterLayer(folderClipRaster, 'basemap', "gdal")
Vérifier si la couche est valide
if coucheClipRaster.isValid():
 # Ajouter la couche au nouveau projet

```

```

nouveau_projet.addMapLayer(coucheClipRaster)
else:
 print("La couche clipRaster n'est pas valide.")

importer couche limites_as dans le nouveau projet

folderLimites = self.filepathLimites
coucheLimites = QgsVectorLayer(folderLimites, 'limites_as', 'ogr')

Vérifier si la couche est valide
if coucheLimites.isValid():
 # Ajouter la couche au nouveau projet
 nouveau_projet.addMapLayer(coucheLimites)
 # Chargement du style
 style_path = os.path.dirname(os.path.abspath(__file__))
 style_path = os.path.join(style_path, "style")
 style_path = os.path.join(style_path, "style_limites_as.qml")
 # Vérification si le fichier de style existe
 if os.path.exists(style_path):
 # Charger le style uniquement si le fichier existe
 coucheLimites.loadNamedStyle(style_path)
 coucheLimites.saveStyleToDatabase(name='style_limites_as', description='Style par défaut de la couche
limites_as', useAsDefault=True, uiFileContent="")
 else:
 print("Le fichier de style n'existe pas :", style_path)

else:
 print("La couche lineaire_curage n'est pas valide.")

folderLineaire = self.filpathLineaire
coucheLineaire = QgsVectorLayer(folderLineaire, 'lineaire_curage', 'ogr')

Vérifier si la couche est valide
if coucheLineaire.isValid():
 with edit(coucheLineaire):
 for feature in coucheLineaire.getFeatures():
 feature["hit_num_tranche_be"] = self.LE_numUnima.text()
 coucheLineaire.updateFeature(feature)
 # Ajouter la couche au nouveau projet
 nouveau_projet.addMapLayer(coucheLineaire)
 # Chargement du style
 style_path = os.path.dirname(os.path.abspath(__file__))
 style_path = os.path.join(style_path, "style")
 style_path = os.path.join(style_path, "style_lineaire_regie.qml")
 # Vérification si le fichier de style existe
 if os.path.exists(style_path):
 # Charger le style uniquement si le fichier existe
 coucheLineaire.loadNamedStyle(style_path)
 coucheLineaire.saveStyleToDatabase(name='style_lineaire_regie', description='Style par défaut de la couche
lineaire_curage utilisé par la Régie', useAsDefault=True, uiFileContent="")
 else:

```

```

 print("Le fichier de style n'existe pas :", style_path)
else:
 print("La couche lineaire_curage n'est pas valide.")

importer couche observations_remarques dans le nouveau projet

folderObservation = self.filepathObservation
coucheObservation = QgsVectorLayer(folderObservation, 'observations_remarques', 'ogr')

Vérifier si la couche est valide
if coucheObservation.isValid():
 # Ajouter la couche au nouveau projet
 nouveau_projet.addMapLayer(coucheObservation)
 # Chargement du style
 style_path = os.path.dirname(os.path.abspath(__file__))
 style_path = os.path.join(style_path, "style")
 style_path = os.path.join(style_path, "style_ponctuel.qml")
 # Vérification si le fichier de style existe
 if os.path.exists(style_path):
 # Charger le style uniquement si le fichier existe
 coucheObservation.loadNamedStyle(style_path)
 else:
 print("Le fichier de style n'existe pas :", style_path)
else:
 print("La couche lineaire_curage n'est pas valide.")

print(f'filepath du nouveau projet : {nouveau_projet_filepath}')
Enregistrer les modifications du projet
nouveau_projet.write(nouveau_projet_filepath)
Effacer le projet actuel avant de lire le nouveau projet
QgsProject.instance().clear()
Attente pour s'assurer que le fichier est complètement écrit
print("dodo de 10 sec")
time.sleep(10)
print("ouverture du nouveau projet")
nouveau_projet.read(nouveau_projet_filepath)

masquer couche limites_as
layer = QgsProject.instance().mapLayersByName("limites_as")[0]
layer_id = layer.id()

QgsProject.instance().layerTreeRoot().findLayer(layer_id).setItemVisibilityChecked(False) # Masquer

Enregistrer les modifications du projet
nouveau_projet.write(nouveau_projet_filepath)

self.snappling_configuration()

Importation de la table 'observation_type' pour récupérer les valeurs stockés dans cette table
print('value map en cours de mise à jour')

```

```

self.drop_layer("observation_type")
try:
 # Déballage des informations de connexion
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 schema = self.schema_regieTravaux

 # Création de l'URI de la source de données
 uri = QgsDataSourceUri()
 uri.setConnection(whost, str(wport), wdatabase, wusername, wpassword)
 uri.setDataSource(schema, 'observation_type', '')

 la_couche_resultat = QgsVectorLayer(uri.uri(), "observation_type", "postgres")

 if la_couche_resultat.isValid():
 QgsProject.instance().addMapLayer(la_couche_resultat)
 print('Couche résultat ajoutée avec succès')
 else:
 print('Erreur: La couche résultat n\'est pas valide')
except Exception as e:
 message = "Erreur : %s" % str(e)
 self.warning(message)

Récupération des donnée depuis la tables + formatage pour la ValueMap
layer = QgsProject.instance().mapLayersByName('observation_type')[0]

Initialisez une liste pour stocker les dictionnaires
liste_dictionnaires = []

Parcourez les entités de la couche
for feature in layer.getFeatures():
 # Récupérez les valeurs des champs 'obs_id' et 'obs_nom' pour chaque entité
 obs_id = feature['obs_id']
 obs_nom = feature['obs_nom']

 # Créez un dictionnaire avec 'obs_id' comme clé et 'obs_nom' comme valeur
 dictionnaire = {obs_nom:obs_id}

 # Ajoutez ce dictionnaire à la liste
 liste_dictionnaires.append(dictionnaire)

layerpoint = QgsProject.instance().mapLayersByName('observations_remarques')[0]

Récupéré l'index du champ 'obs_id'
idx = layerpoint.fields().lookupField('obs_id')

Définir la ValueMap
v_map = {'map': liste_dictionnaires}

Définir la configuration du widget de l'éditeur sur la couche, en passant par l'index du champ et la configuration du
widget.
layerpoint.setEditorWidgetSetup(idx, QgsEditorWidgetSetup('ValueMap', v_map))

```

```

Suppression de la table du projet
self.drop_layer("observation_type")

Ajout de la liste des pelleteurs à une valueMap

if len(nouveau_projet.mapLayersByName("pelleteur")) > 0:
 layer = nouveau_projet.mapLayersByName("pelleteur")[0]
 nouveau_projet.removeMapLayer(layer)

try:
 # Déballage des informations de connexion
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 schema = self.schema_regieTravaux

 # Création de l'URI de la source de données
 uri = QgsDataSourceUri()
 uri.setConnection(whost, str(wport), wdatabase, wusername, wpassword)
 uri.setDataSource(schema, 'pelleteur', '')

 la_couche_resultat = QgsVectorLayer(uri.uri(), "pelleteur", "postgres")

 if la_couche_resultat.isValid():
 nouveau_projet.addMapLayer(la_couche_resultat)
 print('Couche pelleteur ajoutée avec succès')
 else:
 print('Erreur: La couche résultat n\'est pas valide')

Récupération des donnée depuis la tables + formatage pour la ValueMap
layer = nouveau_projet.mapLayersByName('pelleteur')[0]

Initialisez une liste pour stocker les dictionnaires
liste_dictionnaires = []

Parcourez les entités de la couche
for feature in layer.getFeatures():
 # Récupérez les valeurs des champs 'obs_id' et 'obs_nom' pour chaque entité
 pel_id = feature['pel_id']
 pel_nom = feature['pel_nom']

 # Créez un dictionnaire avec 'obs_id' comme clé et 'obs_nom' comme valeur
 dictionnaire = {pel_nom:pel_id}

 # Ajoutez ce dictionnaire à la liste
 liste_dictionnaires.append(dictionnaire)

coucheLineaire = nouveau_projet.mapLayersByName("lineaire_curage")[0]
Récupérer l'index du champ 'pel_id'
idx = coucheLineaire.fields().lookupField('pel_id')
Définir la ValueMap
v_map = {'map': liste_dictionnaires}

Définir la configuration du widget de l'éditeur sur la couche, en passant par l'index du champ et la configuration du
widget.
coucheLineaire.setEditorWidgetSetup(idx, QgsEditorWidgetSetup('ValueMap', v_map))

```

```

Suppression de la table du projet

if len(nouveau_projet.mapLayersByName("pelleteur")) > 0:
 layer = nouveau_projet.mapLayersByName("pelleteur")[0]
 nouveau_projet.removeMapLayer(layer)
except:
 print('erreur dans la mise à jour de la value map des pelleteurs')

print('value map mis à jour')

nouveau_projet.write()
print('enregistré')

def ajoutNumUnimaTranche(self):
 layer = self.lineaire_filtre()
 with edit(layer):
 for feature in layer.getFeatures():
 feature["hit_num_tranche_be"] = self.LE_numUnima.text()
 layer.updateFeature(feature)

def snapping_configuration(self):
 #Mettre les options d'accrochage pour une bonne topologie des observations
 my_snap_config = QgsSnappingConfig()
 my_snap_config.setEnabled(True)
 my_snap_config.setType(QgsSnappingConfig.VertexAndSegment)
 my_snap_config.setUnits(QgsTolerance.ProjectUnits)
 my_snap_config.setTolerance(20)
 my_snap_config.setIntersectionSnapping(True)
 my_snap_config.setMode(QGIS.SnappingMode.AllLayers)

