Rapport de stage

Développement d'un outil de création de bases de données géoréférencées

Stage final de DUT à l'IFSTTAR de Nantes

IFSTTAR

IUT Informatique de Nantes

Réalisation	Rémi TAUNAY
Encadrement	Pierre HANKACH & Pascal GASTINEAU
Suivi	Loïg JEZEQUEL

10 avril au 16 juin 2017

Remerciements

Je tiens tout d'abord à remercier mes tuteurs de stage, Pierre Hankach et Pascal Gastineau, pour m'avoir offert cette possibilité de travailler dans le domaine du traitement de données et des systèmes d'informations géographiques. Je tiens à leur exprimer ma gratitude pour leur écoute, le partage de leur expérience, mais aussi et surtout pour la confiance qu'ils m'ont accordée tout au long du stage.

Je souhaite également remercier Marc Noël, membre de l'équipe projet avec qui j'ai partagé le même bureau. Il m'a transmis son expérience professionnelle et m'a introduit aux systèmes d'informations géographiques.

De plus, j'aimerais remercier l'ensemble des personnes que j'ai pu cotôyer au sein de l'IFSTTAR pour leur accueil.

Enfin, je souhaite remercier Loïg Jezequel pour son suivi, et saluer de façon plus large l'ensemble des enseignants de l'IUT Informatique de Nantes pour m'avoir transmis avec passion leurs savoirs au cours de mes deux années de DUT .

Les mots suivi d'un astérisque (*)
sont explicités dans le glossaire.

Résumé

Le travail présenté dans ce rapport a trait à la mise en œuvre d'une application qui permet de créer et d'alimenter automatiquement une base de données géoréférencées. En effet, plusieurs problématiques traitées à l'IFSTTAR nécessitent d'avoir à disposition des données, souvent volumineuses, relatives au territoire. Ainsi, il est fréquemment nécessaire de se munir d'une base de données géoréférencées. Avant les travaux de ce stage, le processus de manipulation des données exploitées était rudimentaire. En effet, le stockage s'effectuait directement dans un système de fichiers. Également, les utilisateurs procédaient manuellement pour récupérer et transformer ces données, ce qui était laborieux et très coûteux en termes de temps.

Afin d'apporter une solution à ces problématiques, ce stage consiste à réaliser un ensemble d'outils auxquels seront déléguées ces tâches de récupération, de transformation et de stockage dans un serveur centralisé de données. Grâce à notre application, cette opération doit être aisée et reproductible ; c'est-à-dire pouvoir recréer et alimenter, à partir de zéro et sur commande, un serveur avec les données souhaitées. L'utilisateur final sera ainsi libéré des manipulations de pré-traitement, et pourra se concentrer davantage sur l'exploitation des données.

L'application a été développée en python. Elle consiste tout d'abord à lire un fichier récapitulant les adresses des sources de données voulues. En se basant sur ce fichier, les données sont récupérées, décompressées et explorées. Ensuite, une étape de pré-traitement est réalisée, qui consiste à remodeler ces données en fonction de paramètres fournis par l'utilisateur dans un fichier de configuration. Enfin, ces données sont importées dans le système de gestion de base de données PostgreSQL muni de l'extension géographique PostGIS. En outre, plusieurs scripts systèmes ont été développés pour installer et configurer automatiquement le serveur de données.

Abstract

The work exposed in this internship report concerns the production of an application. This application allows to create and fill automatically a database with geographic data. Several issues handled by IFSTTAR needs having an access to datasets, usually huge ones, which are realtive to the territory. That is why it is necessary to possess a georeferenced database. Before this internship's work, the manipulating process of collected data was rudimentary. In fact, the storage was directly handled by the filesystem. Moreover, users were collecting and processing manually the data sets. This was laborious and really heavy, in terms of spent time.

In order to bring a solution that can answer these issues, this internship consists in the production of a toolbox. The retrieving, transforming, and storage tasks in a centralized data platform will be delegated to these tools. Thanks to our application, this operation must be simple and reproducible. More precisely, it means to have the aptitude of re-creating and fill, from scratch and on demand, a data platform containing all the required data. Therefore, the final user will be released from pre-processing constraints so he can have a better focus on the data analysis.

The application was developed in Python. First of all, the application consists in reading the data sets sources specifications, which are compiled in a configuration file. From this point, the data is retrieved, extracted and explored. Then, a pre-processing step is applied. This step allows to remodel the data, taking into consideration the parameters given by the user from a configuration file. At last, these data are imported in a PostgreSQL's database management system, equipped with the PostGIS extension. In addition, several shell scripts have been developed. They are in charge of installing and configuring automatically the data server.

Table des matières

Introduction

Contexte

Dans le cadre de ma formation de DUT , un stage doit être réalisé. D'une durée de dix semaines, il doit permettre de se construire une première expérience du monde du travail en informatique. Il vise également à constituer l'occasion de mettre en pratique les connaissances accumulées tout au long de la formation.

Objectif du stage

Au sein de l'IFSTTAR* sont menées des recherches traitant de problématiques sur l'aménagement du territoire et le transport. Ces recherches nécessitent l'utilisation de données socio-économiques géoréférencées. À l'heure actuelle les tâches de récupération, pré-traitement, stockage et filtrage effectuées sur les données sont réalisées manuellement.

De ce fait, ce stage a pour but de venir répondre à cette problématique avec la réalisation d'un outil de création d'une base de données géoréférencées. De façon plus générique, le développement de cette application a donc trait à l'exploitation de données. Cette application se positionne donc en amont des traitements, ici réalisés avec des logiciels de système d'information géographique.SIG*.

Le programme réalisé devra donc être en mesure de se charger de la récupération, du remodelage et de l'importation en base de données de données géographiques.

Travail réalisé

Recueil du besoin

J'ai tout d'abord recueilli le besoin afin de bien comprendre les tenants et aboutissants du projet :

- les tâches à automatiser, c'est-à-dire la portée du sujet ;
- le profil d'un utilisateur de l'application ;
- les entrées utilisateur (quelles informations sont renseignées, lesquelles peuvent être omises) ;
- les éventuelles évolutions de l'applicatif à plus long terme.

Conception du fichier de configuration des imports

Nous avons conçu un fichier de configuration permettant de préciser exactement le comportement du programme qui est attendu par l'utilisateur sur cette donnée. Ce fichier, en format JSON, a le rôle d'une interface de communication avec l'utilisateur. C'est par le biais de ce fichier que l'utilisateur peut décider de quelle manière doit être remodelée une donnée, mais également de quelle manière elle doit être importée dans la base de données.

