

IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département informatique

Réalisation de plugins pour le système d'information QGIS

Rapport de stage de DUT informatique
Administration : Conseil départemental de Meurthe-et-Moselle

Rodolphe Drouet
Maître de stage : Hervé Vitoux
Tuteur : Thomas Giletti
Année universitaire 2020–2021

IUT Nancy Charlemagne
Université de Lorraine
2 ter boulevard Charlemagne
BP 55227
54052 Nancy Cedex

Département informatique

Réalisation de plugins pour le système d'information QGIS

Rapport de stage de DUT informatique
Administration : Conseil départemental de Meurthe-et-Moselle

Rodolphe Drouet
Maître de stage : Hervé Vitoux
Tuteur : Thomas Giletti
Année universitaire 2020–2021

Remerciements

Je tiens à remercier le conseil départemental de m'avoir accueilli pour mon stage, et qui m'a permis de le réaliser dans de bonnes conditions malgré la période actuelle.

Je remercie mon maître de stage, Mr Hervé Vitoux, responsable du pôle SIG au conseil départemental qui m'a accompagné tout au long de ce stage et qui a été d'une aide indispensable dans la direction de mes travaux. Grâce à lui, j'ai pu perfectionner mes compétences en informatiques et confirmer mon parcours professionnel.

Je remercie aussi l'équipe complète du pôle SIG pour leur gentillesse à mon égard, ainsi que leur bienveillance. Leur travail d'équipe est remarquable et j'ai beaucoup aimé travailler avec eux. Ensemble, nous avons pu progresser plus vite.

Table des matières

1. Introduction.....	6
2. Corps du mémoire	7
2.1 Présentation de l'entreprise.....	7
2.2 Présentation du travail réalisé.....	11
2.2.1 Première mission : Conception d'un plugin de génération de fichier JSON.....	12
2.2.2 Deuxième mission : Implémentation d'une solution pour rendre les formulaires plus ergonomiques.....	21
2.2.3 Troisième mission : Développement d'un plugin de correction orthographique de base de données	24
3. Conclusion.....	28
4. Bibliographie.....	29
5. Annexe.....	30
6. Annexe des figures.....	31

1. Introduction

Dans le cadre de ma deuxième et dernière année de D.U.T. Informatique à l'Institut Universitaire de Technologie de Nancy-Charlemagne, je dois effectuer un stage d'une durée de 2 mois. Celui-ci a pour but de terminer mon parcours au sein du Diplôme Universitaire Technologique Informatique, afin d'affiner mes compétences dans ce domaine et de découvrir les perspectives du monde du travail en entreprise.

Dans ce rapport, je vais commencer par présenter l'administration dans laquelle j'ai effectué ledit stage ainsi que ses différentes missions puis j'expliquerai les missions que j'ai réalisé au sein de cet environnement à savoir la conception et le développement de plugins pour le système d'information géographique QGIS. En effet, certains utilisateurs de QGIS ont un besoin de simplification des tâches qui peuvent être pénibles, ce à quoi je vais tenter de répondre.

2. Corps du mémoire

2.1. Présentation de l'entreprise



Figure 1 : Logo du conseil départemental

Il convient d'abord de présenter l'administration dans laquelle j'ai effectué mon stage.

Le conseil départemental est une collectivité locale qui opère dans tout le département de Meurthe-et-Moselle. Il a été créé au 19^{ème} siècle mais ne devient autonome qu'à partir de 1982 lorsque l'état décide de décentraliser une partie de ses missions et de les confier aux conseils départementaux.

Le cœur de ce lieu se compose d'une assemblée d'élus. C'est eux qui définissent l'orientation politique du département. Parmi cette structure se trouve plusieurs services administratifs qui touchent un grand ensemble de domaines sur lesquels agit le conseil. Ces services sont ici pour préparer le travail des élus et les aider dans la prise de décision.

C'est près de 3000 agents qui agissent ensemble pour faire du territoire un endroit meilleur tous les jours, avec 140 métiers différents qui sont mis en œuvre au quotidien, de l'ingénieur au médecin, de l'agent de service au travailleur social. Ces agents et métiers sont répartis en plusieurs secteurs qui assurent le pilotage hiérarchique. Et ces secteurs sont eux aussi divisés en plusieurs branches. Un organigramme permet de visualiser cette structure.

Organigramme général des services départementaux

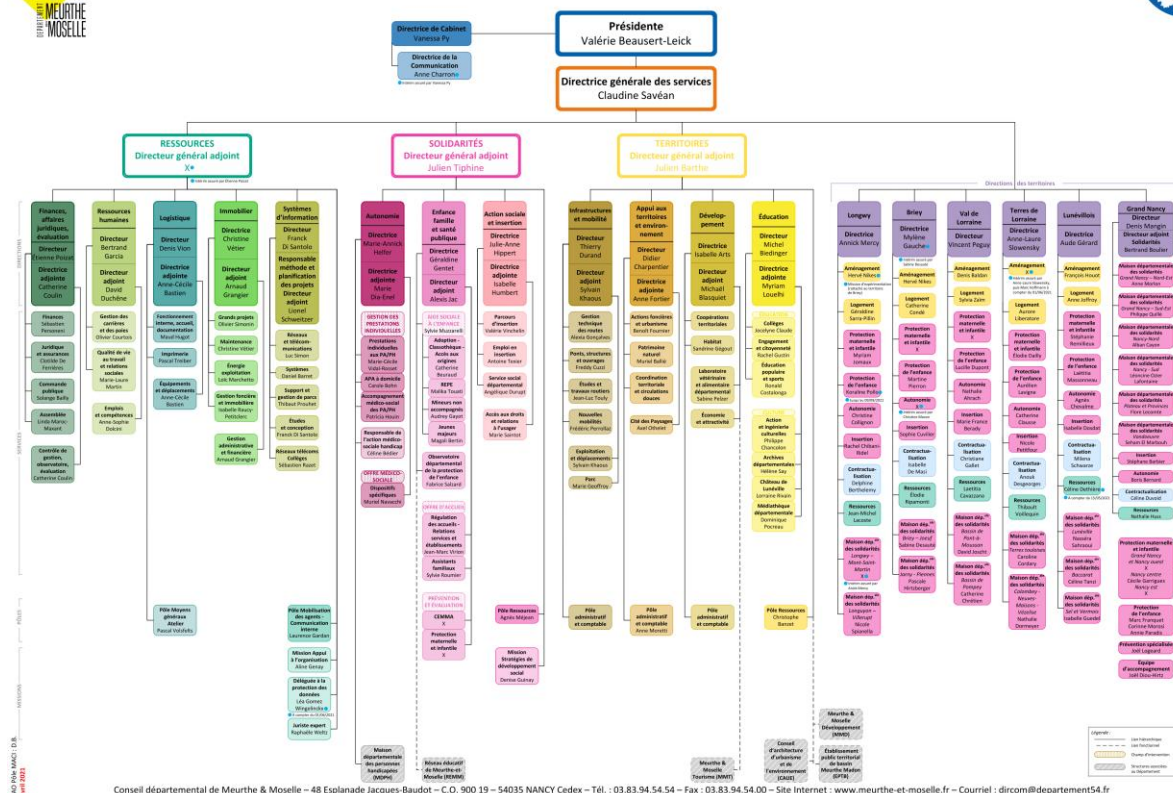


Figure 2 : Organigramme général de la structure du CD

Sur cet organigramme, nous pouvons observer les principaux secteurs d'activités du conseil : Ressources, Solidarités et Territoires.

Le secteur Solidarités s'occupe par exemple des différentes Maisons Départementales des Solidarités. Leurs actions consistent à aider les personnes en difficultés financières à mieux vivre dans le territoire.

Le secteur Ressources contient les ressources humaines et la logistique ainsi que le système d'information principalement. Ils s'occupent de gérer les ressources à disposition du département afin d'assurer le bon fonctionnement du conseil et de ses actions.