 # Appliquer la configuration de snapping au projet
 QgsProject.instance().setSnappingConfig(my_snap_config)

def get_info_postgis(self, wnom):
 # renvoie l'hôte, le port, le nom de la DB, le user et le pwd sous forme de tuple
 resu = None
 dbpluginclass = createDbPlugin("postgis")

 for c in dbpluginclass.connections():
 if c.connectionName() == wnom:
 settings = QgsSettings()
 settings.beginGroup(u"/%s/%s" % ('PostgreSQL/connections', wnom))

 if settings.contains("database"):
 whost = settings.value("host")
 wport = settings.value("port")
 wdatabase = settings.value("database")
 wusername = settings.value("username")
 wpassword = settings.value("password")

```

```

 resu = (whost, wport, wdatabase, wusername, wpassword)
 return resu

def open_cloud(self):
 # Ouvre le dialog de connexion QFieldCloud
 dlg = CloudProjectsDialog(self.network_manager, self iface mainWindow())
 dlg.show()

def drop_layer(self,nom):
 # Fonction pour enlever une couche grâce à son nom
 if len(QgsProject.instance().mapLayersByName(nom)) > 0:
 layer = QgsProject.instance().mapLayersByName(nom)[0]
 QgsProject.instance().removeMapLayer(layer)

def supprimerCouchesTemporaires(self):
 # Obtenir l'instance actuelle du projet QGIS
 projet_actuel = QgsProject.instance()

 # Itérer sur toutes les couches du projet
 for layer in projet_actuel.mapLayers().values():
 # Vérifier si la couche est temporaire
 if layer.isTemporary():
 # Supprimer la couche temporaire
 projet_actuel.removeMapLayer(layer)

def suivantEnabled (self):
 if self.CB_choixReseau.currentText() != " or self.CB_choixObservation.currentText() != " or
 self.CB_choixAs.currentText() != " or self.CB_choixBasemap.currentText() != " :
 self.PB_suivant.setEnabled(True)

def PB_suivant_clicked(self):
 if self.verif1():

 self.stackWidget.setCurrentWidget(self.page2)
 self.remplirComboboxChoixTranche()

def PB_retour_clicked(self):

 self.stackWidget.setCurrentWidget(self.page1)

def verif1(self) -> (bool): #RETOUR AU RAPPORT

 if not self.verif1Lineaire():
 return False
 if not self.verif1Observation():
 return False
 if not self.verif1Limites():
 return False
 return True

```

```

def verif2(self) -> (bool):
 num_unima = self.LE_numUnima.text()
 if not num_unima.isdigit() or not (4 <= len(num_unima) <= 5):
 message = "Merci de saisir un numéro d'étude par tranche, contenant uniquement des chiffres et ayant une longueur entre 4 et 5 caractères."
 self.warning(message)
 return False
 return True

def verif1Lineaire(self) -> (bool):
 layer = self.lineaire_selectionne()

 # Liste des champs requis
 champs_requis =
 ["fid","res_code_tr","longueur","id_as","res_type","hit_annee","hit_tranche","hit_bro_cot","hit_aplo_cot",
 "hit_cot_vase","hit_temps_broyage","hit_temps_troncognage","hit_verif_depotvase","hit_verif_curage","pel_id","hit_hau
 teur_vase","hit_largeur_fosse"]
 # Liste des champs présents dans la couche
 layer_fields = []
 for field in layer.fields():
 layer_fields.append(field.name())

 # Initialisation de la liste des champs manquants
 champs_manquants = []

 # Boucle à travers les champs requis
 for field in champs_requis:
 # Vérifie si le champ actuel n'est pas présent dans la liste des champs de couche
 if field not in layer_fields:
 # Si le champ est manquant, l'ajoute à la liste des champs manquants
 champs_manquants.append(field)

 #fonction de vérification que les champs soient bien renseigné.
 if len(champs_manquants)>0:
 message = f"Il manque des champs obligatoire à votre couche 'lineaire_curage' :
 {champs_manquants} </br>

 Merci de renseigner une autre couche ou de ne pas changer/supprimer les champs obligatoires
"
 self.warning(message)
 return False
 else:
 return True

def verif1Observation(self) -> (bool):
 layer = self.observation_selectionne()

 # Liste des champs requis
 champs_requis = ["fid","hio_commentaire","hio_photo","res_code_tr","hio_date","as_n_unima"]
 # Liste des champs présents dans la couche
 layer_fields = []

```

```

for field in layer.fields():
 layer_fields.append(field.name())

Initialisation de la liste des champs manquants
champs_manquants = []

Boucle à travers les champs requis
for field in champs_requis:
 # Vérifie si le champ actuel n'est pas présent dans la liste des champs de couche
 if field not in layer_fields:
 # Si le champ est manquant, l'ajoute à la liste des champs manquants
 champs_manquants.append(field)

#fonction de vérification que les champs soient bien renseigné.
if len(champs_manquants)>0:
 message = f"Il manque des champs obligatoire à votre couche 'observations_remarques':
 {champs_manquants}"
 </br> Merci de rensigné une autre couche ou de ne pas changer/supprimer les champs obligatoires </br>"
 self.warning(message)
 return False
else:
 return True

def verifILimites(self) -> (bool):

 layer = self.limites_as_selectionne()

 # Liste des champs requis
 champs_requis = ["fid","id","n_unima","statuts","libell_","nom","annee","surface","adherent"]
 # Liste des champs présents dans la couche
 layer_fields = []
 for field in layer.fields():
 layer_fields.append(field.name())

 # Initialisation de la liste des champs manquants
 champs_manquants = []

 # Boucle à travers les champs requis
 for field in champs_requis:
 # Vérifie si le champ actuel n'est pas présent dans la liste des champs de couche
 if field not in layer_fields:
 # Si le champ est manquant, l'ajoute à la liste des champs manquants
 champs_manquants.append(field)

 #fonction de vérification que les champs soient bien renseigné.
 if len(champs_manquants)>0:
 message = f"Il manque des champs obligatoire à votre couche 'limites_as':
 {champs_manquants} </br>

Merci de rensigné une autre couche ou de ne pas changer/supprimer les champs obligatoires </br>"
 self.warning(message)
 return False
 else:
 return True

```

```

def accept(self):
 if self.verif2():
 print('je devrais afficher la page 3 ici')
 self.stackedWidget.setCurrentWidget(self.loadingPage)
 # Définir la barre de progression comme indéterminée

 QTimer.singleShot(50, self.run) # Ajout d'un delay pour que l'interface puisse se mettre à jour

def run (self):
 # Check if a project is currently loaded
 if QgsProject.instance().isDirty():
 # Save the current project if it has unsaved changes
 QgsProject.instance().write()

 self.ajoutNumUnimaTranche()
 self.exportVersFolderTemp()
 self.creerProjetTemp()

 #self.snaping_configuration()

 # Appeler la méthode on_create_button_clicked() sur un objet de la classe CloudProjectsDialog
 dlg = CloudProjectsDialog(self.network_manager, self iface.mainWindow())
 dlg.on_create_button_clicked()

 QDialog.accept(self)

def reject(self):
 QDialog.reject(self)

def warning(self, message):
 QMessageBox.warning(self, "Avertissement", message)

def information(self, message):
 QMessageBox.information(self, "Information", message)
 # self.iface.messageBar().pushMessage("Information", message, level=QGIS.Info)

```

### etape3\_dialog.py programme complet

```
-*- coding: utf-8 -*-
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5 import uic

from QGIS.core import *
from QGIS.gui import *
from QFieldsync.core.cloud_api import CloudNetworkAccessManager
from QFieldsync.gui.cloud_projects_dialog import CloudProjectsDialog
from QFieldsync.gui.cloud_login_dialog import CloudLoginDialog
from QFieldsync.core.cloud_project import CloudProject
from QFieldsync.gui.cloud_transfer_dialog import CloudTransferDialog
import sys, os

sys.path.append(os.path.dirname(os.path.abspath(__file__)) + "/forms")
from main_form import *
ui_path = os.path.dirname(os.path.abspath(__file__))
ui_path = os.path.join(ui_path, "forms")
form_action1, _ = uic.loadUiType(os.path.join(ui_path, "etape_3.ui"))

class Etape3_dialog(QDialog, form_action1):

 local_preference = QtCore.pyqtSignal()

 def __init__(self, iface):
 QWidget.__init__(self)
 self.setupUi(self)
 self.iface = iface
 self.network_manager = CloudNetworkAccessManager(self.iface mainWindow())

 """ Partie 1: Importer Projet BE """
 self.PB_ok1.setEnabled(False)
 self.cloud_login_dialog = CloudLoginDialog(self.network_manager)
 self.cloud_login_dialog.loginFinished.connect(self.login_signal_recu)
 self.PB_CloudConn.clicked.connect(self.open_cloudCon)
 self.PB_ok1.clicked.connect(self.accept1)
 self.PB_annuler1.clicked.connect(self.close)

 self.label_statut_connexion.setText('<html><head/><body><p>Merci de vous connecter au cloud </p></body></html>')

 self.project_transfer = None
 self.transfer_dialog = None

 """FIN Partie 1: Importer Projet BE """
 """ Partie 2: Ouvrir Projet QGIS DL """

 self.pb_definir_reperoire.clicked.connect(self.definir_reperoire)
 self.pb_lancer_scan.clicked.connect(self.lancer_scan)
```