Modules de l'applicatif principal

L'application est découpée en modules. Ce découpage est le résultat de l'isolation des différentes tâches à effectuer par le programme. En effet, ces tâches sont effectuées séquentiellement. On dénombre quatres modules.

data_retrieving : téléchargement et extraction des données ;

data_processing : recherche, parsage, remodelage des données ;

database : construction des requêtes et importation en base de données ;

utils : fonctions utilitaires, utilisées indifféremment par tous les modules.

Scripts

En parallèle du développement principal, j'ai réalisé un ensemble de scripts :

prepare_database (shell) : installe et configure une base de données PostgreSQL ;

prepare_environment (shell) : prépare l'environnement d'exécution ;

create_venv (shell) : crée le fichier décrivant l'environnement virtuel conda* ;

excel2conf (Python) : convertit un fichier excel spécifique en un fichier JSON de configuration.

Technologies manipulées

J'ai été amené à utiliser le langage de script bash afin de réaliser les scripts d'installation et de configuration. Je me suis également familiarisé avec le SGBD* PostgreSQL et l'extension PostGIS. La réalisation du programme principal a été effectuée en Python.

J'ai pour vœu de rapprocher, à terme, mon profil de celui d'un Data Scientist. En ce sens, ce stage constitue pour moi une expérience formidable. En effet, les langages tels que bash et Python sont assez répandus en science des données.

Chapitre 1

Contextualisation du stage

1.1 Présentation de l'IFSTTAR

1.1.1 L'établissement

J'ai effectué mon stage à l'Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux (IFSTTAR). C'est un "Établissement Public à caractère Scientifique et Technologique." (EPST*). Créé en 2011, il est placé sous la double tutelle du Ministère de l'Écologie, du Développement durable et de l'Énergie et du Ministère de l'Enseignement supérieur et de la Recherche.

Au total, l'IFSTTAR rassemble environ un millier d'agents, dont 350 chercheurs. Il y a environ 380 doctorants accueillis dans les laboratoires de l'IFSTTAR . L'IFSTTAR comporte 5 départements de recherche distincts (voir annexe A). Par ailleurs, l'IFSTTAR dispose d'un large patrimoine d'équipements scientifiques, qui lui permet de développer une recherche et une expertise de haut niveau. En effet, il s'agit d'équipements rares qui permettent à l'IFSTTAR de conduire des travaux de recherche. Ces équipements scientifiques sont très diversifiés : des sites et plateformes d'expérimentation, des simulateurs, des véhicules instrumentés, ou encore des recueils de données. Une importante production scientifique (thèses, publications, rapports de recherche) découle de ces équipements indispensables aux structures de recherche pour la mise en œuvre de leurs travaux.

1.1.2 Le site de Bouguenais

Il y a 12 laboratoires représentant 4 départements sur le site de Bouguenais.

1.2 Environnement et outils de travail

1.2.1 Laboratoire d'accueil

J'ai été accueilli au sein du laboratoire "Environnement, Aménagement, Sécurité et Eco-conception"(EASE*) du département "Aménagement, mobilités et environnement" (AME*). Au sein du bâtiment, j'ai également été en contact avec des personnes du laboratoire (LAMES*) du département "Matériaux et structures".

1.2.2 Ressources matérielles et logicielles à disposition

J'avais bien entendu un bureau à disposition. J'avais également à disposition un tableau blanc qui m'a beaucoup servi pour les activités de conception, ainsi que pour réfléchir sur les problématiques qui ont pu se poser au long du projet.

FIGURE 1.1 – Répartition des différents sites de l'IFSTTAR

FIGURE 1.2 – Plan d'accès du site de Bouguenais

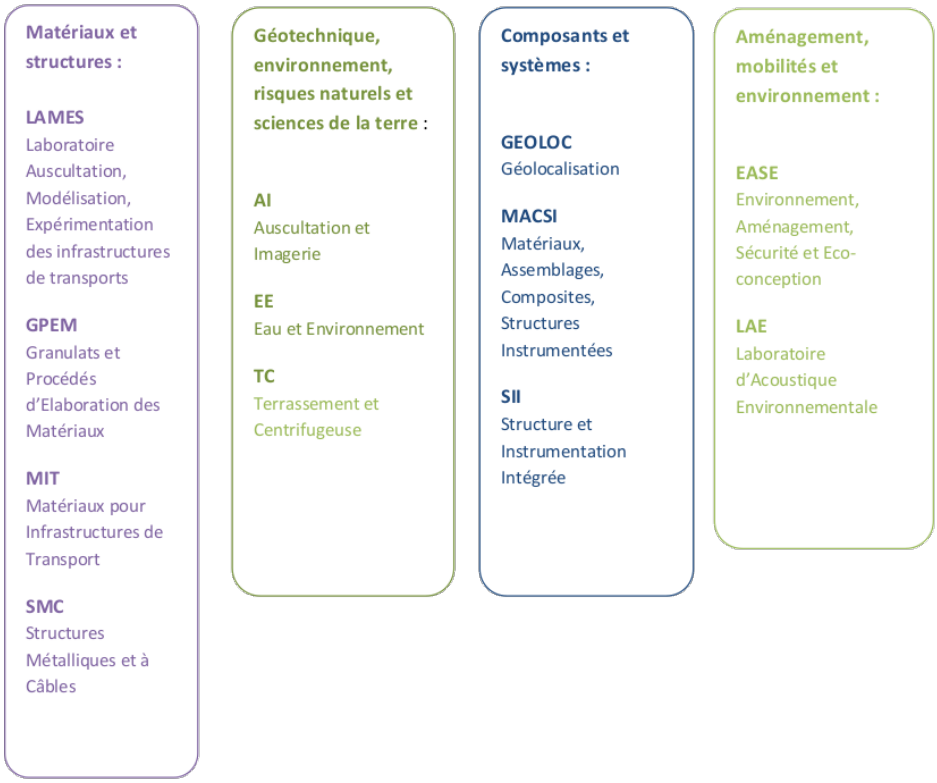


FIGURE 1.3 – Laboratoires présents sur le site de Bouguenais

Pour le développement, j'avais à disposition un ordinateur en dual boot, avec deux systèmes d'exploitation à ma disposition : Windows 7 et Ubuntu 16.04 LTS*. Cependant je n'ai utilisé que Linux.

En ce qui concerne le déploiement de mon application, une machine nous a été rendue disponible en SSH*. Ce serveur a notamment un disque ayant une capacité de stockage nécessaire pour traiter d'importants ensembles de données.

1.2.3 Outils utilisés

Je me suis muni des outils avec lesquels je suis à l'aise. Ils m'ont permis de gérer et de mener à bien mon développement de manière efficace.