Enfin, le secteur Territoire s'occupe d'aider les personnes à mieux se déplacer dans le territoire en réalisant des actions de soutien, notamment auprès des personnes âgées. C'est lui qui gère la Maison départementale des personnes handicapées. Il est subdivisé en plusieurs catégories : Infrastructures et mobilité, Appui aux territoires et environnement, Éducation et la branche dans laquelle j'ai effectué mon stage : Développement. Voici un organigramme plus détaillé de cette branche :

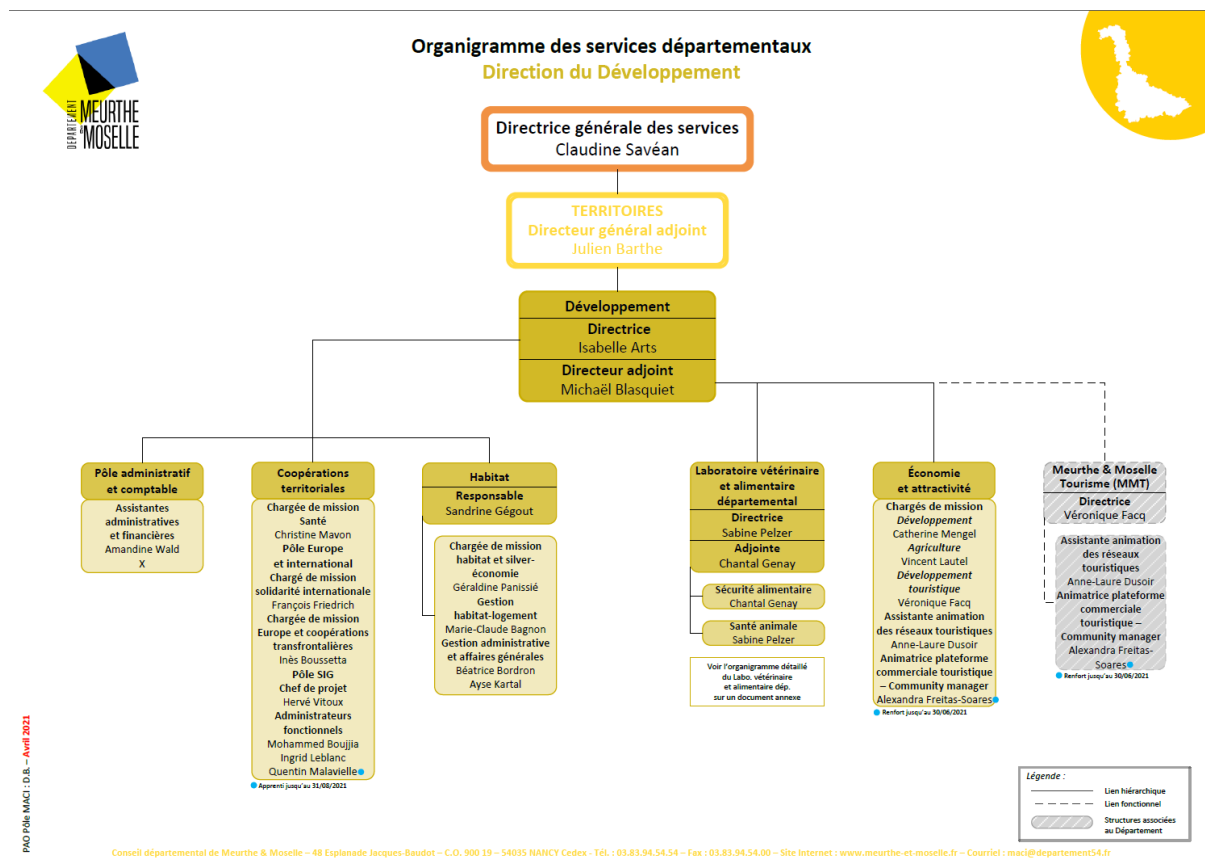


Figure 3 : Organigramme de la branche développement

La sous-branche qui me concerne se trouve être les Coopérations territoriales. Plus particulièrement le pôle SIG (Système d'information géographique). Ce pôle est composé de quatre personnes : Mohammed Boujjia, Ingrid Leblanc, Quentin Malavielle qui sont administrateurs fonctionnels et Hervé Vitoux, Chef de projet. Les trois premiers sont en télétravail 95% du temps, quant à Mr Vitoux, il est présent deux jours par semaine afin de me superviser.

Mohammed, Ingrid et Hervé Vitoux sont des spécialistes du logiciel QGIS, un logiciel de système d'information géographique libre. Il permet de réaliser des couches de territoire contenant des informations comme la liste complètes des lignes de bus ou des lignes de canalisation des eaux usées. QGIS utilise Postgresql, un système de gestion de base de données libre. Il est l'alternative principal au concurrent MySQL. En effet, le conseil privilégie des outils libres de droits car ils sont gratuits mais surtout parce qu'ils permettent de réaliser plus simplement et avec plus de choix les mêmes activités. L'API PyQGIS, permettant de développer des extensions en langage Python, un langage apprécié des informaticiens, est intégrée, il est simple et autorise de nombreuses fonctionnalités.

Quentin Malavielle est un apprenti développeur web en alternance auprès d'Hervé Vitoux. Il s'occupe de développer une application web pour l'aide à la prise de décision pour les maires des communes du département. Cette web application utilise un fichier spécifique JSON pour charger les informations requises et par la suite les afficher dans le navigateur. Avec ces informations, les maires peuvent visualiser rapidement ce dont ils ont besoin et ce qui peut être réalisable comme projet. Cependant, il peut être fastidieux de remplir ce fichier JSON à la main. Quentin a donc besoin d'un moyen d'automatiser cette construction du fichier de manière simple et rapide. Il doit être capable de gérer tous les cas de figures possibles et être conforme aux besoins de son application.

Voilà donc ma première mission : développer une extension QGIS en python capable de répondre au besoin de Quentin.

Cette extension devra utiliser les bases de données via postgresql de QGIS. Cependant, il s'avère que certaines bases de données proviennent d'organisations tiers au conseil. Il arrive donc qu'il y ait des erreurs de frappe ou des problèmes d'uniformisation. Mr Vitoux a exprimé le souhait d'un outil de correction de ces fautes à nouveau via une extension pratique et simple de QGIS.

Ceci est ma deuxième mission : développer une extension capable de remplacer les fautes d'orthographe dans les bases de données de QGIS.

Mr Vitoux m'a fait part d'un troisième besoin. Au sein du conseil départemental, soixante personnes sont des utilisateurs spécialisés de QGIS. Ces utilisateurs travaillent avec le logiciel et parfois ont besoin d'afficher les informations en clair d'un point précis d'une couche, comme les informations d'une école. QGIS propose un formulaire basique pour répondre à cela, mais il n'est pas forcément très ergonomique. Les utilisateurs aimeraient donc une solution simple qui puisse améliorer l'affichage de ces formulaires.

Ce sera ma troisième et dernière mission : trouver et mettre en place une solution qui permette d'améliorer le design de ces formulaires qu'utilisent les soixante référents.

Une fois ces missions définies, il est temps de présenter le travail réalisé pour chacune d'entre elles.

2.2 Présentation du travail réalisé

Afin d'avoir une meilleure vue sur mon travail réalisé durant le stage, j'ai créé un diagramme de Gantt qui récapitule les missions que j'ai effectué pendant ces deux mois.

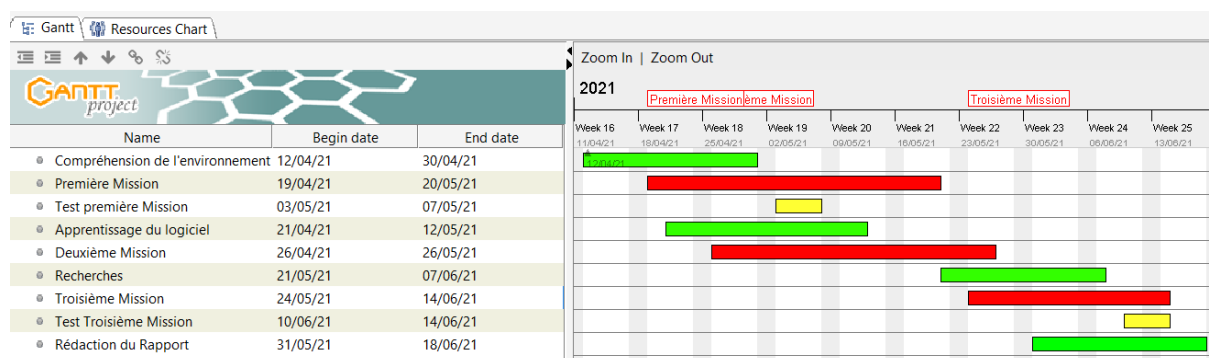


Figure 4 : Diagramme de Gantt

J'ai divisé mes réalisations en trois catégories :

- **Vert** : Cette couleur correspond aux phases de recherches et d'appropriation de l'environnement. C'est essentiellement ce que j'ai dû réaliser avant de commencer à faire quoique ce soit, de la conceptualisation aux recherches nécessaires sur un sujet, à l'apprentissage d'un nouveau langage de programmation.
- **Jaune** : Le jaune correspond aux phases de test que j'ai réalisé afin de vérifier que tout fonctionne et que le travail est conforme aux attentes du client.
- **Rouge** : La dernière couleur correspond aux phases de développement des outils. C'est ce qui représente la majeure partie du travail qui a été réalisé.