```

self.lst_projet_qgs.itemSelectionChanged.connect(self.selection_liste_projet_change)
Obtenir le nom de l'utilisateur courant
nom_utilisateur = os.getlogin()
Construire le chemin du répertoire par défaut
repertoire_defaut = f"C:/Users/{nom_utilisateur}/QField/cloud"
self.repertoireChoisi = repertoire_defaut
self.lbl_repertoire.setText(repertoire_defaut)
self.leProjetCharge = ""

self.pb_lancer_scan.setEnabled(True)
self.pb_ouvrir_projet.setEnabled(False)
self.pb_ouvrir_projet.clicked.connect(self.ouvrir_projet)
self.PB_annuler2.clicked.connect(self.close)

""" FIN Partie 2: Ouvrir Projet QGIS DL """
""" Partie 3 : Exporter projet pour pelleteurs """
self.nom_projet = QgsProject.instance().baseName()
self.label_projetNom.setText(self.nom_projet)
if not self.nom_projet.endswith('_cloud'):
 self.label_projetInfo.setText('<html><head/><body><p>Merci de choisir
un projet cloud compatible</p></body></html>')
 self.PB_sync.setEnabled(False)
 self.label_statut_connexion_2.setText('<html><head/><body><p><span style=" font-size:8pt; font-weight:600;
font-style:italic;">Merci de vous connecter au cloud </p></body></html>')
 self.PB_CloudConn_2.clicked.connect(self.open_cloudCon)
 self.PB_sync.clicked.connect(self.accept3)
 self.PB_annuler3.clicked.connect(self.close)

""" FIN Partie 3 : Exporter projet pour pelleteurs """

self.RB_importBE.toggled.connect(self.afficherImporBEwidget)
self.RB_ouvrirprojet.toggled.connect(self.afficherScanWidget)
self.RB_exportPel.toggled.connect(self.afficherexportPelwidget)

@pyqtSlot()
def login_signal_recu(self):
 QTimer.singleShot(3000, self.peupleComboBoxProjets)
 QTimer.singleShot(3000, self.connection_status)

""" Partie 1: Importer Projet BE
Permet de récupérer un projet stocké sur le cloud parmi tous ceux existants.
Nécessite une connexion à QFieldCloud. """
def peupleComboBoxProjets(self):
 try:
 for cloud_project in self.network_manager.projects_cache.projects:
 self.CB_projetCloud.addItem(cloud_project.name)
 except:
 return

```

```

def recupererProjetCombo(self):
 nomProjetselec = self.CB_projetCloud.currentText()
 if self.network_manager.projects_cache.projects is not None:
 for cloud_project in self.network_manager.projects_cache.projects:
 if cloud_project.name == nomProjetselec:
 self.current_cloud_project = cloud_project

def open_cloudCon(self): #RETOUR AU RAPPORT

 # Ouvre le dialog de connexion QFieldCloud
 dlg = CloudLoginDialog(self.network_manager, self iface mainWindow())

 result = dlg.exec_()
 if result:
 pass

def connection_status(self):
 base_path = os.path.join(os.path.dirname(__file__), 'forms', 'icone')
 if self.network_manager.projects_cache.projects is None:
 self.label_statut_connexion.setText('Connexion échouée')
 self.label_statut_connexion_2.setText('Connexion échouée')
 self.label_statut_connexion_4.setText('Connexion échouée')
 self.PB_sync.setEnabled(False)
 icon_path = os.path.join(base_path, 'cloud-computing_failed.png')
 else:
 self.label_statut_connexion.setText('Connexion Réussie')
 self.label_statut_connexion_2.setText('Connexion Réussie')
 self.label_statut_connexion_4.setText('Connexion Réussie')
 self.PB_sync.setEnabled(True)
 self.PB_ok1.setEnabled(True)
 if not self.nom_projet.endswith('_cloud'):
 self.label_projetInfo.setText('<html><head/><body><p>Merci de
choisir un projet cloud compatible</p></body></html>')
 self.PB_sync.setEnabled(False)
 else:
 self.PB_sync.setEnabled(True)
 icon_path = os.path.join(base_path, 'cloud-computing_success.png')

 pixmap = QPixmap(icon_path)
 pixmap = pixmap.scaled(30, 30, Qt.KeepAspectRatio)
 self.label_statut_connexion_ico.setPixmap(pixmap)
 self.label_statut_connexion_ico_2.setPixmap(pixmap)
 self.label_statut_connexion_ico_4.setPixmap(pixmap)

def open_cloud(self):

 # Ouvre le dialog de connexion QFieldCloud
 dlg = CloudProjectsDialog(self.network_manager, self iface mainWindow())
 dlg.show()

```

```

""" FIN Partie 1: Importer Projet BE """
""" Partie 2: Ouvrir Projet QGIS DL """
def definir_reperatoire(self):
 repertoire = QFileDialog.getExistingDirectory(self, u"Sélectionner le répertoire à scanner", "e:/")
 if repertoire != "":
 self.lbl_reperatoire.setText(repertoire)
 self.pb_lancer_scan.setEnabled(True)
 self.reperatoireChoisi = repertoire
 else:
 if self.reperatoireChoisi == "":
 self.pb_lancer_scan.setEnabled(False)

def lancer_scan(self): #RETOUR AU RAPPORT
 listeProjetQgs = []
 self.lst_projet_qgs.clear()
 self.setCursor(Qt.WaitCursor)
 for root, dirs, files in os.walk(self.reperatoireChoisi):
 # Exclure les répertoires nommés .QFieldsync
 dirs[:] = [d for d in dirs if d != ".QFieldsync"]
 # L'utilisation de dirs[:] = [d for d in dirs if d != ".QFieldsync"]
 # Assure que les modifications sont faites directement sur la liste référencée par os.walk.
 # Cela garantit que les répertoires à exclure sont correctement filtrés durant le processus de parcours,
 # ce qui n'est pas assuré par une simple réaffectation de la variable dirs.
 # La syntaxe utilisé [:] s'appelle le slicing
 for file in files:
 if file.endswith("_cloud.qgs"):
 chemin_complet = os.path.join(root, file)
 # Ajouter uniquement le nom du projet à la liste
 nom_projet = os.path.splitext(os.path.basename(chemin_complet))[0]
 listeProjetQgs.append((nom_projet, chemin_complet))
 # Ajouter les projets à la liste avec uniquement leur nom affiché
 self.lst_projet_qgs.addItems([nom_projet for nom_projet, _ in listeProjetQgs])
 # Stocker la liste complète des chemins des projets pour une utilisation ultérieure
 self.liste_chemins_projets = listeProjetQgs
 self.setCursor(Qt.ArrowCursor)
 self.pb_ouvrir_projet.setEnabled(False)

def ouvrir_projet(self):
 if len(self.lst_projet_qgs.selectedItems()) != 0:
 # Récupérer le chemin complet du projet sélectionné
 nom_projet = self.lst_projet_qgs.selectedItems()[0].text()
 # "compréhension de liste" combinée avec la fonction next() pour récupérer le chemin complet d'un projet
 # à partir de self.liste_chemins_projets lorsque le nom du projet correspond à nom_projet.
 chemin_complet = next((chemin for nom, chemin in self.liste_chemins_projets if nom == nom_projet), None)
 if chemin_complet:
 if nom_projet == self.leProjetCharge:
 QMessageBox.warning(self, u"Ouverture de projet", u"Ce projet est déjà chargé !")
 else:
 leProjet = QgsProject.instance()
 leProjet.clear()
 leProjet.removeAllMapLayers()

 self.leProjetCharge = nom_projet

```

```

 leProjet.read(chemin_complet)

 self.organiseArbreCouche()

 self.close()

def selection_liste_projet_change(self):
 if len(self.lst_projet_qgs.selectedItems()) == 0:
 self.pb_ouvrir_projet.setEnabled(False)
 else:
 self.pb_ouvrir_projet.setEnabled(True)

def organiseArbreCouche(self):

 # Récupérer la session QgsProject
 la_session = QgsProject.instance()

 # Récupérer la racine de l'arbre des couches
 racine = la_session.layerTreeRoot()

 # Récupérer toutes les couches de la carte
 lesCouches = la_session.mapLayers()
 # Calcule le nombre de noeuds enfants du noeud racine de l'arbre des couches.
 nb_noeud = len(racine.children())

 # Créer des listes pour chaque type de géométrie
 couches_ponctuels = []
 couches_lignes = []
 couches_polygones = []
 couches_rasters = []

 # Parcourir chaque couche et la placer dans la liste correspondante
 for cle, valeur in lesCouches.items():
 if valeur.type() == QgsMapLayer.VectorLayer:
 if valeur.geometryType() == QgsWkbTypes.PointGeometry:
 couches_ponctuels.append((cle, valeur))
 elif valeur.geometryType() == QgsWkbTypes.LineGeometry:
 couches_lignes.append((cle, valeur))
 elif valeur.geometryType() == QgsWkbTypes.PolygonGeometry:
 couches_polygones.append((cle, valeur))
 elif valeur.type() == QgsMapLayer.RasterLayer:
 couches_rasters.append((cle, valeur))

 # Placer les couches dans l'ordre désiré à la racine de l'arbre des couches
 for couches in [couches_ponctuels, couches_lignes, couches_polygones, couches_rasters]:
 for cle, valeur in couches:
 # Cloner le noeud de la couche
 noeudCouche = racine.findLayer(cle)
 noeudCoucheClone = noeudCouche.clone()

 # Ajouter le noeud cloné à la racine de l'arbre des couches

```

```

racine.addChildNode(noeudCoucheClone)