PyCharm* : IDE* en version professionnelle (licence étudiante offerte par JetBrains*)

conda* + pip* : facilitateurs de gestion des dépendances via la création d'un environnement virtuel d'exécution

git* + GitHub* : gestionnaire de version

markdown* : langage à la syntaxe enfantine pour les productions rapides de petits documents

LaTeX* : langage de présentation utilisé pour la rédaction du cahier des charges et de ce rapport

Atom* : éditeur de texte, que j'ai surtout utilisé pour le markdown et le LaTeX

SublimeText* : éditeur de texte, utilisé en complément d'Atom

1.2.4 Organisation et méthodologie

L'organisation d'équipe que l'on a suivi reprend plusieurs point importants que l'on peut retrouver en méthode agile :

- des évolution rapides au niveau des spécifications ;
- des réunions assez courtes avec quelques questions par personne et un tableau à disposition ;
- une communication permanente au sein de l'équipe sur la fonctionnalité dont le développement est à prioriser ;
- un développement itératif ;
- une version fonctionnelle de l'application à tous les stades du développement ;
- des tests effectués sur les fonctionnalités au fur et à mesure de l'avancement.

1.2.5 Prise de notes

Tout au long de mon stage, j'ai tenu un journal de bord qui décrit ce que j'ai réalisé au jour le jour. Par ailleurs, j'ai conservé les liens des ressources utiles que j'ai trouvées lors de mes recherches ou que l'on a pu me signaler. Je les ai accompagnés d'une description succincte indiquant la nature des informations intéressantes que l'on peut y trouver.

Chapitre 2

Scripts shell d'installation

2.1 Objectif

Avant la phase de traitement des données vient celle de la préparation de l'infrastructure. La création de la plateforme de données géographiques se traduit par l'installation d'une base de données sur un serveur distant. Dans l'optique d'automatiser ces étapes préparatoires, plusieurs scripts ont été réalisés.

Par ailleurs, si les capacités du serveur sont adaptées à exécuter l'application Python afin de rapatrier les données, il serait néanmoins possible d'exécuter le programme sur une machine client. Cela motive le choix de séparer le script de préparation de la base de données, du script de préparation de l'environnement d'exécution. Ainsi, on peut les lancer indépendamment : le premier sur le serveur de données et le second sur la machine où le code Python s'exécutera. La création de ces deux scripts répond donc à la problématique suivante. Comment effectuer facilement l'ensemble des installations nécessaires, de manière automatisée ?

Enfin, un dernier script, qui n'est pas à destination de l'utilisateur final, à été réalisé. Il permet de générer à nouveau l'environnement virtuel conda.

2.2 Détail des différents scripts

2.2.1 Installation et configuration de PostgreSQL

Ce script, développé en bash, a pour rôle d'installer et de configurer correctement PostgreSQL. C'est le premier script à lancer lors d'une installation à partir de zéro. Il se charge notamment :

- de définir les modes d'authentification ;
- d'autoriser les connexions à distance ;
- d'ajouter la journalisation des connexions et déconnexions ;
- d'ajouter le support de l'UTF-8* par défaut ;
- de créer les extensions ;
- de créer deux rôles (lecture et écriture) ;
- d'assigner un mot de passe à l'utilisateur "postgres" ;
- de créer la base de données, vide.

En ce qui concerne les installations, ce sont de simples appels à la commande `apt-get`. L'ajout des rôles et du mot de passe pour l'utilisateur postgres, la création de la base et des extensions ainsi que le support de l'UTF-8 est assez simple. Ce sont des requêtes SQL soumises à la base à l'aide du client `psql`.

En revanche, la configuration des méthodes d'authentification, l'autorisation des connexions à distance, ainsi que la configuration de la journalisation s'effectuent moins facilement. En effet, il est nécessaire de modifier deux fichiers de configuration dans le dossier `/etc/postgresql/9.6/main` :

`postgresql.conf` : paramètres généraux

`pg_hba.conf` : règles d'authentification

Heureusement, de nombreuses commandes bash existent afin de traiter des données textuelles. Ainsi, l'utilisation de `sed` combinée aux expressions régulières (`glsregex`*) permet d'éditer comme souhaité ces deux fichiers de configuration.

2.2.2 Préparation de l'environnement d'exécution

Ce script, développé en bash, a pour rôle de préparer l'environnement dans lesquels l'applicatif Python s'exécutera. C'est le second script à lancer lors d'une installation à partir de zéro. Il se charge notamment :

- de changer les permissions sur la racine de travail qui va être utilisée par le code Python ;
- d'installer le paquet `p7zip` ;
- d'installer miniconda ;
- d'importer puis d'activer l'environnement conda.

Afin d'éviter tout incident lors du lancement de l'application Python, ce script vient, en amont, effectuer les ajustements nécessaires. En effet, par défaut, le programme va travailler dans `/srv`, qui est un emplacement approprié pour recevoir et stocker des données vouées à être partagées. On spécifie donc des droits permissifs sur ce répertoire.

L'installation du paquet `p7zip` permet d'obtenir l'exécutable qui gère le format de compression `.7zip`. L'exécutable en question (`7zr`) sera appelé indirectement par le programme via la bibliothèque `patool`.

`Miniconda`, quant à lui, s'installe à partir d'un script qui est téléchargé. Il est ensuite ajouté au path, configuré, puis mis-à-jour. Tout est alors prêt pour la dernière étape, très simple, qui consiste à importer l'environnement virtuel conda. Celui-ci a été généré au préalable et est fourni dans l'arborescence du projet.

De cette manière, une fois ce script exécuté, l'ensemble des installations sont terminées.

2.2.3 Création d'un environnement virtuel conda

Ce script est à destination des personnes souhaitant étoffer le projet à posteriori. Exécuté sur une machine ayant conda d'ores-et-déjà installé, il recrée et exporte l'environnement virtuel conda.

Voici un exemple élagué qui montre à quoi peut ressembler le fichier YAML* de l'environnement conda exporté.

Cela permet de faciliter l'ajout ultérieur d'une bibliothèque. En effet, il suffit d'ajouter le paquet nécessaire au sein du script et de le lancer. De cette manière, l'environnement virtuel conda nouvellement généré comportera la ou les bibliothèques ajoutées.

```
name: gd
channels: !!python/tuple
- conda-forge
- defaults
dependencies:
- conda-forge::fiona=1.7.6=np112py35_0
- conda-forge::json-c=0.12=0
- conda-forge::pcre=8.39=0
- conda-forge::pip=9.0.1=py35_0
- conda-forge::psycopg2=2.7.1=py35_0
- conda-forge::python=3.5.3=3
- conda-forge::readline=6.2=0
- conda-forge::shapely=1.5.17=np112py35_2
- conda-forge::xlrd=1.0.0=py35_1
- util-linux=2.21=0
- pip:
  - patool==1.12
prefix: /home/taunay/miniconda/envs/gd
```

FIGURE 2.1 – Environnement virtuel conda exporté (simplifié)

Chapitre 3

Interactions utilisateur : fichiers de configuration

3.1 Rôle des fichiers de configuration

Les fichiers de configuration sont au cœur du fonctionnement du logiciel. Renseignés par l'utilisateur, ils formalisent le traitement souhaité sur les données ainsi que leur conditions d'import. On peut donc énoncer les points nécessaires que doivent respecter les fichiers de configuration. Ainsi, l'utilisateur doit être en mesure, à travers ces fichiers :

- d'exprimer simplement les pré-traitements et les conditions d'import ;
- de spécifier le tout de manière assez formalisée afin que le fichier, ensuite lu par le programme, ne soit pas ambigü ;
- d'omettre des informations lorsque c'est possible, ce qui correspond à une délégation implicite au programme de certaines tâches.