Chaque fin de journée au cours de laquelle mon maitre de stage était présent, je lui faisais un rapport sur mes avancés et ensemble nous discussions de la direction que je devais prendre afin de correspondre aux attentes. Ces points réguliers m'ont permis de me rassurer quant à ma démarche. Ceci m'amène à parler de ma première mission.

2.2.1 Première Mission : Conception d'un plugin de génération de fichier JSON

La première chose que j'ai réalisé au début de mon stage a été de commencer par analyser mon environnement de travail. J'ai donc apprivoisé les outils dont j'allais avoir besoin, à savoir en premier lieu le logiciel QGIS. J'ai ainsi découvert comment fonctionnait l'outil, quel était son but et comment il était utilisé par l'équipe SIG. Ce dernier permet donc de charger des couches de données depuis postgresql afin de réaliser des cartes avec les données désirées.

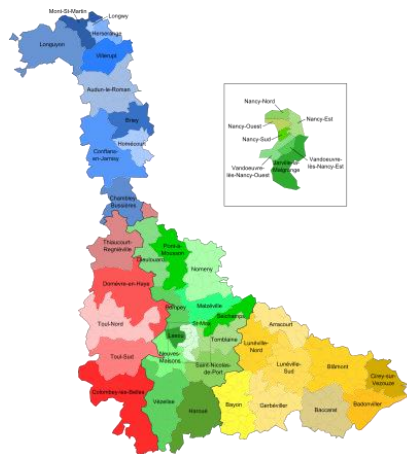


Figure 5 : Carte du département réalisé à l'aide de QGIS

J'ai obtenu un accès en lecture seule afin de pouvoir expérimenter avec de véritables couches sans pour autant risquer de modifier la base de données originale. C'est en découvrant comment fonctionne les couches de QGIS que j'ai commencé à pouvoir me faire une idée de la façon dont j'allais procéder pour répondre à ce besoin. Chaque couche est décomposée en champ/attribut et en entité.

Pour illustrer cela, voici un exemple : il existe une couche Parking. Cette couche contient plusieurs entités, un parking numéro un, deux, trois etc, autant que nécessaire. Chaque parking contient une liste d'attributs/champs : id, nom, type, accès et autres mentions utiles. Chaque entité possède donc des valeurs pour ces attributs qui sont stockées dans une base de données et importées via postgresql. Un attribut geometry est bien sur présent afin de pouvoir localiser avec précision le lieu exact de l'entité.

Après avoir bien intégré et maîtrisé le fonctionnement du logiciel, je me suis intéressé à la partie qui concerne les extensions de QGIS. En effet, le logiciel étant libre de droits, n'importe qui peut développer son extension personnalisée afin de répondre à un besoin spécifique. C'est un des avantages de la licence libre de droits. J'ai donc commencé mes recherches sur le développement d'extension pour QGIS. Pour cela, la documentation faite par les créateurs a été très utile. Elle y recense les étapes du développement nécessaires à l'élaboration de l'extensions. Il y a des fichiers clés qu'il faut absolument faire et le langage de programmation est Python, un langage que je ne connaissais pas auparavant.

La première étape est donc d'étudier la structure d'un plugin QGIS. On a donc plusieurs fichiers clés :

- `__init__.py` est le fichier d'entrée de l'extension. C'est lui qui lance l'extension.
- `mainPlugin.py` est le code principal de l'extension qui contient toutes les actions.
- `resources.qrc` est le document en xml qui contient tous les chemins relatifs vers les ressources de l'extension.
- `resources.py` est la traduction en python du fichier `resources.qrc`
- `metadata.txt` qui contient les informations générales du plugin comme la version, le nom et autres informations utiles sur le plugin.
- `form.ui` est un fichier non obligatoire mais qui concerne la forme graphique que va prendre le plugin dans l'interface de QGIS

Une fois que l'on maîtrise ce squelette, on peut commencer à travailler à l'intérieur afin de construire le corps. Mon premier réflexe a été de vérifier si ce type de plugin n'existait pas déjà ailleurs. Après de nombreuses recherches, il n'y en avait pas d'existant à ce type de travail, cependant, à la suite d'une discussion avec mon maître de stage, j'ai appris que son ancien apprenti avait développé une extension qui faisait en partie ce qui était demandé. Malheureusement, le plugin avait quelques problèmes et ne fonctionnait plus. J'ai donc décidé de reprendre son travail et de le mettre au goût du jour pour répondre au besoin actuel.

Le travail de l'ancien apprenti est stocké sur un répertoire Github, un outil que je connais bien qui permet de stocker du code. Il m'a donc fallu analyser le code de mon prédécesseur. C'est alors que j'ai compris une chose essentielle : la documentation du code est quelque chose de très important. En effet, ce dernier n'était malheureusement que peu détaillé et expliqué, ce qui m'a ralenti dans sa compréhension. S'est ajouté à cela le fait je ne connaissais pas non plus le langage utilisé. J'ai donc commencé par développer des programmes simples en Python afin de me familiariser avec ce nouveau langage. Puis progressivement, je suis arrivé à écrire mon premier plugin Python qui écrit « Hello World ! ». Dans ce premier plugin, on retrouve la structure définie par la documentation de QGIS. Ceci m'a permis de mieux comprendre chaque partie de la structure et ce qu'elle réalise, afin de mieux me familiariser avec le code de Hugo, l'ancien apprenti. Son code utilise l'API PyQGIS, qui permet d'interagir avec QGIS via une console python. Le plugin s'initialise en récupérant tous les attributs de toutes les couches présentes dans le projet QGIS, et les disposent dans un tableau qui est lui-même transformé au format JSON puis est écrit dans un nouveau fichier tel quel. Entre temps, un formulaire apparaît qui contient tous les champs et qui permet à l'utilisateur de modifier certaines propriétés comme si l'affichabilité du champ ou pas. En appuyant sur le bouton Generate, le plugin écrit dans un fichier JSON la configuration de base. Voici un schéma qui résume cela de manière plus simple et ergonomique :

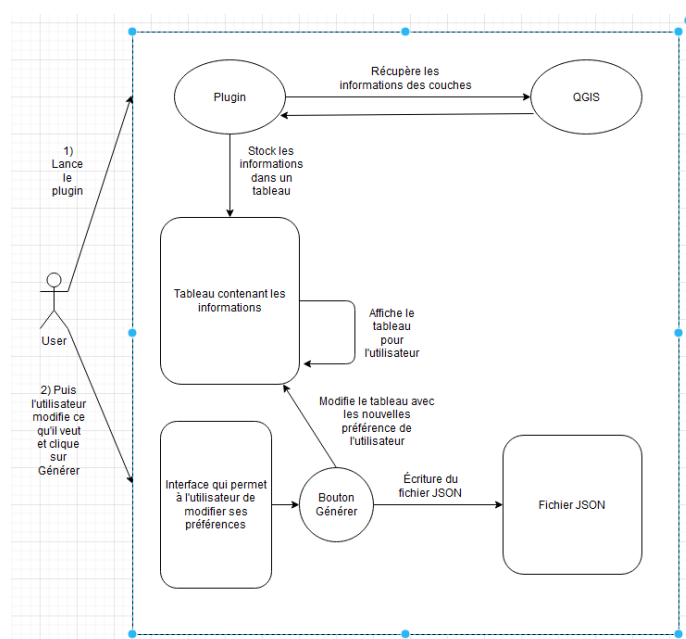


Figure 6 : Cas d'utilisation du plugin de Hugo réalisé avec Draw.io

Ce développement m'a permis d'expérimenter le travail en équipe. Mon client ici étant Quentin, l'apprenti de Mr Vitoux, j'ai donc pris l'initiative de le contacter pour commencer à travailler avec lui. La communication entre le client et le prestataire est très importante, c'est pourquoi je dois beaucoup communiquer avec Quentin pour définir précisément son besoin. La première chose à faire était de lui demander en quoi le plugin actuel d'Hugo ne convenait pas à la demande de Quentin. Il m'a donc expliqué que le plugin présentait plusieurs problèmes :

- Le plugin gère les doublons des champs, ils ne les affichent qu'une seule fois.
- Le plugin ne gère pas tous les cas de figures possibles au niveau des types de champs.
- Le plugin est surchargé d'informations inutiles, il faut donc traiter cela.
- La transparence est un problème, on ne sait pas à quelle couche correspond quel attribut.

C'est avec ce cahier des charges que j'ai orienté mon travail tout au long de ma première mission.