Supprimer les anciens groupes
racine.removeChildren(0, nb_noeud)

def visible_seulement_dernier(self, noeudGroupe):
 nb_noeud = len(noeudGroupe.children())
 i = 0
 for enfant in noeudGroupe.children():
 i += 1
 if i == nb_noeud:
 # enfant.setVisible(Qt.Checked)
 enfant.setItemVisibilityChecked(True)
 else:
 # enfant.setVisible(Qt.Unchecked)
 enfant.setItemVisibilityChecked(False)

""" FIN Partie 2: Ouvrir Projet QGIS DL """
""" Partie 3 : Exporter projet pour pelleteurs """
def show_cloud_synchronize_dialog_local(self, firstTry=True): #RETOUR AU RAPPORT
 if self.network_manager.projects_cache.is_currently_open_project_cloud_local:
 self.transfer_dialog = CloudTransferDialog.show_transfer_dialog(
 self.network_manager,
 preference_action='local', # Passer 'local' à preference_action
 parent=self iface.mainWindow(),
)

""" FIN Partie 3 : Exporter projet pour pelleteurs """

def afficherImporBEwidget(self):
 self.stackedWidget.setCurrentWidget(self.page1)

def afficherScanWidget(self):
 self.stackedWidget.setCurrentWidget(self.page2)
 self.lancer_scan()

def afficherexportPelwidget(self):
 self.stackedWidget.setCurrentWidget(self.page4)
 self.lancer_scan()

def accept1(self):
 self.recupererProjetCombo() # appelez la méthode pour récupérer les projets dans le combo
 self.show_sync_popup() # Affichez une popup de synchronisation

def accept3(self):

```

```

 self.close()
 self.show_cloud_synchronize_dialog_local() # Affichez une popup de synchronisation

 #QDialog.accept(self)

def reject(self):
 QDialog.reject(self)

def show_sync_popup(self) -> None:
 assert self.current_cloud_project is not None, "No project to download selected"

 self.transfer_dialog = CloudTransferDialog.show_transfer_dialog(
 self.network_manager, self.current_cloud_project, None, None, self
)
 self.transfer_dialog.rejected.connect(self.on_transfer_dialog_rejected)
 self.transfer_dialog.accepted.connect(self.on_transfer_dialog_accepted)
 self.transfer_dialog.open()

def on_transfer_dialog_rejected(self) -> None:
 if self.project_transfer:
 self.project_transfer.abort_requests()

 if self.transfer_dialog:
 self.transfer_dialog.close()

 self.project_transfer = None
 self.transfer_dialog = None

def on_transfer_dialog_accepted(self) -> None:
 QgsProject().instance().reloadAllLayers()

 if self.transfer_dialog:
 self.transfer_dialog.close()
 self.close()

 self.project_transfer = None
 self.transfer_dialog = None

```

### etape4\_dialog.py programme complet

```
-*- coding: utf-8 -*-
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5 import uic
from db_manager.db_plugins import createDbPlugin
from QFieldsync.core.cloud_api import CloudNetworkAccessManager
from QFieldsync.gui.cloud_projects_dialog import CloudProjectsDialog
from QFieldsync.gui.cloud_login_dialog import CloudLoginDialog
from QFieldsync.core.cloud_project import CloudProject
from QFieldsync.gui.cloud_transfer_dialog import CloudTransferDialog

from QGIS.core import *
from QGIS.gui import *
import sys, os, psycopg2, shutil
from psycopg2 import sql

class RangeCalendarWidget(QWidget):
 def __init__(self, parent=None):
 super().__init__(parent)

 self.selection_mode = None # Mode de sélection initiale (utilisé dans la méthode de classe date_click)
 self.selected_dates = [] # Initialisation d'un dictionnaire pour contenir les dates sélectionnées
 self.date_autorise = set() # Ensemble des dates autorisées

 self.clicked.connect(self.date_click) # Connexion du signal de clic sur une date

 self.updateDateValide() # Mise à jour des dates valides

 def date_click(self, date) -> (None): # Méthode attribuant les valeurs des dates cliqué à start_date et end_date
 if date not in self.date_autorise: # Si la date cliquée n'est pas dans la liste des dates autorisées
 self.selection_mode = False # Mode de sélection réinitialisé
 return # On ne fait rien d'autre
 if not self.selection_mode: # Si le mode de sélection = None
 self.start_date = None # On remet à jour start_date
 self.end_date = None # On remet à jour end_date
 self.start_date = date # start_date prends la valeur de la date cliquée
 self.end_date = date # end_date prends la valeur de la date cliquée
 self.selection_mode = True # On passe le mode de sélection à True
 else:
 self.end_date = date # end_date prends la valeur de la date cliquée
 self.selection_mode = False # Mode de sélection réinitialisé
 if self.start_date > self.end_date:
 self.start_date, self.end_date = self.end_date, self.start_date # interversion des données des dates (afin de toujours avoir end_date plus grand que start_date)

 self.surlignePlageDate() # Lancement de la méthode de classe surlignePlageDate

 def surlignePlageDate(self) -> (None): # Méthode permettant de surligné la plage des données valide sélectionnée
 fmt = QTextCharFormat() # Initialisation de la classe QTextCharFormat
```

```

 fmt.setBackground(Qt.white) # Utilisation de la méthode setBackground pour mettre le fond des case en blanc
(couleur par défaut)

 # Effacer le formatage des dates précédemment sélectionnées
for d in self.date_autorise: # Pour chaque date présentes dans la liste des dates autorisées
 self.setDateTextFormat(d, fmt) # Utilisation de la méthode setDateTextFormat pour attribuer le format
précédemment initialisé (fond blanc)

self.selected_dates = self.plageDate() # Utilisation de la méthode permettant de connaitres les dates se situant
entre star_date et end_date

if self.start_date and self.end_date: # Si les deux variables ne sont pas None
 fmt.setBackground(Qt.yellow) # Couleur de mise en surbrillance en Jaune
 d = self.start_date # d prend la valeur actuel de start_date
 while d <= self.end_date: # Tant que d est inférieur à end_date
 if d in self.date_autorise: # Si d est dans la liste des dates autorisées
 self.setDateTextFormat(d, fmt) # On applique le fond jaune
 d = d.addDays(1) # On ajoute un jour a d pour continuer la boucle

 self.repaint() # Rafraîchir l'affichage du calendrier

def plageDate(self) -> (list): # Renvoie une liste de QDate
 dates = [] # Initialisation d'un dictionnaire contenant les dates à retourner
 if self.start_date and self.end_date: # Si les deux variables ne sont pas None
 d = self.start_date # d prend la valeur actuel de start_date
 while d <= self.end_date: # Tant que d est inférieur à end_date
 if d in self.date_autorise: # Si d est dans la liste des dates autorisées
 dates.append(d) # Ajout de d à la liste dates
 d = d.addDays(1) # On ajoute un jour a d pour continuer la boucle
 return dates

def plageDateString(self) -> (list): # Renvoie une liste de String au format PostgreSQL
 dates = []
 if self.start_date and self.end_date:
 d = self.start_date
 while d <= self.end_date:
 if d in self.date_autorise:
 dates.append(d.toString("yyyy-MM-dd")) # Mise au format PostgreSQL
 d = d.addDays(1)
 return dates

def setDateAutorise(self, dates) -> (None): # Methode pour mettre à jour les dates autorisées
 self.date_autorise = set(dates)
 self.updateDateValide() # Appel à la méthode de classe

def updateDateValide(self) -> (None): # Méthode permettant de mettre ne forme le calendrier en fonction des dates
autorisées
 today = QDate.currentDate() # Récupération de la date du jour
 start_date = QDate(today.year() - 10, 1, 1) # start_date équivaut au 1er Janvier n-10
 end_date = QDate(today.year() + 10, 12, 31) # end_date équivaut au 31 Décembre n+10

 disabled_fmt = QTextCharFormat() # Instanciation d'un objet de classe QTextCharFormat pour les dates non
autorisées

```

```

disabled_fmt.setForeground(QColor('gray')) # Police d'écriture en gris
disabled_fmt.setBackground(QColor('#d3d3d3')) # Background en gris claire pour mettre en avant les dates non
autorisées

enabled_fmt = QTextCharFormat() # Instanciation d'un objet de classe QTextCharFormat pour les dates
autorisées
enabled_fmt.setForeground(QColor('black')) # Police d'écriture en noir (par défaut)
enabled_fmt.setBackground(QColor('white')) # Fond en blanc (par défaut)

d = start_date
while d <= end_date:
 if d not in self.date_autorise:
 self.setDateTextFormat(d, disabled_fmt) # Application du format Non autorisé
 else:
 self.setDateTextFormat(d, enabled_fmt) # Application du format autorisé
 d = d.addDays(1)
Fin de la classe RangeCalendarWidget

ui_path = os.path.dirname(os.path.abspath(__file__))
ui_path = os.path.join(ui_path, "forms")
form_action1, _ = uic.loadUiType(os.path.join(ui_path, "etape_4.ui"))

class Etape4_dialog(QDialog, form_action1):
 def __init__(self, iface):
 QWidget.__init__(self)
 self.setupUi(self)
 self.iface = iface

 self.network_manager = CloudNetworkAccessManager(self.iface mainWindow())

 self.project_transfer = None
 self.transfer_dialog = None

 self.cloud_login_dialog = CloudLoginDialog(self.network_manager)
 self.cloud_login_dialog.loginFinished.connect(self.login_signal_recu)

 # Remplacez le QCalendarWidget par RangeCalendarWidget
 self.calendar_layout = QVBoxLayout(self.QCalendarWidget_container)
 self.range_calendar = RangeCalendarWidget()
 self.calendar_layout.addWidget(self.range_calendar)
 self.range_calendar.setVisible(False)