3.2 Profil utilisateur

L'utilisateur final doit répondre à certains critères. Ces prérequis permettent de vérifier que l'utilisateur sera en mesure d'utiliser correctement le programme en suivant simplement le manuel utilisateur.

Ainsi l'utilisateur doit être capable, au minimum, d'utiliser basiquement la ligne de commande. C'est souhaitable afin de pouvoir lancer les scripts. Il doit également, si possible, être apte à naviguer dans l'arborescence voire effectuer des manipulations courantes sur les fichiers (déplacement, édition).

De cette manière, le logiciel peut tout-à-fait être utilisé par une personne qui ne développe pas.

3.3 Fichier général de configuration

Ce fichier, séparé des paramètres d'import spécifiques à chaque donnée, contient les paramètres communs à toutes les données. Ils sont d'une importance très variables : caractère visuel de séparation utilisé par les logs et IP* de la base de données, par exemple.

Dans un logiciel graphique, on peut se représenter ce fichier de configuration comme étant la version JSON de l'onglet "préférences" ou "options".

```
"res": {
  "import_mode": "controlee_nouvelle_table",
  "params": {
    "shortname": "epci_lamb93_2013",
    "data_name": "epci-20131220-5m",
    "uri": "https://osm13.openstreetmap.fr/.../epci-20131220-5m-shp.zip",
    "schema": "geofla",
    "table": "epci_lamb93",
    "year": 2013,
    "version": "epci_osm_2013",
    "srid_source": 4326,
    "srid_destination": 2154,
    "bindings": [
      {"from": "nom_epci", "type": "str:64", "to": "libelle_epci"},
      {"from": "ptot_epci", "type": "", "to": ""}
    ],
    "mode":"modified"
  }
}
```

FIGURE 3.1 – Exemple de fichier de configuration des imports

3.4 Fichier JSON de configuration des imports

3.4.1 Définition d'un import

Ce fichier a pour vocation de permettre à l'utilisateur de définir les données à importer et les traitements souhaités sur celles-ci. Face au grand nombre de sources de données qui peuvent être précisées par l'utilisateur, le défi ici consiste à construire une configuration assez souple. En effet, seule le strict nécessaire doit être précisé par l'utilisateur. En procédant de cette manière et à force de faire évoluer ce fichier de configuration, nous avons aboutit à un fichier semblable à l'exemple présenté ci-contre.

Sur cet l'exemple ci-contre, un champs de la donnée d'entrée est renommé, un est supprimé, et les autres sont conservés en l'état. En effet, un `"to"` vide est synonyme de suppression. Le mode `"modified"` précise que les autres attributs seront conservés. Il s'oppose au mode `"keep_only"` qui lui ne conservera que les champs `"from"` précisés dans la node `"bindings"`.

Ce type de souplesse permet de limiter le nombre de champs à renseigner. En effet, il serait extrêmement contraignant pour l'utilisateur d'avoir à préciser l'ensemble des attributs alors que seul quelques champs seront renommés ou laissés de côté. Dans le cas des données que l'on traite ici, l'intérêt est de taille car certaines données ont un nombre assez important d'attributs.

3.4.2 Problème soulevé

Le fichier JSON de configuration des imports permet de renseigner les imports à effectuer. Cependant, même muni d'un éditeur de texte approprié, éditer un fichier JSON pour y ajouter de nombreux imports présente un certain nombre d'inconvénients non négligeables.

- Les sources d'erreurs au niveau du formatage dûes aux copier-coller sont nombreuses : il est facile d'oublier une virgule ou une accolade fermante.
- Le format JSON, bien que décemment espacé et indenté, ne permet pas une vue d'ensemble satisfaisante : il est difficile pour l'utilisateur de se repérer au sein du fichier.

C'est pourquoi il a fallu trouver une réponse à ces problématiques, c'est-à-dire un autre moyen de saisie de l'entrée utilisateur.

3.4.3 Solution : fichier tableur et script de conversion

Le choix de ce moyen de saisie s'est porté sur une feuille de tableur, par souci de simplicité et de commodité. Une fois sa syntaxe fixée après une phase de concertation, il fallait pouvoir obtenir un fichier de configuration JSON à partir de ce fichier.

Le script `excel2conf.py` vient remplir ce rôle. Il prend en entrée un ensemble de chemins absolus pointant vers des fichiers excel contenant les informations sur les imports à effectuer. Ce script va produire, pour chacun de ces fichiers tableur un équivalent JSON. Les fichiers JSON alors produits pourront ensuite être fournis au système de la même manière qu'un fichier d'import réalisé manuellement.

Afin de mener à bien cette tâche, le script se repose sur la bibliothèque `xlrd`, qui lui permet de parcourir le fichier tableur. De cette manière, chaque feuille correspondant à un mode d'import valide (nouvelle table ou table existante) est parcourue. Au sein de ces feuilles, chaque ligne correspond à un import. Il devient alors très visuel et facile de remplir les imports souhaités. Cette solution se révèle d'autant plus satisfaisante lorsque le nombre de données à importer augmente.

Enfin, il est possible, et judicieux, de vérifier les fichiers JSON produits par le script avant de les fournir au programme. En effet, il arrive que certaines erreurs, difficilement remarquables au sein du tableur, apparaissent de façon beaucoup plus flagrante en JSON. On peut par exemple pointer du doigt les sauts de ligne issus de copier-coller, bien visibles en tant qu'`\n` dans le fichier de configuration JSON produit.

3.5 Fichier répertoriant les imports effectués

Une fois un fichier de configuration des imports obtenu et le programme principal lancé, toutes les données correspondant aux entrées du fichier de configuration des imports n'ont pas toutes subi le même traitement. Par exemple, l'URI* fournie par l'utilisateur peut tout à fait être momentanément inaccessible ou erronée. De ce fait, chaque ressource aboutit à un stade final qui lui est propre :

- non récupérée, ressource inaccessible via l'URI renseignée ;
- récupérée avec succès, mais format de compression non géré par un exécutable, disponible sur le système, auxquel la bibliothèque `patool` puisse faire appel ;
- récupérée et extraite, mais donnée souhaitée introuvable au sein de l'arborescence ;
- succès du parsage, des modifications structurelles et de la construction des requêtes mais échec au niveau de l'import en base de données ;
- chaîne complète de succès aboutissant à un import effectif en base de données.