La première phase fut la conceptualisation de mes modifications. Il a fallu que je repère l'endroit qui pose un problème et quelles parties me seraient utiles et lesquelles ne le seraient pas. J'ai donc effectué une phase poussée de test afin de reproduire les erreurs encourues par Quentin. Cela m'a permis de trouver les lignes incorrectes et ainsi focaliser mon intervention précisément. Le code étant très décomposé, il m'est alors relativement facile de modifier ce dont j'ai besoin.

Deux fichiers me concernent :

- Le fichier `output.py` qui s'occupe de retourner le tableau contenant les éléments et qui contient la fonction principale de recherche d'éléments.
- Le fichier `form.py` qui lui s'occupe de l'affichage de ce tableau dans un formulaire pour l'utilisateur.

Le plugin se base principalement sur la modification de la structure de données des informations. En effet, les informations sont d'abord recueillies en tant que liste de valeurs, puis implémentées en dictionnaire de liste et enfin transformées en dictionnaire de dictionnaire.

Le dictionnaire en Python est une structure de donnée très intéressante car elle est très proche de la structure des fichiers JSON : c'est un couple clé/valeur qui permet de stocker efficacement les données et de les parcourir très vite. Ceci est essentiel à comprendre car c'est le cœur du mécanisme du plugin. J'ai donc dû effectuer de nombreux tests de transformation de structure de données afin d'obtenir le résultat souhaité et de comprendre comment cela fonctionne car c'est important.

Pour rappel, on souhaite initialement ne pas gérer les doublons, on veut obtenir toutes les occurrences d'une couche. En effectuant un travail de réflexion sur ce que cela implique d'avoir des doublons, je me suis rendu compte qu'il faudrait un système qui permet de singulariser une occurrence. Plusieurs solutions s'offraient à moi, mais j'ai retenu celle de l'identification par nom de couche et du tri par couche. En effet, chaque champ aura le nom de sa couche à côté et sera stocker avec les autres champs de la même couche. Ce choix technique permet de fournir à l'utilisateur une certaine ergonomie sur l'affichage des champs.

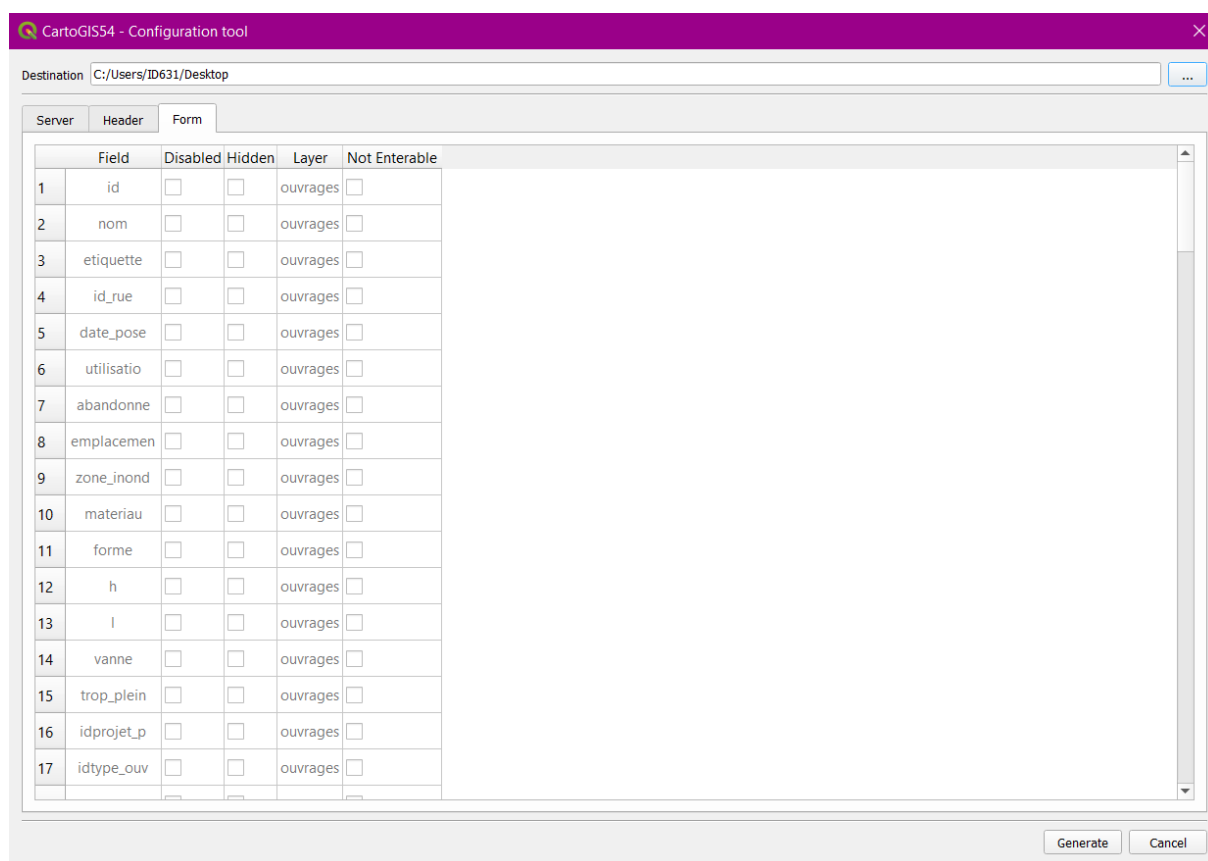


Figure 7 : Image du plugin après modification

Sur ce schéma, on peut observer que les noms des champs apparaissent avec le nom de leurs couches. L'utilisateur peut désormais mieux s'y retrouver dans la liste des champs. Pour se faire, j'ai dû ajouter une propriété Layer (couche) au dictionnaire de base qui contiendra la

couche en question de l'attribut. Le parcours se fait par couche et par attribut et désormais affiche tous les champs sans problème de doublons car le champ est identifié par sa couche. J'ai donc modifié le formulaire pour qu'il accueille désormais le nom de la couche.

Pour répondre au deuxième point du cahier des charges, il faut d'abord comprendre que QGIS classe ses champs par type. Ces types peuvent être « texte », « nombre », « liste de valeurs » et autres. Il faut donc traiter tous les types possibles car le type est un élément déterminant dans le fichier JSON car c'est lui qui va indiquer à la web application de Quentin comment l'afficher. Si l'attribut est une boîte à cocher mais qu'il a le type « zone de texte », la web application l'affichera comme une zone de texte et non une boîte à cocher.

J'ai donc dû rechercher comment était nommé chaque type dans QGIS et ainsi écrire des vérifications supplémentaires dans le code pour traiter tous les cas possibles. Sous la photo ci-dessous, on peut observer les différents tests réalisés afin de vérifier quelle est la valeur de « Style ». S'il correspond à une telle valeur, alors on ajoute dans le JSON le type correspondant.

```
if t == "TextEdit":
    if "IsMultiline" in o:
        if o["IsMultiline"]:
            return self.add_textArea(config)
        return self.add_inputText(config)
if t == "ValueMap":
    return self.add_selectBox(config)
if t == "Range":
    test = o.get("Style")
    if test:
        if o["Style"] == "Slider":
            return self.add_inputRangeSlider(config)
        if o["Style"] == "Dial":
            return self.add_inputRangeSlider(config)
        return self.add_inputRangeSpinBox(config)
    else:
        attribute = config["name"]
        message1 = "The style of this Range attribute does not exist, please change it in order to make it appear in the plugin : " + attribute
        iface.messageBar().pushMessage("Warning", message1, level=Qgis.Critical, duration=5)
if t == "DateTime":
    return self.add_inputDate(config)
if t == "CheckBox":
    return self.add_checkBox(config)
if t == "ExternalResource":
    return self.add_fileInput(config)
if t == "":
    layer = config["layer"]
    message2 = "You must make a modification on one of the attributes of this layer in order to make it appear in the plugin : " + layer
    iface.messageBar().pushMessage("Warning", message2, level=Qgis.Critical, duration=5)
```

Figure 8 : Code correspondant aux vérifications des cas de figures

J'ai pris le soin d'ajouter des messages aux utilisateurs concernant un problème que j'ai rencontré. Il s'agit du fait que la configuration d'un champ, donc son type et les autres informations qui vont avec, ne se crée pas tant qu'il n'y a pas eu de modification de la couche du champ en question. Cela est dû à un problème du logiciel QGIS qui est donc hors de ma

portée. En effet, il existe un cache dans QGIS et ce cache n'est pas mis à jour tant qu'il n'y a pas de mise à jour d'une des couches pour une raison que je n'ai pas pu comprendre.