 # Dimensionnement de la fenêtre
 self.setFixedSize(1109, 403)
 self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
 self.setFixedSize(QWIDGETSIZE_MAX, QWIDGETSIZE_MAX) # Supprimer la taille fixe

 # Connexion des boutons à leurs méthodes respectives
 self.PB_suivant.clicked.connect(self.suivant)
 self.PB_retour.clicked.connect(self.retour)
 self.PB_annuler.clicked.connect(self.close)
 self.PB_ok.clicked.connect(self.accept)
 self.CkB_selecReseau.stateChanged.connect(self.montrer_entre_select_troncon)

```

```

self.CkB_selecObservation.stateChanged.connect(self.montrer_enti_select_observation)
self.CkB_date.stateChanged.connect(self.montrer_calendrier)
self.CkB_tronconCure.stateChanged.connect(self.filtreTronconCure)
self.PB_modifPel.toggled.connect(self.afficherModifPelwidget)
self.RB_envoiBD.toggled.connect(self.afficherEnvoieBDwidget)

Appel des méthodes pour remplir les ComboBox associé dès l'ouverture du formulaire
self.remplirComboboxReseau()
self.remplirComboboxObservation()

Nom de la BD et du Schema PostgreSQL
self.nomBD = "unima"
self.schemaTampons = "regie_travaux"

Assignation des variables global
self.no_pgError = True

""" Partie 4 : Exporter projet pour pelleteurs """
self.PB_sync4.setEnabled(False)
self.label_statut_connexion_4.setText('<html><head/><body><p>Merci de vous connecter au cloud </p></body></html>')
self.PB_CloudConn_4.clicked.connect(self.open_cloudCon)
self.PB_sync4.clicked.connect(self.accept4)
self.PB_annuler4.clicked.connect(self.close)

""" FIN Partie 4 : Exporter projet pour pelleteurs """

@pyqtSlot()
def login_signal_recu(self):
 QTimer.singleShot(1000, self.peupleComboBoxProjets4)

""" DEBUT : Retour projet pelleteurs """
def open_cloudCon(self):

 # Ouvre le dialog de connexion QFieldCloud
 dlg = CloudLoginDialog(self.network_manager, self iface mainWindow())

 result = dlg.exec_()
 if result:
 QTimer.singleShot(5000, self.connection_status)
 pass

def connection_status(self):
 base_path = os.path.join(os.path.dirname(__file__), 'forms', 'icone')
 if self.network_manager.projects_cache.projects is None:
 self.label_statut_connexion_4.setText('Connexion échouée')
 self.PB_sync4.setEnabled(False)
 icon_path = os.path.join(base_path, 'cloud-computing_failed.png')
 else:
 self.label_statut_connexion_4.setText('Connexion Réussie')
 self.PB_sync4.setEnabled(True)
 icon_path = os.path.join(base_path, 'cloud-computing_success.png')

```

```

pixmap = QPixmap(icon_path)
pixmap = pixmap.scaled(30, 30, Qt.KeepAspectRatio)
self.label_statut_connexion_ico_4.setPixmap(pixmap)

def peopleComboBoxProjets4(self):
 try:
 for cloud_project in self.network_manager.projects_cache.projects:
 self.CB_projetCloud_4.addItem(cloud_project.name)
 except:
 return
def recupererProjetCombo4(self):
 nomProjetselec = self.CB_projetCloud_4.currentText()
 if self.network_manager.projects_cache.projects is not None:
 for cloud_project in self.network_manager.projects_cache.projects:
 if cloud_project.name == nomProjetselec:
 self.current_cloud_project = cloud_project

def show_sync_popup_cloud(self) -> None: #RETOUR AU RAPPORT
 assert self.current_cloud_project is not None, "No project to download selected"

 self.transfer_dialog = CloudTransferDialog.show_transfer_dialog(
 self.network_manager, self.current_cloud_project, None, None, self, preference_action='cloud'
)
 self.transfer_dialog.rejected.connect(self.on_transfer_dialog_rejected)
 self.transfer_dialog.accepted.connect(self.on_transfer_dialog_accepted)
 self.transfer_dialog.open()

def on_transfer_dialog_rejected(self) -> None:
 if self.project_transfer:
 self.project_transfer.abort_requests()

 if self.transfer_dialog:
 self.transfer_dialog.close()

 self.project_transfer = None
 self.transfer_dialog = None

def on_transfer_dialog_accepted(self) -> None:
 QgsProject().instance().reloadAllLayers()

 if self.transfer_dialog:
 self.transfer_dialog.close()
 self.close()

 self.project_transfer = None
 self.transfer_dialog = None

""" FIN : Retour projet pelleteurs """
""" DEBUT: Partie récupération des entités sélectionner """

```

```

def remplirComboboxReseau(self) -> (None): # Méthode qui remplit la CoboBox associé

 self.CB_choixReseau.clear() # Effacer tous les éléments existants de la combobox
 layers = QgsProject.instance().mapLayers().values() # Obtenir une liste de toutes les couches dans le projet
 # Filtrer les couches pour ne garder que celles qui sont des lignes ou des multilignes
(geometryType() == 1)
 line_layers = [layer for layer in layers if layer.type() == QgsMapLayerType.VectorLayer and layer.geometryType() == 1]
 for layer in line_layers: # Pour chaque couche dans la liste line_layer
 self.CB_choixReseau.addItem(layer.name()) # Ajouter les noms des couches filtrées à la QComboBox

def montrer_enti_select_troncon(self) -> (None): # Méthode qui met à jour le label associé pour mettre en avant le
 # nombre d'entités sélectionnées

 self.label_troncon_select.setText("")
 if self.CkB_selecReseau.isChecked():
 layer = self.lineaire_selectionne()
 nb_enti = layer.featureCount()
 self.label_troncon_select.setText(f'{nb_enti}')

```

```

 return layer_selection

def observation_selectionne(self) -> (QgsVectorLayer): # Renvoie la couche corespondante aux observations
 (complète ou seulement les entités selectionnées)

 nom_layer = self.CB_choixObservation.currentText()
 layer = QgsProject.instance().mapLayersByName(nom_layer)[0]

 if not self.CkB_selecObservation.isChecked():
 return layer

 else:
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterFids(layer.selectedFeatureIds()))

 return layer_selection

def remplirCalendar(self) -> (None): # Permet de définir les dates autorisées dans le calendrier

 layer_troncon = self.lineaire_selectionne() # Obtenir le layer sélectionné pour les tronçons
 idx_troncon = layer_troncon.fields().indexOf('hit_date') # Trouver l'index du champ 'hit_date'
 values_troncon = layer_troncon.uniqueValues(idx_troncon) # Obtenir les valeurs uniques de ce champ

 layer_observation = self.observation_selectionne() # Obtenir le layer sélectionné pour les observations
 idx_observation = layer_observation.fields().indexOf('hio_date')
 values_observation = layer_observation.uniqueValues(idx_observation)

 all_values = values_troncon.union(values_observation) # Fusionner les ensembles de valeurs
 filtered_values = {v for v in all_values if not v.isNull()} # Exclure les valeurs NULL et None
 self.range_calendar.setDateAutorise(filtered_values) # Définir les dates autorisées dans le calendrier

def montrer_calendrier(self) -> (None): # Permet d'afficher ou non le calendrier pour un meilleur
 accès utilisateur

 if self.CkB_date.isChecked():
 self.range_calendar.setVisible(True)
 else:
 self.range_calendar.setVisible(False)
 self.setFixedSize(1109, 403) # Fixer la taille de la fenêtre à 1109 x 403
 self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding) # Permettre le redimensionnement
 self.setFixedSize(QWIDGETSIZE_MAX, QWIDGETSIZE_MAX) # Supprimer la taille fixe

 """ Mise en place des différents filtres d'exports """ #RETOUR AU RAPPORT
 def filtreTronconCure(self) -> (QgsVectorLayer): # Methode retournant les entités qui sont curés
 ("hit_verif_curage" = true)

 layer = self.lineaire_selectionne() # Récupération du layer sélectionné
 if not self.CkB_tronconCure.isChecked():
 return layer
 else:
 expression = "hit_verif_curage" = true # Utilisation d'une selection par expression de

```

```

QGISFeatureRequest
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterExpression(expression))
 return layer_selection

def filtreDateTroncon(self) -> (QgsVectorLayer): # Methode retournant les entités linéaires qui sont dans
la liste de date choisi par l'utilisateur
 layer = self.filtreTronconCure()

 if not self.CkB_date.isChecked():
 return layer
 else:
 list_date = self.range_calendar.plageDateString()
 dates_str = ';' .join(f'{date}' for date in list_date)

 expression = f'"hit_date" in ({dates_str})'
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterExpression(expression))
 return layer_selection

def filtreDateObservation(self) -> (QgsVectorLayer): # Methode retournant les entités ponctuelles qui sont
dans la liste de date choisi par l'utilisateur

 layer = self.observation_selectionne()
 if not self.CkB_date.isChecked():
 return layer
 else:
 list_date = self.range_calendar.plageDateString()
 dates_str = ';' .join(f'{date}' for date in list_date) # Formatage des dates pour être lu par PostgreSQL (type
string)

 expression = f'"hio_date" in ({dates_str})'
 layer_selection = layer.materialize(QgsFeatureRequest().setFilterExpression(expression))
 return layer_selection