La cause d'un échec fait l'objet de détails dans les logs, ce qui permet de la corriger avant le lancement suivant. Dès lors, la reprise s'effectuera à partir de la dernière étape réussie. C'est très facile de connaître le statut en se basant sur le système de fichiers pour les étapes de téléchargement de de décompression.

Cependant ce n'est pas le cas pour l'étape d'import en base. Afin de palier à ce problème, un fichier,

accessible à l'utilisateur est placé dans le même répertoire que le fichier de configuration des imports. Il retient chaque donnée importée, ce qui permettra à l'utilisateur de connaître facilement les imports finalement effectués. De même, s'il a besoin d'importer à nouveau une donnée il lui suffit de supprimer la ligne correspondant à cette donnée au sein de ce fichier.

Chapitre 4

Applicatif Python

4.1 Présentation générale du problème

Les scripts d'installation ont répondu, en partie, à la problématique générale posée initialement : construire, à partir de zéro, une base de données géoréférencées. En effet, ce sont les installations automatisées qui vont permettre de "partir de zéro". Vient dorénavant la question de l'import des données qui vont venir alimenter cette base de données géoréférencées.

Les données géoréférencées qui nous intéressent ici sont hétérogènes et de qualité variables. En effet les données à disposition sont mises en ligne par différents organismes. On peut citer par exemple l'INSEE*, l'IGN*, OpenStreetMap ou encore la CAF*. Cela induit le fait que les données peuvent être contenues dans des fichiers de formats variables comme le format `.shp`, le plus répandu, ou encore les formats `.mif/.mid` et `.geojson`. On trouve même des données, moins sujettes à du traitement automatique, par exemple des feuilles excel `.xls`. La variété des formalismes de chaque organisme rend difficile ces données à traiter, d'où l'intérêt de la création d'une base de données géoréférencées.

Vient ensuite l'aspect géoréférencé des données : ils donnent lieu à des traitements spécifiques. En effet, il peut être nécessaire de reprojeter des données afin que les données soient homogènes au sein d'une même table en base. L'extension PostGIS, dans sa table `spatial_ref_sys`, répertorie plus de 5000 projections différentes. Enfin, la colonne `geometry` gérant l'aspect géoréférencé des données d'entrée n'est pas à même d'être importée en base en l'état : la représentation attendue par PostGIS diffère.

4.2 Finalité du projet

L'applicatif Python constitue l'outil principal, il est au cœur du projet. L'application à concevoir et développer devra permettre de répondre au problème posé, c'est-à-dire de remplir une base de données géoréférencées. Les imports en base doivent pouvoir être effectués de manière souple, mais contrôlée.

De plus, les imports devront pouvoir s'effectuer de manière assistée, où les différentes actions effectuées ainsi que les problèmes rencontrés devront être indiqués.

Enfin, la mise en œuvre de l'application permet d'omettre certains paramètres dans les fichiers de configuration d'import. En cela, le programme est automatisé. Néanmoins, il reste soumis à la demande de l'utilisateur : le script n'est pas voué à se lancer tout seul à intervalles régulièrs, à l'aide de `cron` par exemple. Les tâches de vérification de nouvelles parutions de données restent manuelles. Ainsi les données stockées au sein de la base peuvent être mises-à-jour ou non au bon vouloir de l'utilisateur. C'est seulement alors qu'interviendra l'utilisateur afin de préciser les imports qu'il souhaite effectuer.

4.3 Fonctionnalités attendues

4.3.1 Paramétrabilité et facilité d'utilisation

L'utilisateur ne fait que déléguer à l'application les traitements à effectuer, ce qui ne le dispense pas de renseigner des paramètres afin de guider le programme. Les fichiers de configuration viennent répondre à ce besoin.

La principale contrainte pour ceux-ci est qu'ils doivent être à la fois concis, c'est-à-dire qu'il y ait le moins d'informations possible à renseigner par l'utilisateur ; mais aussi complets, c'est-à-dire que l'utilisateur puisse donner exactement ce qu'il souhaite.

Enfin, afin de faciliter l'utilisation du programme, un manuel utilisateur a été rédigé (voir annexe B). Il guide l'utilisateur et lui permet de s'orienter au sein du projet.

4.3.2 Journalisation des actions effectuées

La clarté d'exécution est un critère particulièrement important et attendu. De ce fait, une attention toute particulière à été portée sur ce point. C'est pourquoi les logs de l'application se veulent les plus clairs possibles. Le défi ici est d'être assez précis et concis, tout en restant assez général afin que l'utilisateur final ne se perde pas dans les détails pour autant.

Au sein du code source, cela se traduit par une vérification constante assez fine quant aux retours des fonctions clés de traitement. Ainsi, ces fonctions retournent un paramètre supplémentaire, indiquant à leur fonction appelante la réussite ou l'échec du traitement demandé, le cas échéant. La fonction appelée se sera alors chargée, si nécessaire, de produire une ligne de journalisation détaillant le problème rencontré pour cet appel.

```
1  def log(msg, lvl='info'):
2      """
3      Journalise les evenements de maniere formatee.
4
5      Les messages seront prefixes du niveau de severite.
6      Le LOG_LEVEL precise l'importance moindre a partir de laquelle afficher le message.
7      :param msg: string le message
8      :param lvl: string le niveau
9      """
10
11     def print_it():
12         printerr('[{}] {}'.format(lvl.upper(), msg))
13
14     levels_to_print = SEVERITY_LEVELS[:SEVERITY_LEVELS.index(LOG_LEVEL)+1]
15
16     if lvl in levels_to_print:
17         print_it()
18
19     # niveau special non defini par defaut
20     # on affiche donc le message independemment du LOG_LEVEL
21     if lvl not in SEVERITY_LEVELS:
22         print_it()
```

La principale fonction de journalisation, appelée à de nombreuses reprises au sein de l'exécution est détaillée ci-contre. On peut notamment remarquer la présence d'une configuration du niveau de gravité minimum à partir duquel afficher un log. Cela permet, par exemple, si l'utilisateur effectue un grand nombre d'imports simultanément, d'être en mesure de ne visualiser que les entrées de journalisation d'erreur et de mise en garde, s'il le souhaite.