Afin d'être le plus claire possible sur ce problème, j'ai fait en sorte d'afficher quelle couche et quel champ étaient concernés par ce problème afin que l'utilisateur puisse le corriger. J'ai bien entendu expliqué le problème dans le fichier README du dépôt GitHub, car ce plugin est destiné au grand public. C'est d'ailleurs pour cette raison que j'ai commenté tout le code plus en détail et en anglais afin de toucher un maximum de personne.

Le dernier point m'a donné plus de fil à retordre. Il s'agit d'éliminer les informations superflues afin de minimiser la taille du fichier JSON. Plusieurs questions se posent : Qu'appelle-t-on information superflue ? Comment les retirer ? Comment implémenter cela ?

En analysant la structure du plugin je me suis rendu compte de quelque chose de problématique : il se peut qu'une information superflue soit changer en non superflue lors de la phase de l'interface par l'utilisateur. Et cela pose une difficulté car le JSON existe en réalité deux fois. Je m'explique, au chargement du plugin, les informations sont mises telles quelles dans le tableau, cependant, le plugin fait une copie de ce fichier qui est modifié par l'utilisateur. Le problème vient du moment où les modifications sont faites : elles sont faites au moment où on appuie sur le bouton Générer. Pour supprimer les informations superflues, il faut donc agir sur la copie. Seulement ceci se révéla plus complexe que je ne le pensais. Afin de réaliser ces changements précis, j'ai dû recourir à un parcours brut de la copie et tester chaque champ, ce qui, en termes d'exécution, n'est pas très élégant. Mais cette technique fonctionne et permet en même temps de résoudre le besoin de pouvoir modifier la nouvelle propriété que Quentin m'a demandé d'implémenter : indiquer si un champ est modifiable ou non et ce pour des raisons de sécurité de la web application. Ainsi, modifier la copie fut compliqué. Il a fallu déconstruire et reconstruire ce pseudo fichier, ce qui n'est pas une mince à faire.

Voici le code qui n'est pas très élégant, mais qui fonctionne très bien malgré tout.

```

def set_fields_display(self, fields_display):
    for field in fields_display:
        matching_field = self.field(field["field_name"])
        test = matching_field.get("options")
        if test:
            matching_field["options"].update(dict(disabled=field["disabled"], hidden=field["hidden"]))
        if field["notEnterable"] == True or field["hidden"] == True or field["disabled"] == True:
            matching_field["notEnterable"] = field["notEnterable"]
            for i in range(len(self.form["inputText"])):
                if self.form["inputText"][i]["name"] == matching_field["name"]:
                    del self.form["inputText"][i]
                    break
            for i in range(len(self.form["textArea"])):
                if self.form["textArea"][i]["name"] == matching_field["name"]:
                    del self.form["textArea"][i]
                    break
            for i in range(len(self.form["inputDate"])):
                if self.form["inputDate"][i]["name"] == matching_field["name"]:
                    del self.form["inputDate"][i]
                    break
            for i in range(len(self.form["inputRangeSpinBox"])):
                if self.form["inputRangeSpinBox"][i]["name"] == matching_field["name"]:
                    del self.form["inputRangeSpinBox"][i]
                    break
            for i in range(len(self.form["inputRangeSlider"])):
                if self.form["inputRangeSlider"][i]["name"] == matching_field["name"]:
                    del self.form["inputRangeSlider"][i]
                    break
            for i in range(len(self.form["selectBox"])):
                if self.form["selectBox"][i]["name"] == matching_field["name"]:
                    del self.form["selectBox"][i]
                    break
            for i in range(len(self.form["checkBox"])):
                if self.form["checkBox"][i]["name"] == matching_field["name"]:
                    del self.form["checkBox"][i]
                    break
            for i in range(len(self.form["thumbnail"])):
                if self.form["thumbnail"][i]["name"] == matching_field["name"]:
                    del self.form["thumbnail"][i]
                    break
            for i in range(len(self.form["fileInput"])):
                if self.form["fileInput"][i]["name"] == matching_field["name"]:
                    del self.form["fileInput"][i]
                    break
            self.add_config(matching_field)
            for i in range(len(self.form["inputText"])):
                del self.form["inputText"][i]["notEnterable"]
            for i in range(len(self.form["textArea"])):
                del self.form["textArea"][i]["notEnterable"]
            for i in range(len(self.form["inputDate"])):
                del self.form["inputDate"][i]["notEnterable"]
            for i in range(len(self.form["inputRangeSpinBox"])):
                del self.form["inputRangeSpinBox"][i]["notEnterable"]
            for i in range(len(self.form["inputRangeSlider"])):
                del self.form["inputRangeSlider"][i]["notEnterable"]
            for i in range(len(self.form["selectBox"])):
                del self.form["selectBox"][i]["notEnterable"]
            for i in range(len(self.form["checkBox"])):
                del self.form["checkBox"][i]["notEnterable"]
            for i in range(len(self.form["thumbnail"])):
                del self.form["thumbnail"][i]["notEnterable"]
            for i in range(len(self.form["fileInput"])):
                del self.form["fileInput"][i]["notEnterable"]
            for i in range(len(self.form["notEnterable"])):
                del self.form["notEnterable"][i]["notEnterable"]

```

Figure 9 : Code maladroit

Après réflexion avec mon maitre de stage et lui avoir expliqué que malheureusement c'est la structure même du plugin qui oblige cela, il a été convenu de conserver cette solution.

Une longue phase de test avec Quentin a permis de décider que le plugin convenait aux besoins de ce dernier.

Ayant rempli le cahier des charges, j'ai déposé le plugin sur le dépôt GitHub officiel du département, il est désormais fonctionnel. Voici un exemple de fichier JSON généré avec le plugin, conforme aux attentes :

```
{
  "server": {
    "host": "",
    "queryParams": []
  },
  "header": {
    "brand": "",
    "modals": []
  },
  "form": {
    "inputText": [
      {
        "name": "wikipedia",
        "alias": "wiki1",
        "type": "TextEdit",
        "layer": "limad_communes",
        "options": {
          "IsMultiline": false,
          "UseHtml": false,
          "disabled": false,
          "hidden": false,
          "required": false
        }
      },
      {
        "name": "surf_ha",
        "alias": "surf1",
        "type": "TextEdit",
        "layer": "limad_communes",
        "options": {
          "IsMultiline": false,
          "UseHtml": false,
          "disabled": false,
          "hidden": false,
          "required": false
        }
      },
      {
        "name": "insee",
        "alias": "insee2",
        "type": "TextEdit",
        "layer": "limad_communes_grandest",
        "options": {
          "IsMultiline": false,
          "UseHtml": false,
          "disabled": false,
          "hidden": false,
          "required": false
        }
      },
      {
        "name": "nom",
        "alias": "nom2",
        "type": "TextEdit",
        "layer": "limad_communes_grandest",
        "options": {
          "IsMultiline": false,
          "UseHtml": false,
          "disabled": false,
          "hidden": false,
          "required": false
        }
      },
      {
        "name": "wikipedia",
        "alias": "wiki2",
        "type": "TextEdit",
        "layer": "limad_communes_grandest",
        "options": {
          "IsMultiline": false,
          "UseHtml": false,

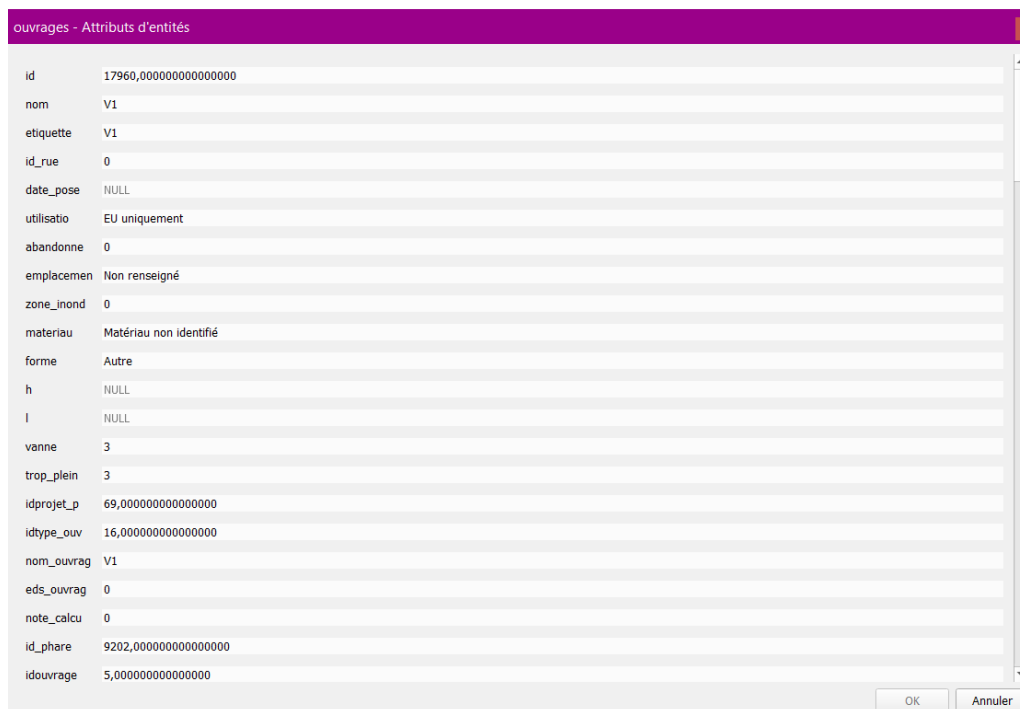
```

Figure 10 : Exemple de fichier JSON généré avec le plugin

Maintenant que la première mission est réalisée, il est temps de passer à la seconde.