""" FIN: Partie récupération des entités sélectionner
DEBUT: Partie INSERT-UPDATE dans la base de données tampons """

def get_info_postgis(self, wnom) -> (tuple):
 # renvoie l'hôte, le port, le nom de la DB, le user et le pwd sous forme de tuple. Méthode par Mr.Alain LAYEC
 resu = None
 dbpluginclass = createDbPlugin("postgis")

 for c in dbpluginclass.connections():
 if c.connectionName() == wnom:
 settings = QgsSettings()
 settings.beginGroup(u"/%s/%s" % ('PostgreSQL/connections', wnom))

 if settings.contains("database"):
 whost = settings.value("host")
 wport = settings.value("port")
 wdatabase = settings.value("database")
 wusername = settings.value("username")
 wpASSWORD = settings.value("password")

```

```

 resu = (whost, wport, wdatabase, wusername, wpassword)
 return resu

def linaireCurageExistant(self, conn, schema, res_code_tr, rt_geom) -> tuple: #RETOUR AU RAPPORT
 """
 Méthode renvoyant l'id d'une entité existant déjà dans la BD.
 Vérification si l'entité existe déjà grâce à l'id du réseau associé et as_n_unima.
 """
 if self.no_pgError:
 try:
 with conn.cursor() as cur:
 # Vérifier l'existence dans reseau_travaux
 query_lin = sql.SQL("""
 SELECT rt_id
 FROM {::}
 WHERE res_code_tr = %s AND rt_geom = %s
 """).format(
 sql.Identifier(schema),
 sql.Identifier('reseau_travaux')
)
 cur.execute(query_lin, (res_code_tr, rt_geom))
 lin_result = cur.fetchone()
 return lin_result[0] if lin_result else None
 except psycopg2.Error as e:
 self.no_pgError = False
 message = "Erreur lors de la vérification de l'existence de la donnée: " + e.pgerror
 self.warning(message)
 else:
 return

def travauxExistant(self, conn, schema, hit_num_tranche_be, rt_id) -> tuple:
 """
 Méthode renvoyant l'id d'une entité existant déjà dans la BD.
 Vérification si l'entité existe déjà grâce à son numéros de tranche du BE et rt_id.
 """
 if self.no_pgError:
 try:
 with conn.cursor() as cur:
 query_hit = sql.SQL("""
 SELECT hit_id
 FROM {::}
 WHERE hit_num_tranche_be = %s AND rt_id = %s
 """).format(
 sql.Identifier(schema),
 sql.Identifier('historisation_travaux')
)
 cur.execute(query_hit, (hit_num_tranche_be, rt_id))
 hit_result = cur.fetchone()
 return hit_result[0] if hit_result else None
 except psycopg2.Error as e:
 self.no_pgError = False
 message = "Erreur lors de la vérification de l'existence de la donnée: " + e.pgerror
 self.warning(message)

```

```

else:
 return

def insert_or_update_lineaire_curage(self, conn, schema, layer) -> (None): #RETOUR AU RAPPORT
 if self.no_pgError:
 try:
 for feature in layer.getFeatures(): # Pour chaque entité dans la couche
 attributes = feature.attributes() # Récupération des attributs de l'entité

 res_code_tr = str(attributes[layer.fields().indexOf('res_code_tr')]) # Attribution de la valeur de l'attribut
 'res_code_tr' par son index + cast en string
 as_n_unima = str(attributes[layer.fields().indexOf('id_as')])
 rt_geom = feature.geometry().asWkt()

 # Insérer ou mettre à jour dans la table linaire_curage
 existing_rt_id = self.linaireCurageExistant(conn, schema, res_code_tr, rt_geom)
 with conn.cursor() as cur:
 if existing_rt_id:
 update_query = sql.SQL("""
 UPDATE {}({})
 SET res_code_tr = %s, as_n_unima = %s, rt_geom = ST_GeomFromText(%s, 2154)
 WHERE rt_id = %s
 """).format(
 sql.Identifier(schema),
 sql.Identifier('reseau_travaux')
)
 cur.execute(update_query, (res_code_tr, as_n_unima, rt_geom, existing_rt_id))
 rt_id = existing_rt_id
 print("UPDATE DANS LA TABLE reseau_travaux: ", (rt_id, res_code_tr, as_n_unima, rt_geom))
 else:
 insert_query = sql.SQL("""
 INSERT INTO {}({}) (res_code_tr, as_n_unima, rt_geom)
 VALUES (%s, %s, ST_GeomFromText(%s, 2154))
 RETURNING rt_id
 """).format(
 sql.Identifier(schema),
 sql.Identifier('reseau_travaux')
)
 cur.execute(insert_query, (res_code_tr, as_n_unima, rt_geom))
 rt_id = cur.fetchone()[0]
 print("INSERT DANS LA TABLE reseau_travaux: ", (rt_id, res_code_tr, as_n_unima, rt_geom))

 # Maintenant traiter la table historisation_travaux
 hit_annee = str(attributes[layer.fields().indexOf('hit_annee')])
 hit_tranche = str(attributes[layer.fields().indexOf('hit_tranche')])
 hit_bro_cot = str(attributes[layer.fields().indexOf('hit_bro_cot')])
 hit_aplo_cot = str(attributes[layer.fields().indexOf('hit_aplo_cot')])
 hit_cot_vase = str(attributes[layer.fields().indexOf('hit_cot_vase')])
 hit_temps_broyage = str(attributes[layer.fields().indexOf('hit_temps_broyage')])
 hit_temps_tronconnage = str(attributes[layer.fields().indexOf('hit_temps_tronconnage')])
 hit_verif_depotvase = str(attributes[layer.fields().indexOf('hit_verif_depotvase')])
 hit_verif_curage = str(attributes[layer.fields().indexOf('hit_verif_curage')])
 hit_hauteur_vase = str(attributes[layer.fields().indexOf('hit_hauteur_vase')])

```

```

hit_largeur_fosse = str(attributes[layer.fields().indexOf('hit_largeur_fosse')])
pel_id = str(attributes[layer.fields().indexOf('pel_id')])
hit_date_value = attributes[layer.fields().indexOf('hit_date')]
if isinstance(hit_date_value, QDate):
 hit_date = hit_date_value.toString('yyyy-MM-dd') # Formatage d'un objet QDate en String au format
PostgreSQL
else:
 hit_date = None
hit_num_tranche_be = str(attributes[layer.fields().indexOf('hit_num_tranche_be')])
hit_num_devis = str(attributes[layer.fields().indexOf('hit_num_devis')])

data_to_insert = (
 rt_id,
 hit_annee if hit_annee != 'NULL' else None,
 hit_tranche if hit_tranche != 'NULL' else None,
 hit_bro_cot if hit_bro_cot != 'NULL' else None,
 hit_aplo_cot if hit_aplo_cot != 'NULL' else None,
 hit_cot_vase if hit_cot_vase != 'NULL' else None,
 hit_temps_broyage if hit_temps_broyage != 'NULL' else 0,
 hit_temps_tronconnage if hit_temps_tronconnage != 'NULL' else 0,
 hit_verif_depotvase if hit_verif_depotvase != 'NULL' else None,
 hit_verif_curage if hit_verif_curage != 'NULL' else None,
 hit_largeur_fosse if hit_largeur_fosse != 'NULL' else None,
 hit_hauteur_vase if hit_hauteur_vase != 'NULL' else None,
 pel_id if pel_id != 'NULL' else None,
 hit_date if hit_date != 'NULL' else None,
 hit_num_tranche_be if hit_num_tranche_be != 'NULL' else None,
 hit_num_devis if hit_num_devis != 'NULL' else None)

existing_entity_id = self.travauxExistant(conn, schema, hit_num_tranche_be, rt_id)
with conn.cursor() as cur:
 if existing_entity_id:
 update_query = sql.SQL("""
 UPDATE {}({})
 SET rt_id = %s, hit_annee = %s, hit_tranche = %s, hit_bro_cot = %s, hit_aplo_cot = %s,
 hit_cot_vase = %s, hit_temps_broyage = %s, hit_temps_tronconnage = %s, hit_verif_depotvase = %s,
 hit_verif_curage = %s, hit_largeur_fosse = %s, hit_hauteur_vase = %s, pel_id = %s, hit_date = %s,
 hit_num_tranche_be = %s, hit_num_devis = %s
 WHERE hit_id = %s
 """).format(
 sql.Identifier(schema),
 sql.Identifier('historisation_travaux')
)
 cur.execute(update_query, data_to_insert + (existing_entity_id,))
 print("UPDATE DANS LA TABLE historisation_travaux: ", data_to_insert)
 else:
 insert_query = sql.SQL("""
 INSERT INTO {}({})
 rt_id, hit_annee, hit_tranche, hit_bro_cot, hit_aplo_cot, hit_cot_vase,
 hit_temps_broyage, hit_temps_tronconnage, hit_verif_depotvase, hit_verif_curage, hit_largeur_fosse,
 hit_hauteur_vase, pel_id, hit_date,
 hit_num_tranche_be, hit_num_devis
) VALUES (
 """

```