4.3.3 Récupération des données

Le récupération des données s'effectue différemment en fonction des URI. En effet, le traitement se distingue entre les ressources locales (URI en `file:///`) et les ressources web distantes (URI en `http`). Pour les données accessibles en local, elles sont simplement copiées dans l'arborescence de travail du programme. Les ressources distantes sont téléchargées via des appels aux fonctions d'`urllib`.

```
with open(save_as, 'wb') as output_file:
    response = get(uri, verify=False)
    output_file.write(response.content)
```

Afin de pouvoir se déplacer manuellement dans les données téléchargées à des fins de consultation, copie ou partage, il est nécessaire que ces données soient correctement agencées. L'arborescence est tout d'abord découpée selon chaque site de téléchargement. Les données copiées sont dans un dossier `"copied"` dans ce même répertoire. Puis, chaque dossier de site se découpe en nom de données téléchargées. Il y en a un par archive distincte.

En procédant de la sorte, il est facilement possible de savoir si une donnée a d'ores-et-déjà été téléchargée. En effet, le chemin vers l'emplacement attribué à cette donnée fait office de clé. Il suffit alors de constater la présence ou l'absence de l'archive en question.

4.3.4 Décompression des archives

Les données mises à disposition par les différentes institutions sont fournies comme archives. On trouve surtout des `.zip` et `.7z`. La bibliothèque `patool` permet conserver un traitement standard pour tous les formats au niveau du code source de l'application. Cette bibliothèque se charge en réalité d'appeler l'exécutable sur le système approprié pour décompresser l'archive, après détection du type MIME* de celle-ci.

En se basant sur `patool`, j'ai donc défini une fonction de test pour déterminer si un fichier est constitutif ou non d'une archive.

```
def is_archive(params):
    """
    Determine si le fichier passe en parametre est une archive.

    :param params: les parametres
    :return: True si c'est une archive, False sinon
    """
    try:
        test_archive(params['path'], verbosity=-1, interactive=False)
    except Exception:
        return False
    return True
```

Cette fonction sert à sélectionner les archives lors de la recherche des archives à décompresser :

```
archives_to_extract = search_with_criteria(DOWNLOAD_ROOT, is_archive, search_depth=2)
```

La bibliothèque `patool` permet ensuite d'extraire chaque archive en fournissant uniquement son chemin, comme suit.

```
extract_archive(archive_path, verbosity=-1, outdir=archive_dir, interactive=False)
```

4.3.5 Recherche d'une donnée

Une problématique s'est posée lors de la recherche des données au sein des archives décompressées ayant été téléchargées. En effet, il s'est avéré que certains organismes regroupent plusieurs données au sein d'une même archive ressource. Il est donc fréquent d'observer plusieurs données au sein d'une même archive. Ces données peuvent être un fichier unique ou un ensemble de fichiers.

Deux cas de figures se présentent :

- L'utilisateur a renseigné le nom de la donnée de manière précise, c'est-à-dire le nom du ou des fichier(s).
- L'utilisateur a omis le nom de la donnée.

Dans le premier cas, il suffit d'utiliser ce nom au sein de la recherche. Ci-contre le code source de la fonction utilitaire de recherche récursive avec critère. Le `validator` qui sera passé ici est une fonction renvoyant un booléen. Ce booléen vaut `True` si le fichier correspondant au chemin absolu passé en paramètre correspond à un fichier qui nous intéresse, `False`.

```
# Prototype de la fonction de recherche recursive avec criteres.
def search_with_criteria(path_to_search_in, validator, validator_params=None, search_depth=0):
"""Recherche les fichiers satisfaisant un ensemble de critÃ¨res au sein du dossier specifie."""
```

La distinction avec le second cas s'exprime donc au niveau de la fonction de validation passée en paramètre. Si le nom précis de la donnée n'est pas renseigné, l'algorithme de recherche ne se basera que sur les extensions de fichiers afin de trouver des groupes intéressants de fichiers. Si plusieurs groupes sont trouvés, alors le traitement pour cette donnée s'arrête là. En effet, le programme avertit l'utilisateur que le nom de la donnée doit impérativement être renseigné pour cet import. Néanmoins, il n'y aura souvent qu'une seule et unique donnée potentielle au sein d'une archive. C'est pourquoi cette souplesse qui permet de ne pas renseigner le nom de la donnée est essentielle puisqu'elle épargne à l'utilisateur cette action.

4.3.6 Modification de leur structure

Les fichiers sont ouverts à l'aide de la bibliothèque `fiona`. Cette bibliothèque se concentre sur la lecture et l'écriture de donnée. Ce n'est pas elle qui est à même de gérer des transformations géographiques, par exemple. Cette bibliothèque permet notamment de lire un bon nombre de fichiers qui contiennent des données géoréférencées et de construire une structure python, en l'occurrence un dictionnaire, qui les représente en mémoire.

De plus, `fiona` permet de travailler avec les données en flux, ce qui est essentiel au vu de l'importante volumétrie des données que l'on traite ici. En effet, il est possible, en fonction du système sur lequel sera lancé l'application, que certains fichiers de données à importer soient trop conséquent pour être chargé en mémoire vive (RAM*). De ce fait, le traitement en flux revête toute son importance.

L'utilisateur peut préciser, au sein du fichier de configuration des imports, un certain nombre de modifications structurelles. Il peut aisément spécifier les actions suivantes :

renommage : modifier le nom d'un attribut ;

filtrage : ne conserver que les attributs qu'il spécifie ;

suppression : supprimer des attributs qu'il spécifie ;

choix déterminer le comportement du logiciel pour cette donnée, le "mode".

Le mode aura pour valeur `"modified"` ou `"keep_only"`. Dans le premier cas, les attributs non spécifiés par l'utilisateur seront conservés en l'état, sans aucune action de la part du logiciel. Dans le second cas, les attributs non spécifiés seront jetés.

4.3.7 Reprojection de jeux de données

Il arrive que des données que l'on souhaite ajouter à une table existante en base soient projetées dans un système de référence spatiale différent. Chacun de ces systèmes est associé a un identifiant unique, son SRID*. Le SRID source ainsi que le SRID de destination de la table doivent tous les deux à renseigner dans le fichier de configuration des imports.

Néanmoins, si les données le permettent (présence d'un fichier `.prj`), le logiciel doit pouvoir détecter la projection source. De même, il doit détecter le SRID de destination via une introspection de la table désirée pour l'import.

Une fois les SRID source et destination connus, la bibliothèque `pyproj` pourra par exemple être utilisée.