2.2.2 Deuxième mission : Implémentation d'une solution pour rendre les formulaires plus ergonomiques

Pour ce qui est de ma deuxième mission, c'est davantage un exercice de communication. Pour réaliser cette dernière, je devais donc trouver un moyen de modifier ces formulaires. Pour cela, en cherchant un peu dans QGIS, j'ai remarqué que l'on pouvait passer en paramètre un fichier .ui afin de modifier le formulaire. Voici un exemple de formulaire de base sur QGIS :



The screenshot shows a QGIS dialog box titled 'ouvrages - Attributs d'entités'. It contains a form with the following fields and values:

Field	Value
id	17960,0000000000000000
nom	V1
etiquette	V1
id_rue	0
date_pose	NULL
utilisatio	EU uniquement
abandonne	0
emplacemen	Non renseigné
zone_inond	0
matériau	Matériau non identifié
forme	Autre
h	NULL
l	NULL
vanne	3
trop_plein	3
idprojet_p	69,0000000000000000
idtype_ouv	16,0000000000000000
nom_ouvrag	V1
eds_ouvrag	0
note_calcu	0
id_phare	9202,0000000000000000
idouvrage	5,0000000000000000

At the bottom right, there are 'OK' and 'Annuler' buttons.

Figure 11 : Formulaire de base de QGIS

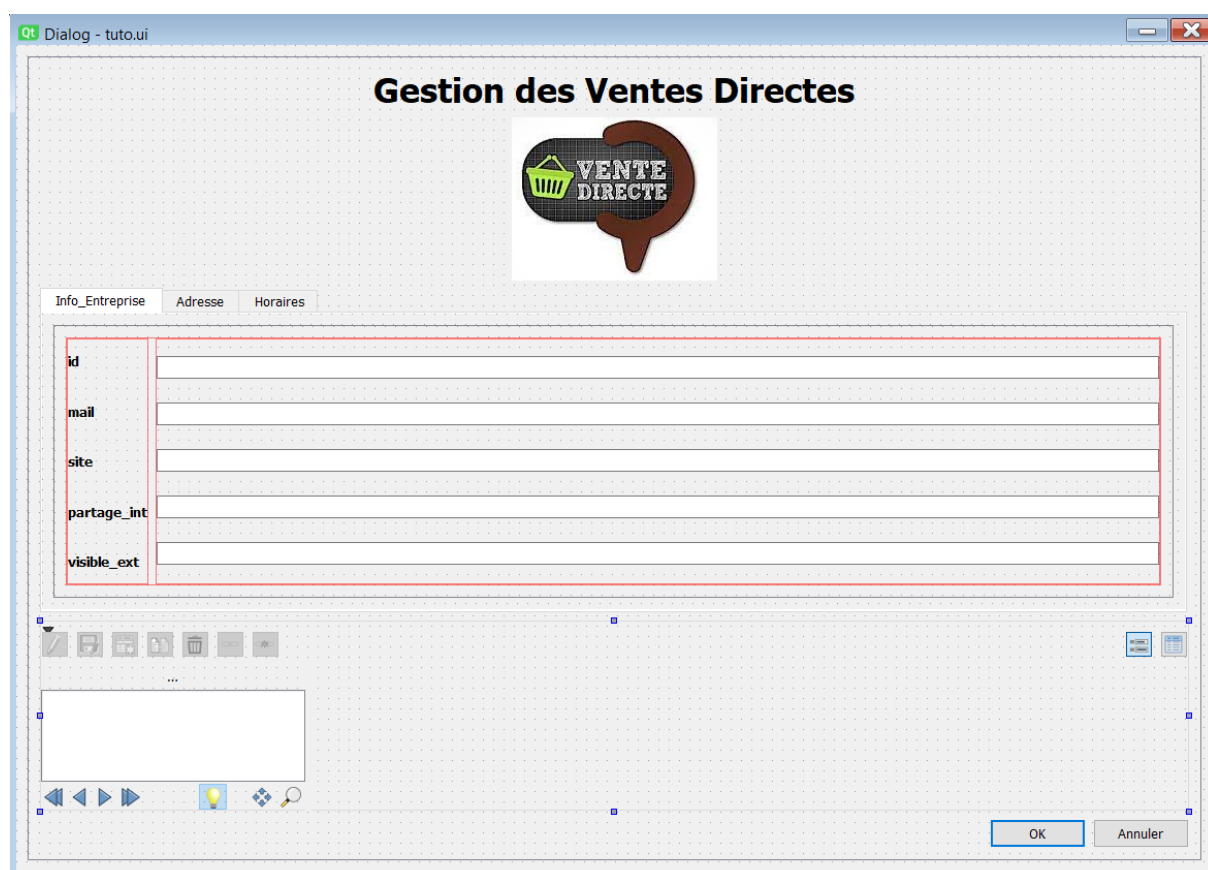
J'ai donc cherché un moyen de créer simplement ces fichiers .ui. Plusieurs solutions se sont présentées à moi, cependant, je devais effectuer une certaine réflexion quant au choix de l'outil. Il fallait qu'il soit simple, ergonomique et qui ne dérangerait pas les utilisateurs de QGIS. Le logiciel intégré au pack Qt, QtDesigner, dont QGIS fait parti, et que j'ai utilisé pour modifier le formulaire de la première mission, me semblait être la meilleure option pour plusieurs raisons :

- La première étant l'environnement de travail. Les soixante utilisateurs de QGIS ont déjà le logiciel installé, ainsi plus simple pour eux.
- QtDesigner faisant partie du pack Qt, il est donc 100% compatible avec QGIS.
- Après utilisation par moi-même et quelques tests, il me semblait être adéquat pour la tâche.

Après cette étude, j'avais acté mon choix : QtDesigner serait la solution que je proposerais.

C'est à ce moment qu'arrive l'exercice de communication : je devais prendre le rôle d'un consultant informatique et être capable de présenter un logiciel qui répondrai aux besoins de soixante personnes. N'ayant jamais fait ce genre de travail auparavant, j'ai donc fait des recherches sur comment présenter un logiciel. Le contexte sanitaire ne me le permettant pas, je devais donc faire une présentation à distance. J'ai donc choisi de réaliser la présentation du logiciel de manière ludique et démonstratif : j'ai réalisé un Power Point qui présente le logiciel en tutoriel pour écrire leur premier formulaire personnalisé. Faire cet exercice demande des techniques de communications différentes de ce à quoi je suis habitué, et bien sûr me demande de connaître l'outil proposé de manière poussée afin de l'expliquer au mieux aux autres personnes.

Dans cette optique, j'ai donc effectué plusieurs formulaires personnalisés dont voici le résultat :



The image shows a Qt Designer window titled "Qt Dialog - tuto.ui". The main area contains a form titled "Gestion des Ventes Directes" with a logo that says "VENTE DIRECTE". Below the title, there are three tabs: "Info_Entreprise", "Adresse", and "Horaires". The "Info_Entreprise" tab is selected, showing a form with the following fields: "id", "mail", "site", "partage_int", and "visible_ext". Each field has a corresponding text input box. At the bottom of the form, there are "OK" and "Annuler" buttons. The Qt Designer interface includes a toolbar with various icons for design and a palette on the right side.