```

 %s, %s
)
 """).format(
 sql.Identifier(schema),
 sql.Identifier('historisation_travaux')
)
 cur.execute(insert_query, data_to_insert)
 print("INSERT DANS LA TABLE historisation_travaux: ", data_to_insert)

 conn.commit()
except psycopg2.Error as e:
 self.no_pgError = False
 message = "Erreur lors de l'insert ou l'update de la donnée: " + e.pgerror
 self.warning(message)
else:
 return
}

def observationExiste(self, conn, table_name, schema, geometry, res_code_tr) -> (tuple):
#RETOUR AU RAPPORT
Vérification si l'entité existe grâce à sa géométrie et l'id du réseau associé

if self.no_pgError:
 try:
 with conn.cursor() as cur:
 query = sql.SQL("""
 SELECT hio_id
 FROM {}
 WHERE hio_geom = %s AND res_code_tr = %s
 """
).format(
 sql.Identifier(schema),
 sql.Identifier(table_name)
)
 cur.execute(query, (geometry, res_code_tr))
 result = cur.fetchone()
 return result[0] if result else None
 except psycopg2.Error as e:
 self.no_pgError = False
 message = "Erreur lors de l'insert ou l'update de la donnée" + e.pgerror
 self.warning(message)
else:
 return

def insert_or_update_observations_remarques(self, conn, schema, layer) -> (None):
 if self.no_pgError:
 try:
 for feature in layer.getFeatures():
 attributes = feature.attributes()
 geometry = feature.geometry().asWkt()

 obs_id = str(attributes[layer.fields().indexOf('obs_id')])
```

```

hio_commentaire = str(attributes[layer.fields().indexOf('hio_commentaire')])
hio_photo = str(attributes[layer.fields().indexOf('hio_photo')])
hio_date_value = attributes[layer.fields().indexOf('hio_date')]
if isinstance(hio_date_value, QDate):
 hio_date = hio_date_value.toString('yyyy-MM-dd')
else:
 hio_date = None
obs_id = str(attributes[layer.fields().indexOf('obs_id')])
as_n_unima = str(attributes[layer.fields().indexOf('as_n_unima')])
res_code_tr = str(attributes[layer.fields().indexOf('res_code_tr')])

Créer un nouveau tuple avec les valeurs correctes
data_to_insert = (
 hio_commentaire if hio_commentaire != 'NULL' else None,
 hio_photo if hio_photo != 'NULL' else None,
 hio_date if hio_date != 'NULL' else None,
 as_n_unima if as_n_unima != 'NULL' else None,
 obs_id if obs_id != 'NULL' else None,
 res_code_tr if res_code_tr != 'NULL' else None,
 geometry
)

existing_observation_id = str(self.observationExiste(conn, 'historisation_observation', schema, geometry,
res_code_tr))

with conn.cursor() as cur:
 if existing_observation_id != 'None':
 update_query = sql.SQL("""
 UPDATE {}({})
 SET hio_commentaire = %s, hio_photo = %s, hio_date = %s, as_n_unima = %s, obs_id = %s, res_code_tr =
%s, hio_geom = ST_GeomFromText(%s, 2154)
 WHERE hio_id = %s
 """).format(
 sql.Identifier(schema),
 sql.Identifier('historisation_observation'))
 cur.execute(update_query, (data_to_insert + (existing_observation_id,)))
 print("UPDATE DANS LA BD : ", data_to_insert)
 else:
 insert_query = sql.SQL("""
 INSERT INTO {}({}) (hio_commentaire, hio_photo, hio_date, as_n_unima, obs_id, res_code_tr, hio_geom)
 VALUES (%s, %s, %s, %s, %s, %s, ST_GeomFromText(%s, 2154))
 """).format(
 sql.Identifier(schema),
 sql.Identifier('historisation_observation'))
 cur.execute(insert_query, data_to_insert)
 print("INSERT DANS LA BD : ", data_to_insert)

 conn.commit()
except psycopg2.Error as e:
 self.no_pgError = False
 message = "Erreur lors de l'insert ou l'update de la donnée" + e.pgerror

```

```

 self.warning(message)
else:
 return

def sqlQuery(self) -> (None):
 # Connexion à la base de données PostgreSQL
 whost, wport, wdatabase, wusername, wpassword = self.get_info_postgis(self.nomBD)
 conn = psycopg2.connect(
 host=whost,
 port=wport,
 database=wdatabase,
 user=wusername,
 password=wpassword
)
 schema = self.shemaTampons

 layer = self.filtreDateTroncon()
 self.insert_or_update_lineaire_curage(conn, schema, layer)

 layer = self.filtreDateObservation()
 self.insert_or_update_observations_remarques(conn, schema, layer)

 # Fermeture de la connexion à la base de données
 conn.close()

def photoToThot(self): #RETOUR AU RAPPORT

project_path = QgsProject.instance().fileName() # Obtenir le chemin absolu du fichier de projet QGIS

if not project_path:
 message = "Impossible de trouver le chemin du projet. Pensez à enregistrer votre projet"
 self.warning(message)
 return

project_dir = os.path.dirname(project_path) # Extraire le dossier contenant le projet
project_name = os.path.splitext(os.path.basename(project_path))[0] # Obtenir le nom du projet sans l'extension

local_dcim_path = os.path.join(project_dir, 'DCIM') # Construire le chemin vers le dossier DCIM
server_dcim_path = os.path.join(r'\\thot\echange\photo_terrain', project_name) # Construire le chemin vers le dossier sur le thot
server_dcim_path = os.path.join(server_dcim_path, 'DCIM')

if not os.path.exists(local_dcim_path):
 message = f"Le dossier source {local_dcim_path} n'existe pas.
 Peut être il n'y a pas de photo dans votre projet</br>"
 self.warning(message)
 return

if not os.path.exists(server_dcim_path): # Crée le répertoire destination s'il n'existe pas
 try:
 os.makedirs(server_dcim_path)
 except OSError as e:

```

```

message = f"Erreur lors de la création du répertoire destination: {e}"
self.warning(message)
return

try:

 for item in os.listdir(local_dcim_path): # Copier les fichiers en conservant les métadonnées
 src_path = os.path.join(local_dcim_path, item)
 dst_path = os.path.join(server_dcim_path, item)

 if os.path.isfile(src_path): # Vérifie si le fichiers est bien présent
 shutil.copy2(src_path, dst_path)
 except Exception as e:
 message = f"Erreur lors de la copie des fichiers: {e}"
 self.warning(message)
 return

def afficherModifPelwidget(self):
 self.stackedWidget.setCurrentWidget(self.retour_terrain)

def afficherEnvoieBDwidget(self):
 self.stackedWidget.setCurrentWidget(self.expo_cloud_couche)

def suivant(self):
 self.stackedWidget.setCurrentWidget(self.expo_cloud_option)
 self.remplirCalendar()

def retour(self):
 self.stackedWidget.setCurrentWidget(self.expo_cloud_couche)

def accept(self):
 self.sqlQuery()
 self.photoToThot()
 self.close()

def accept4(self):

 self.recupererProjetCombo4()
 self.show_sync_popup_cloud() # Affichez une popup de synchronisation

 #QDialog.accept(self)