4.3.8 Importation en base

À l'aide la bibliothèque `psycopg`, il est très simple de se connecter au SGBD PostgreSQL :

```
1  connect(host=DB_HOST, port=DB_PORT, dbname=DB_NAME, user=DB_USER_NAME, password=DB_USER_PASSWORD)
```

La fonction `connect()` renvoie une connexion vers la base de données. Pour effectuer une requête, on peut procéder comme suit :

```
1  cur = conn.cursor()
2  cur.execute(query)
3  conn.commit()
```

Par ailleurs, les données sont traitées en flux. À partir de chaque entrée d'un fichier d'entrée, c'est-à-dire pour chaque futur tuple en base de données, le logiciel construit dynamiquement une requête d'insertion à envoyer en base de données, puis l'envoie.

```python
def build_insert_query(properties, params):
    """Construit une requete d'insertion dans une table a partir d'une feature donnee."""

    # valeurs intermediaires
    columns = list(properties.keys())
    wkt = shape(properties['geometry'])
    srid = params['srid_source']
    geometry_field = "ST_GeomFromText('{}', {})".format(wkt, srid)

    # champs pour requetes
    table_name = '"{}"."{}"'.format(params['schema'], params['table'])
    fields_list = '({})'.format(', '.join(columns))
    values = '({})'.format(', '.join([geometry_field if attr == GEOMETRY_NAME else "'{}'".format(p.
        replace("'", "''")) if isinstance(p, str) else str(p) for attr, p in properties.items()]))

    query = 'INSERT INTO {} {} VALUES {};'.format(table_name, fields_list, values)

    return query
```

Cette construction des requêtes d'insertion, dont la fonction est détaillée ci-contre, permet de mettre en évidence le traitement spécifique réalisé pour l'aspect géoréférencé des données. En effet, la colonne nommée `geometry` des données d'entrées — ici, une clé au sein du dictionnaire délivré par `fiona` — n'est pas de la forme attendue par l'extension PostGIS, côté base de données. En effet, la représentation de l'attribut `geometry` dans PostGIS diffère de celle qui est attendue. Cependant, on ne peut l'obtenir directement.

Le Well-Known Text (WKT*) est un format standard qui sert d'intermédiaire. En effet, du côté base de données, l'appel à `ST_GeomFromText` permet de transformer les données afin de pouvoir les stocker dans la colonne `geometry`, à partir du well know text. Ainsi, côté Python, il faut obtenir le WKT. Comme on peut le voir ici, c'est à l'aide de la fonction SHAPE() de la bibliothèque SHAPELY que le logiciel obtient le WKT à partir des données d'entrée.

4.3.9 Comportement intelligent

Un comportement dit "intelligent" est attendu. Lors de la recherche d'une donnée, le nom du fichier souhaité pouvait être omis par l'utilisateur afin que le programme le détecte de lui-même. C'est un exemple de comportement "intelligent".

De même, le logiciel produit des avertissements. Ceux-ci n'interrompent pas le traitement, mais font état d'un petit problème qui a été rencontré. En effet, il avertit l'utilisateur lorsque les fichiers de configuration comportent des fautes mineures de formatage ou de logique au sein du fichier de configuration des imports :

- l'utilisateur a renseigné une clause "drop" alors que le mode est "keep_only", c'est un non-sens qui est signalé car il est inutile de renseigner un attribut à retirer qui sera de toute manière laissé de côté ;
- l'utilisateur a renseigné un nom peu approprié pour un dossier de stockage d'une donnée à récupérer ; un nom corrigé est alors suggéré.

Enfin, il est inconfortable de retirer du fichier de configuration les données ayant d'ores-et-déjà été traitées. De ce fait, le programme ne téléchargera pas à nouveau une donnée ayant déjà été téléchargée. Il en va de même au sujet des imports en base de données.

Conclusion

État de l'avancement du projet

Le logiciel réalisé permet de répondre au besoin général initialement énoncé, à savoir créer, facilement et à partir de zéro, une base de données géographiques. Des scripts permettent d'effectuer automatiquement les installations et configurations adéquates. Finalement, le programme principal permet de :

- récupérer les données à traiter (téléchargement) ;
- décompresser les archives récupérées ;
- explorer ces données afin d'en trouver la donnée intéressante ;
- parser ces données (supporté pour un nombre limité de formats) ;
- y appliquer les pré-traitements nécessaires ;
- les importer en base.

L'ensemble des fonctionnalités implémentées ont été testées manuellement. J'ai cherché à éprouver les cas spécifiques. Ainsi j'ai notamment vérifié le comportement de l'application pour différentes configuration d'import.

Limites de l'outil principal développé

Il est important de mettre en avant que l'application en elle-même n'est pas limitée par la taille des données. En effet, l'ensemble des traitements s'effectuant en flux, la capacité du système en mémoire vive ne pose pas de problème.

Seule la capacité de stockage en mémoire morte des données d'entrée peut constituer un facteur limitant. par ailleurs la qualité de la connexion internet influe beaucoup sur la durée de rapatriement des données, puisque la majorité des ressources sont des ressources distantes. Ainsi la vitesse de traitement est environ linéaire par rapport au volume des données d'entrée.

En ce qui concerne la structure générale du projet, le choix d'avoir développé les différents modules en les rendant indépendants les uns des autres peut être critiqué sur le long terme. Certes, chacun constitue une brique réutilisable à part. Cependant, cette conception trouve ses limites car chaque module doit aller retrouver et parser les fichiers de configuration, tour à tour. Un changement de structure au niveau de ces configuration serait difficile à opérer. Cela explique le fait d'avoir fixé au plus tôt la structure de ces fichiers.

Ainsi, il aurait été possible de s'appuyer sur le paradigme de la programmation orientée objet afin de créer une classe représentant une donnée à importer. Cette dernière aurait par exemple un attribut représentant son stade actuel ; c'est-à-dire récupérée, extraite ou importée. En procédant de la sorte, un gain de modularité supplémentaire serait obtenu.

Perspectives d'amélioration

Les scripts shell développés respectent les bonnes pratiques de développement, mais ne se plient pas pour autant aux normes POSIX*. Respecter ces contraintes au sein de ces scripts augmenterait leur portabilité.

Le développement d'une interface graphique pourrait être réalisé. Celle-ci permettrait par exemple de créer plus simplement les fichiers de configuration, de façon totalement transparente à l'utilisateur. Ainsi, le programme gagnerait énormément en interactivité et en sûreté. En effet le fichier de configuration généré serait ainsi systématiquement correct sur la question du formatage. De même, l'utilisateur pourrait avoir la possibilité d'importer en filtrant via des critères, ou de n'effectuer que l'étape de téléchargement pour une donnée par exemple. Une telle surcouche demanderait un développement assez conséquent car on peut alors imaginer beaucoup de nouvelles fonctionnalités comme ces quelques exemples.

Pour ce qui est de l'outil principal, la gestion d'un plus grand nombre de formats est un point à travailler. En effet, la diversité des sources de données, et, de façon analogique, des données qu'elles mettent à disposition est assez importante. Néanmoins il faut garder à l'esprit que la souplesse de l'application développée permet de distinguer aisément les traitement associés aux différents cas.