Figure 12 : Formulaire personnalisé fait avec QtDesigner

Le Power Point se présente donc sous la manière d’un tutoriel ludique qui donne envie aux lecteurs d’aller au bout de l’outil et de l’utiliser. J’ai dû adapter mon vocabulaire et expliquer dans les moindres détails ce qui pourrait porter à confusion. J’ai demandé des retours fréquents de la part de Mr Vitoux sur comment je devrais m’y prendre ou ce dont les utilisateurs auraient besoin. Cela m’a permis d’ajuster la présentation au mieux. Cependant, la création du formulaire n’a pas été exempté de problèmes.

Un problème auquel j’ai fait face fut lorsque mon maitre de stage m’expliqua les relations de dépendances entre les bases de données. En effet, certaines bases de données sont liées entre elles et ont donc des valeurs communes. Cela fut un problème dans le sens où QtDesigner propose un outil pour gérer cela, malheureusement c’est le même que QGIS qui ne convenait pas à Mr Vitoux car pas assez ergonomique. J’ai fini par trouver une solution, mais cela demande la modification du code du formulaire, chose qui pourrait rebuter les référents QGIS.

Pour rester dans mon optique de simplicité, après discussion de la situation avec mon maitre de stage, j’ai donc renoncé à cette solution et gardé la solution proposée par QtDesigner. Cela m’a appris que lorsque l’on présente un nouveau logiciel à des personnes, il faut éviter de compliquer la tâche car certains peuvent être réticents à faire des tâches complexes.

Malheureusement la situation sanitaire et le temps du stage ne m’ont pas permis d’avoir un retour sur ma présentation de la part des référents QGIS.

Cela n’enlève en rien le fait que c’était un bon exercice que d’apprendre à présenter un nouveau logiciel à un grand groupe de personnes. Cela démontre le coté plus social du travail de l’informaticien que l’on a tendance un peu à négliger.



Figure 13 : Diaporama de la présentation de QtDesigner

Il me reste désormais une troisième et dernière mission que j’ai réalisé vers la fin de mon stage.

2.2.3 Troisième mission : Développement d'un plugin de correction orthographique de base de données

Cette fois-ci, il s'agit de développer un plugin QGIS du début à la fin. Ayant maintenant l'expérience du premier plugin, j'ai pu gagner énormément de temps sur la création du squelette. Une fois ce dernier fait, j'ai réitéré le processus utilisé pour le premier plugin : conceptualisation, codage, test. Cette mission n'a pas avancé aussi vite que les autres mais j'y est beaucoup réfléchi avec mon maître de stage en amont, ce qui m'a permis d'avoir une idée globale de comment allait fonctionner le plugin. D'autant plus qu'avec l'expérience acquise avec le premier plugin et QGIS, je visualisais bien comment m'orienter.

La phase de conceptualisation est ce qui a pris le plus de temps. Bien penser son application est la clé pour la réussir. Mes premières recherches se sont portées sur un potentiel existant du projet. Il s'est avéré encore une fois que non, il n'y avait pas d'existant au projet. Cependant, la correction orthographique est un domaine largement discuté sur internet et parmi la communauté des informaticiens. Étant un sujet relativement complexe, j'ai décidé de lire un mémoire sur le sujet d'une personne ayant réalisé son propre système de correction orthographique. J'ai pu apprendre différentes techniques pour mener à bien mon projet.

Rappelons le but : il arrive que dans les bases de données, par faute de convention ou par importation de bases de données externes, il y ait des coquilles ou des fautes d'orthographe. Le cahier des charges est défini tel que suit :

- Le plugin doit permettre de corriger les fautes d'orthographe.
- Il doit être simple d'utilisation et ne doit pas être trop complexe.

Il faut donc une technique qui permette de corriger des fautes d'orthographe, mais qui soit simple d'utilisation.

Au début je n'avais aucune idée de comment faire. Cependant, le mémoire que j'ai lu m'a donnée des indices sur la démarche à suivre.

Premièrement, il est fait mention qu'il est très facile de tomber dans l'intelligence artificielle lorsqu'il s'agit de correcteur orthographique. C'est en effet ce que propose Google qui est très actif sur le sujet. Certains correcteurs orthographiques utilisent l'intelligence artificielle pour prendre en compte l'avant et l'après d'un mot, certains prennent même en compte le contexte. C'est ce qui se passe par exemple lorsque l'on tape une phrase dans la barre de recherche de Google, s'il y a une faute, il le corrigera automatiquement lorsqu'il va chercher les résultats en gardant en même temps les résultats de ce qui est vraiment tapé. Je n'ai malheureusement pas les capacités de Google, il fallait donc me tourner vers une autre solution. Il me fallait être plus simple. C'est en parcourant le mémoire que j'ai retenu deux choses qui pourrait potentiellement marcher :

- L'algorithme de Norvig
- L'algorithme de la distance de Levenshtein

Je vais expliquer les tenants et aboutissants de chacun de ces deux algorithmes.

Commençons par celui que je n'ai pas retenu : l'algorithme de Norvig.

En 2007, deux amis de Mr Norvig sont fascinés par la capacité de la barre de recherche Google. Elle est capable de trouver des résultats malgré les fautes d'orthographes présentes. Ils n'arrivaient pas à comprendre comment cela fonctionnait, et pourtant ils étaient ingénieur et mathématicien. Norvig a donc décidé de réaliser un algorithme qui montrerait à ses deux compères comment cela fonctionne. Cet algorithme a plusieurs avantages et inconvénients :

- Sa taille est un avantage, seulement 21 lignes de code en Python
- Son taux de réussite est élevé, près de 90%
- Malheureusement, il a besoin d'une référence bibliographique pour fonctionner.
- Il se base sur des probabilités, ce qui peut être dérangerant.

Pour illustrer comment il fonctionne, utilisons un exemple :

Je veux corriger le mot anglais « speling » en « spelling ». En le passant en paramètre de cet algorithme il en ressort le mot corrigé.

```
>>> correction('speling')
'spelling'

>>> correction('korrektud')
'corrected'
```

Figure 14 : Démonstration de l'algorithme de Norvig

L'algorithme se base fortement sur des probabilités. Il essaie de choisir la correction orthographique la plus probable pour le mot passé en paramètre. Pour cela il utilise des probabilités mathématiques et un fichier .txt contenant de nombreuses références littéraires de Shakespeare afin de trouver le meilleur résultat.

Pourquoi je n'ai pas retenu cette solution bien que prometteuse ? Il y a un inconvénient majeur ici : l'utilisation d'un fichier externe pour exécuter le programme. En effet, on veut un algorithme simple et rapide qui ne nécessite pas de ressources externes.

Il faudra donc se pencher sur l'algorithme de la Distance de Levenshtein.

Selon Wikipédia, « *La Distance de Levenshtein est une distance, au sens mathématique du terme, donnant une mesure de la différence entre deux chaînes de caractères. Elle est égale au nombre minimal de caractères qu'il faut supprimer, insérer ou modifier pour passer d'une chaîne à l'autre.* »

Cet algorithme est petit, relativement rapide et est complètement autonome, il ne requiert pas de ressources externes. Voici un exemple d'utilisation :

Nous avons deux chaînes de caractères : « oui » et « ouii » respectivement « a » et « b ». Si nous faisons passer ces deux chaînes dans l'algorithme de Levenshtein, $lev(a,b)$, nous obtenons 1 car il y a une modification à faire pour atteindre « oui » depuis « ouii ».

Cet algorithme semble parfait pour répondre à mon problème. Il reste cependant à prendre avec des pincettes, car encore une fois, il joue sur des probabilités.

Pour remédier à ce problème potentiel, il suffit de faire intervenir l'utilisateur. J'ai décomposé l'algorithme en trois étapes distinctes :

- 1) Demandes utilisateur
- 2) Algorithme de Levenshtein
- 3) Modification dans la base

En premier lieu, lorsque l'on lance le plugin, il demande trois choses à l'utilisateur :

- Le nom de la colonne qu'il veut changer
- Le numéro de colonne qu'il veut changer
- Le mot original avant modification

Avec ces informations, le plugin va se placer dans la colonne demandée, la parcourir et effectuer l'algorithme de Levenshtein entre la chaîne passée en paramètre par l'utilisateur puis en fonction de ce que retourne l'algorithme, corrige ou non le mot interrogé.

```
def levenshtein(self, mot1, mot2):
    ligne_i = [k for k in range(len(mot1) + 1)]
    for i in range(1, len(mot2) + 1):
        ligne_prec = ligne_i
        ligne_i = [i] * (len(mot1) + 1)
        for k in range(1, len(ligne_i)):
            cout = int(mot1[k - 1] != mot2[i - 1])
            ligne_i[k] = min(
                ligne_i[k - 1] + 1, ligne_prec[k] + 1, ligne_prec[k - 1] + cout
            )
    return ligne_i[len(mot1)]
```

Figure 15 : Algorithme de Levenshtein en Python

L'algorithme utilise les matrices mathématiques afin de déterminer cette distance. Son fonctionnement sera disponible en annexe.