def warning(self, message):
 QMessageBox.warning(self, "Avertissement", message)

```

---

## ANNEXE 2

### *Code de l'application Django*

---

d

---

## ANNEXE 3

*Documents annexes de la partie Récolelement Unima*

---

| Questionnement                                  | Réponse Régie                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Réponse BE                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Complémentaire ou Contradictoire | Choix Pole SIG              |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|-----------------------------|
| Que doit afficher le webSig ?                   | <ul style="list-style-type: none"> <li>- Pour les gars terrains :</li> <li>- Plan du réseau sur lequel effectué des travaux</li> <li>- Information sur le broyage/tronçonnage</li> <li>- Sens de vase (indiqué par une flèche)</li> <li>- Système d'alerte ponctuelle (avec remarque et photo optionnelle)</li> <li>- Géolocalisation</li> <li>- Chemin d'accès</li> </ul><br><ul style="list-style-type: none"> <li>- Pour les chefs de chantiers :</li> <li>- Idem</li> <li>- Déco coupe par tranche + zone géographique</li> </ul> | <ul style="list-style-type: none"> <li>- Dépot de vase</li> <li>- Possibilité d'ajout de commentaire</li> <li>- Indication du coté du dépôt</li> <li>- Type de débroussaillage</li> <li>- Indication du coté du débroussaillage</li> <li>- Numéros mare de tonne</li> <li>- Différence entre le prévisionnel et le réel</li> <li>- Système d'authentification pour savoir qui voit quoi</li> <li>- Possibilité de couper des troçon en plusieurs parties</li> </ul> | Complémentaire                   |                             |
| Quels fonctionnalités doivent être présentent ? | <ul style="list-style-type: none"> <li>- Temps passé par broyage mécanique</li> <li>- Temps passé par tronçonnage manuel</li> <li>- Curage effectué (oui/non)</li> <li>- Indication du sens de vase (indiqué par des flèches opposé/réel)</li> <li>- Filtrer par tranche</li> <li>- Filtrer par programme</li> </ul>                                                                                                                                                                                                                  | <ul style="list-style-type: none"> <li>- Filtrer par type de travaux</li> <li>- Filtrer par tranche</li> <li>- Filtrer par type d'observation</li> </ul>                                                                                                                                                                                                                                                                                                            | Complémentaire                   |                             |
| A qui ça va servir                              | <ul style="list-style-type: none"> <li>- Règle</li> <li>- Conducteurs d'Engin</li> <li>- Chef de Chantier</li> <li>- Commanditaire</li> <li>- Président d'AS pour sulif</li> </ul>                                                                                                                                                                                                                                                                                                                                                    | Ne comprend pas trop l'utilité d'un tel outils, surtout pour des petits tronçons. Ne sait pas trop à quoi ça peut servir.                                                                                                                                                                                                                                                                                                                                           | Contradictoire                   | 1 pt pour la régie          |
| Ils vont s'en servir pour faire quoi ?          | <ul style="list-style-type: none"> <li>- Suivre l'avancement du chantier</li> <li>- Sens de vase</li> <li>- Se repérer facilement sur le chantier</li> </ul>                                                                                                                                                                                                                                                                                                                                                                          | <ul style="list-style-type: none"> <li>- Suivre avancement du chantier</li> <li>- Préparer le chantier</li> </ul>                                                                                                                                                                                                                                                                                                                                                   | Complémentaire                   |                             |
| Quelle charte graphique ?                       | - Plan de référence papier                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Charte graphique différentes par personnes qui réalise le ppt                                                                                                                                                                                                                                                                                                                                                                                                       | Contradictoire                   | 1pt régie                   |
| Géolocalisation                                 | - Oui                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Complémentaire                   |                             |
| Hors connection ?                               | - Oui                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Ne savent pas                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                  |                             |
| Facile d'utilisation et de compréhension        | <ul style="list-style-type: none"> <li>- Cliquer sur un tronçon &gt; mettre à jour les informations pertinente</li> <li>- Cliquer sur la carte &gt; Afficher problème ponctuel</li> </ul>                                                                                                                                                                                                                                                                                                                                             | Veulent la possibilité que les gars sur le terrain puisse dire efficacement ce qu'ils ont fait, ce qui implique de créer ou modifier de la géométrie                                                                                                                                                                                                                                                                                                                | Contradictoire                   | 1pt régie (à éclaircir)     |
| Possibilité d'indiquer où déverser la vase      | Peut revoyer comment c'est indiqué à l'heure actuelle. Mettre une flèche pour dire de quelle coté est le sens de vase                                                                                                                                                                                                                                                                                                                                                                                                                 | Indication précise de où peut être mis la vase                                                                                                                                                                                                                                                                                                                                                                                                                      | Contradictoire                   | be = résultat ou possible ? |
| Quel utilitaire choisir                         | Qfield privilégié                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Web Sig type geo marennes                                                                                                                                                                                                                                                                                                                                                                                                                                           | Complémentaire                   |                             |

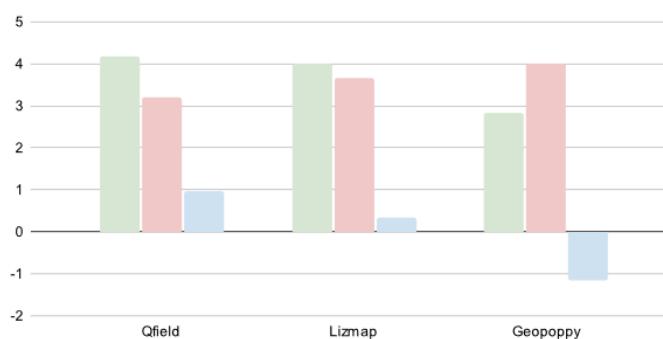
|        |                                                                                    |        |                                                                                                              |       |
|--------|------------------------------------------------------------------------------------|--------|--------------------------------------------------------------------------------------------------------------|-------|
| Qfield | Possibilité d'utilisation hors connexion                                           | 4      | Nécessite la récupération des tablettes par le Conducteur de travaux ou une connexion internet à la tablette | 3     |
|        | Utilisation lié à un projet Qgis (relation, symbologie, attributaires)             | 4      | Dessin peut être peu intuitif pour l'utilisateur                                                             | 3     |
|        | Synchronisation possible du projet via extension QfieldSync en connection internet | 4      | Solution gratuite mais pourrait devenir payante si stockage important ou écriture sur du PostGis             | 4     |
|        | Édition de formulaire complet et simple                                            | 5      | Nécessite l'achat potentielle de tablette                                                                    | 4     |
|        | Déverser les données de la BD Unima vers Lizmap via script                         | 3      | Mise à jour de la BD manuel (via synchronisation ou récupération du projets)                                 | 2     |
|        | Ecriture des données dans la BD Unima via Qgis                                     | 5      |                                                                                                              |       |
|        | Note Positive                                                                      | 4,17/5 | Note Négative                                                                                                | 3,2/5 |

|        |                                                                        |     |                                                                                          |        |
|--------|------------------------------------------------------------------------|-----|------------------------------------------------------------------------------------------|--------|
| Lizmap | Pas d'installation nécessaire                                          | 4   | Nécessite une connexion internet                                                         | 3      |
|        | Utilisation lié à un projet Qgis (relation, symbologie, attributaires) | 4   | Formulaire nécessite un travail en amont supplémentaire (Qgis et Lizmap) et peu intuitif | 5      |
|        | Édition de formulaire complet                                          | 4   | Nécessite de récupérer les données lizmap par un administrateur                          | 3      |
|        | Données directement intégré dans le WebSIG et BD Lizmap                | 5   |                                                                                          |        |
|        | Fonctionne sur n'importe quel device                                   | 3   |                                                                                          |        |
|        |                                                                        |     |                                                                                          |        |
|        | Note Positive                                                          | 4/5 | Note Négative                                                                            | 3,67/5 |

| Logiciel | Pro                                                                              | pondération | Cons                                                                     | pondération |
|----------|----------------------------------------------------------------------------------|-------------|--------------------------------------------------------------------------|-------------|
| GeoPoppy | Open source                                                                      | 1           | Achat du matériel de base (environ 100€ par geopolopy)                   | 2           |
|          | Utilisation lié à un projet Qgis (relation, symbologie, attributaires)           | 4           | Formulaire nécessite un travail en amont supplémentaire (Qgis et Lizmap) | 4           |
|          | Édition de formulaire complet                                                    | 4           | Nécessite de récupérer les geopolopy pour synchronisation                | 5           |
|          | Données directement intégré dans le WebSIG et BD Lizmap et potentiellement UNIMA | 4           | Interface potentiellement peu intuitive                                  | 4           |
|          | Fonctionne sur n'importe quel device par connection WIFI                         | 3           | Demande de la R&D pour la compréhension et l'utilisation de l'outils     | 4           |
|          | Fait fonctionné le local (Développé à St Laurent de la prés)                     | 1           |                                                                          |             |
|          | Note Positive                                                                    | 2,83/5      | Note Négative                                                            | 3,8/5       |

### Qfield, Lizmap et Geopropy

■ note positive ■ note negative ■ Note globale



```

1 CREATE TABLE observation_type (
2 obs_id SERIAL PRIMARY KEY,
3 obs_nom VARCHAR(255) NOT NULL
4);
5 CREATE TABLE pelleteur (
6 pel_id SERIAL PRIMARY KEY,
7 pel_nom VARCHAR(255) NOT NULL
8);
9 CREATE TABLE test_reseau.historisation_observation (
10 hio_id SERIAL PRIMARY KEY,
11 hio_commentaire TEXT,
12 hio_photo VARCHAR,
13 hio_date DATE ,
14 as_n_unima TEXT REFERENCES test_reseau.limites_as(n_unima),
15 obs_id INT REFERENCES test_reseau.observation_type(obs_id),
16 res_id INT REFERENCES test_reseau.reseau_syndical(id),
17 hio_geom GEOMETRY(Point, 2154)
18);
19 CREATE TABLE test_reseau.historisation_travaux [
20 hit_id SERIAL PRIMARY KEY,
21 lin_id INT REFERENCES (test_reseau.linaire_curage(lin_id)),
22 hit_annee INT,
23 hit_tranche INT,
24 hit_bro_cot INT CHECK (hit_bro_cot IN (0, 1, 2)),
25 hit_aplo_cot INT CHECK (hit_aplo_cot IN (0, 1, 2)),
26 hit_cot_vase INT CHECK (hit_cot_vase IN (0, 1, 2)),
27 hit_temps_broyage INT,
28 hit_temps_troncottage INT,
29 hit_verif_depotvase INT CHECK (hit_verif_depotvase IN (0, 1, 2)),
30 hit_verif_curage INT CHECK (hit_verif_depotvase IN (0, 1)),
31 pel_id INT REFERENCES test_reseau.pelleteur(pel_id),
32 hit_date DATE,
33 hit_num_tranche_be TEXT,
34 hit_num_devis TEXT
35];
36 CREATE TABLE test_reseau.linaire_curage(
37 lin_id SERIAL PRIMARY KEY,
38 res_code_tr INT REFERENCES test_reseau.reseau_syndical(code_tr),
39 as_n_unima VARCHAR(255) REFERENCES test_reseau.limites_as(n_unima),
40 lin_geom GEOMETRY(MultiLineString, 2154)
41)

```

Script SQL de la base de données test

regex utilisé pour requête SQL ASAPerimetre

### Expression régulière pour code\_postal

`^\d{5}`

- `^` : Ancre qui correspond au début de la chaîne. Cela signifie que nous cherchons un motif qui commence dès le début de la chaîne.
- `\d{5}` : Correspond exactement à 5 chiffres (de 0 à 9).
- `(espace)` : Correspond à un espace après les 5 chiffres.

Ainsi, l'expression `^\d{5}` vérifie si la chaîne commence par exactement 5 chiffres suivis d'un espace.

### Expression régulière pour nom\_ville

`\d{5} (.*)$`

- `\d{5}` : Correspond exactement à 5 chiffres (de 0 à 9).
- `(espace)` : Correspond à un espace après les 5 chiffres.
- `(.)` : Capture tout ce qui suit l'espace, c'est-à-dire n'importe quel caractère (.) répété 0 ou plusieurs fois (\*). Les parenthèses () sont utilisées pour capturer cette partie de la chaîne, ce qui nous permet de l'extraire.
- `$` : Ancre qui correspond à la fin de la chaîne. Cela signifie que nous voulons correspondre jusqu'à la fin de la chaîne après les 5 chiffres et l'espace.

L'expression `\d{5} (.*)$` extrait tout ce qui suit les 5 chiffres et l'espace jusqu'à la fin de la chaîne.