Avec ce mécanisme il est très simple de corriger des petites fautes, en fonction de l'écart que l'on définit dans le code. Il faut tout de même rester critique vis-à-vis de cet algorithme. En effet il convient bien ici au besoin pour corriger des petites fautes, mais est inadéquat dès qu'il s'agit de corriger des mots entiers ou des mots avec plus de fautes.

J'ai cependant dressé une liste d'améliorations possibles pour plus tard car je n'ai pas eu le temps nécessaire à cela. Cette liste se trouve dans le fichier README du dépôt GitHub.

J'y fais mention de laisser l'utilisateur choisir l'intervalle de l'algorithme de Levenshtein.

3. Conclusion

J'ai effectué mon stage au sein du Conseil Départemental de Meurthe-et-Moselle. Mes missions étaient principalement de répondre aux besoins des utilisateurs du logiciel QGIS.

J'ai beaucoup apprécié travailler au sein de cet environnement, j'ai découvert des facettes de l'informatique que je ne connaissais pas. J'ai eu beaucoup de liberté quant à mes choix de développement et toute l'équipe SIG m'a bien encadré pendant ce stage. J'ai pu expérimenter le travail d'équipe en situation réel, en communiquant régulièrement avec mon collègue Quentin Malavielle.

J'ai appris un nouveau langage de programmation ainsi que l'utilisation du logiciel QGIS, ce qui est un plus pour mes compétences personnelles ainsi que ma culture informatique.

Pour finir, mes missions ont été remplies, les plugins marchent et remplissent leur cahier des charges, et seront bientôt utilisés par le conseil départemental.

Ce stage m'a permis de découvrir le monde professionnel, de travailler avec une équipe sur des projets concrets qui servent un objectif plus grand, et qui assiste plusieurs personnes. Il m'a aussi permis de concrétiser mon projet professionnel, car les systèmes d'informations m'ont vraiment plu et intrigué, ce pourquoi je veux me diriger vers l'école Miage, où les systèmes d'informations sont leur spécialité. Je pourrais donc y développer mes compétences en lien avec ces derniers.

Bibliographie

- Article Wikipédia sur l'algorithme de Levenshtein :

https://fr.wikipedia.org/wiki/Distance_de_Levenshtein

- Documentation QGIS :

<https://www.qgis.org/fr/docs/index.html>

- Article de Mr Norvig sur son algorithme :

<https://norvig.com/spell-correct.html>

- Lien du site du conseil départemental :

<http://www.meurthe-et-moselle.fr/>

- Documentation QtDesigner :

<https://doc.qt.io/qt-5/qtdesigner-manual.html>

- Mémoire de Augustin Borsu : Implémentation d'un correcteur orthographique automatique pour l'extraction d'information

https://www.academia.edu/7434276/Impl%C3%A9mentation_dun_correcteur_orthographique_automatique_pour_lextraction_dinformation

Annexe

Fonctionnement de l'algorithme de Levenshtein :

Fonctionnement [modifier | modifier le code]

Soit n la longueur de la chaîne1 (ici 'NICHE', $n=5$).

Soit m la longueur de la chaîne2 (ici 'CHIENS', $m=6$).

Si $n=0$ alors retourner $d=m$ et quitter.

Si $m=0$ alors retourner $d=n$ et quitter.

Construire une matrice D de $n+1$ lignes et $m+1$ colonnes.

Initialiser la première ligne par la matrice ligne $[0, 1, \dots, m-1, m]$ et la première colonne par la matrice colonne $[0, 1, \dots, n-1, n]$.

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	1	0	0	0	0	0	0
I	2	0	0	0	0	0	0
C	3	0	0	0	0	0	0
H	4	0	0	0	0	0	0
E	5	0	0	0	0	0	0

Soit $\text{Cout}[i, j] = 0$ si $a[i] = b[j]$ et $\text{Cout}[i, j] = 1$ si $a[i] \neq b[j]$. C'est le coût de substitution.

On a donc ici la matrice Cout:

	C	H	I	E	N	S
N	1	1	1	1	0	1
I	1	1	0	1	1	1
C	0	1	1	1	1	1
H	1	0	1	1	1	1
E	1	1	1	0	1	1

On remplit ensuite la matrice D en utilisant la règle suivante : $D[i, j]$ est égale au minimum entre les éléments suivants :

- L'élément directement au-dessus et on ajoute 1 (effacement) : $D[i-1, j] + 1$.
- L'élément directement avant et on ajoute 1 (insertion) : $D[i, j-1] + 1$.
- L'élément diagonal précédent et on ajoute le coût (substitution) : $D[i-1, j-1] + \text{Cout}[i-1, j-1]$.

Attention ! Il s'agit de $\text{Cout}[i-1, j-1]$ et non de $\text{Cout}[i, j]$ car la matrice Cout est moins grande que la matrice D , ce qui entraîne un décalage.

Dans notre cas, le remplissage de la première ligne donne alors :

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	1	1	1	2	3	4	5
I	2	0	0	0	0	0	0
C	3	0	0	0	0	0	0
H	4	0	0	0	0	0	0
E	5	0	0	0	0	0	0

Nous réitérons cette opération jusqu'à remplir la matrice :

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	1	1	1	2	3	4	5
I	2	2	2	2	3	4	5
C	3	2	3	3	3	4	5
H	4	3	2	3	4	4	5
E	5	4	3	3	3	4	5

La distance de Levenshtein entre les 2 chaînes est en $D[n, m]$, c'est-à-dire en bas à droite.

Ici, on retrouve bien les 5 opérations trouvées de manière intuitive.

La matrice D fournit aussi explicitement les suites d'opérations possibles permettant de passer d'une chaîne de caractères à l'autre (il existe ici 6 suites possibles). On part du 5 en bas à droite et on examine les trois cases du quartier supérieur gauche ; (4,4,5) dans le sens d'une montre. On suit la ou les valeurs minimums. Ici on a deux 4 qui donnent donc deux chemins (une bifurcation), et ainsi de suite on crée un arbre par récurrence. Aller au dessus s'interprète comme *détruire la lettre de la colonne*; aller en diagonale est *substituer la lettre de la ligne par la lettre de la colonne* (ou pas si elles sont égales) ; aller à gauche est *ajouter la lettre de la colonne*. Bien sûr à la fin, il faut lire les étapes à l'envers.

Annexe des figures

Figure 1 : Logo du conseil départemental

Figure 2 : Organigramme général de la structure du CD

Figure 3 : Organigramme de la branche développement

Figure 4 : Diagramme de Gantt

Figure 5 : Carte du département réalisé à l'aide de QGIS

Figure 6 : Cas d'utilisation du plugin de Hugo réalisé avec Draw.io

Figure 7 : Image du plugin après modification

Figure 8 : Code correspondant aux vérifications des cas de figures

Figure 9 : Code maladroit

Figure 10 : Exemple de fichier JSON généré avec le plugin

Figure 11 : Formulaire de base de QGIS

Figure 12 : Formulaire personnalisé fait avec QtDesigner

Figure 13 : Diaporama de la présentation de QtDesigner

Figure 14 : Démonstration de l'algorithme de Norvig

Figure 15 : Algorithme de Levenshtein en Python

FICHE RAPPORT DESTINEE A LA BIBLIOTHEQUE

(à insérer à la fin du rapport)

RAPPORT CONFIDENTIEL ET NE DEVANT PAS FIGURER A LA BIBLIOTHEQUE :

☒ Oui

NOM ET PRENOM DE L'ETUDIANT :

Drouet Rodolphe

DUT : INFORMATIQUE

☒ S4

☐ Année Spéciale

LICENCE PROFESSIONNELLE

☐ ASRALL ☐ CIASIE

TITRE DU RAPPORT : Réalisation de plugins pour le système d'information QGIS

Nom de l'Entreprise : Conseil départemental de Meurthe-et-Moselle

Adresse : 48 Esplanade Jacques Baudot, 54000 Nancy

Type d'activité (domaines couverts par l'entreprise) : Action sociale, éducation, transports, aménagement du territoire, culture, tourisme, sport et environnement

Nom du parrain (enseignant IUT) : Thomas Giletti

Mots-clés (sujets traités) : QGIS, Plugin, Informatique, Python, JSON

Résumé

Développement de plugins QGIS en langage Python pour les utilisateurs du logiciel. L'un permet de générer un fichier JSON, l'autre permet de corriger des fautes d'orthographe.