Expérience acquise

Premièrement, j'ai constaté l'importance d'accorder une attention toute particulière aux étapes en amont de l'implémentation. Ainsi, j'ai progressé au niveau du recueil du besoin et de sa définition, mais également sur les activités de conception. En effet, au cours de l'élaboration de la solution, la majorité des problèmes potentiels ont été soulevés tôt. Cela a permis d'orienter la conception de l'application, notamment au niveau de son découpage. De cette manière, l'application est suffisamment modularisée pour gérer tous les cas : l'ensemble des traitements communs à plusieurs scénarios d'exécution sont factorisés.

De plus, ce stage a été l'occasion d'utiliser mes compétences de scripting shell en bash lors de la réalisation des scripts d'installation et de configuration.

Par ailleurs, je suis monté en compétence sur le langage Python. En particulier, j'ai été amené à manipuler plusieurs bibliothèques, énoncées ci-après. Chacune est accompagnée d'un bref descriptif de l'utilisation qui en est faite au sein du projet.

patool : décompression en fonction du type MIME ;

xlrd : parsage du fichier excel précisant les imports ;

fiona : parsage des données à traiter ;

shapely : modifications sur la colonne geometry ;

psycopg2 : connexion à la base de données.

En ce qui concerne les outils, j'ai appris à utiliser conda afin de gérer efficacement des environnements virtuels. J'ai aussi progressé sur le langage de présentation LaTeX lors de la rédaction du cahier des charges et de ce rapport.

Glossaire

AME Département Aménagement, Mobilité et Environnement.. 3

Atom Éditeur de texte développé par GitHub.. 5

CAF Caisse d'Allocations Familiales.. 14

conda Gestionnaire de paquets et d'environnements virtuels.. 2, 5

EPST Établissement Public à caractère Scientifique et Technologique.. 3

git Logiciel de gestion de versions décentralisé.. 5

GitHub Service web d'hébergement et de gestion de développement de logiciels, qui utilise le logiciel de gestion de versions git.. 5

IDE Integrated Development Environment — Environnement de développement.. 5

IFSTTAR Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux.. 1

IGN IInstitut géographique national.. 14

INSEE Institut National de la Statistique et des Études Économiques.. 14

IP Internet Protocol address — Identifiant attribué à un appareil au sein d'un réseau informatique.. 10

JetBrains Entreprise informatique éditrice, entre autres, de l'IDE PyCharm.. 5

JSON JavaScript Object Notation — Notation objet empruntée au JavaScript.. 2

LAMES Laboratoire auscultation, modélisation, expérimentation des infrastructures de transport.. 3

LaTeX Langage et système de composition de documents.. 5

LTS Long-Term Support — Support sur le long terme.. 5

markdown Langage de balisage léger qui offre une syntaxe facile à lire et à écrire.. 5

MIME Multipurpose Internet Mail Extensions — Identifiant de format de données.. 16

pip Pip Installs Packages — Gestionnaire de paquets pour Python.. 5

POSIX Portable Operating System Interface for uniX — Famille de normes techniques qui standardisent les interfaces de programmation des logiciels destinés à fonctionner sous un OS UNIX. 20

PyCharm IDE pour le Python développé par JetBrains.. 5

RAM Random Access Memory — Mémoire vive.. 17

SGBD Système de Gestion de Base de Données.. 2

SIG Système d'Information Géographique.. 1

SRID Spatial Reference System Identifier — Identifiant de Système de Référence.. 18

SSH Secure Shell — À la fois programme informatique et protocole de communication sécurisé.. 5

SublimeText Éditeur de texte développé par Jon SKINNER.. 5

URI Uniform Resource Identifier — Chaîne de caractères identifiant une ressource sur un réseau.. 12

UTF-8 Universal Character Set Transformation Format — Codage de caractères informatiques compatible avec le standard Unicode.. 7

WKT Well-Known Text — C'est un format standard en mode texte utilisé pour représenter des objets géométriques vectoriels issus de SIG, mais aussi des informations s'y rattachant, tels les SRID.. 19

YAML YAML Ain't Markup Language — Format de données textuelles, aéré visuellement.. 8

Sources

PostgreSQL/PostGis

Comparatif MySQL et PostgreSQL

https://openclassrooms.com/courses/mysql-et-postgresql-lequel-choisir

Tutoriel sur les rôles et permissions

https://www.digitalocean.com/community/tutorials/how-to-use-roles-and-manage-grant-permissions-in-postgresql-on-a-vps--2

Gestion des droits

https://wiki.postgresql.org/images/d/d1/Managing_rights_in_postgresql.pdf

Introduction à PostGIS

http://suite.opengeo.org/docs/latest/dataadmin/pgGettingStarted/firstconnect.html

Les schémas

https://www.postgresql.org/docs/current/static/ddl-schemas.html

Scripting shell

Génération d'un mot de passe aléatoire

https://www.howtogeek.com/howto/30184/10-ways-to-generate-a-random-password-from-the-command-line/

Documentation conda

https://conda.io/docs/using/envs.html

Python

Tutoriel SDZ/OC

https://openclassrooms.com/courses/apprenez-a-programmer-en-python

Erreurs courantes au sujet des imports

http://python-notes.curiousefficiency.org/en/latest/python_concepts/import_traps.html

Notions basiques de manipulation de dossiers

http://www.diveintopython.net/file_handling/os_module.html

PEP 257 sur les docstrings

https://www.python.org/dev/peps/pep-0257/

PEP 343 sur le mot-clé "with"

https://www.python.org/dev/peps/pep-0343/

Bibliothèques et aspect géo-référencé des données

Librairie GDAL (Geospatial Data Abstraction Library)

http://www.gdal.org/

Spécifications GeoJSON

https://tools.ietf.org/html/rfc7946

Modules python à finalités géospatiales : index à tutos, récapitulatif / descriptifs

http://www.portailsig.org/content/les-modules-python-finalites-geospatiales-quid-quando-ubi

Coment combiner l'utilisation des bibliothèques shapely et fiona

https://macwright.org/2012/10/31/gis-with-python-shapely-fiona.html

Fiona, geojson, et attributs

https://gis.stackexchange.com/questions/41465/generating-geojson-with-python

Fiona : ajout de propriétés

https://gis.stackexchange.com/questions/191900/fiona-error-adding-property-to-shapefile/191953

Fiona, geojson, et projection

https://gis.stackexchange.com/questions/192830/fiona-does-not-specify-crs#192873

Documentation sur l'utilisation d'osgeo

https://pcjericks.github.io/py-gdalogr-cookbook/vector_layers.html

Documentation pyproj

https://jswhit.github.io/pyproj/

Rédaction du rapport

Tutoriel SDZ/OC

https://openclassrooms.com/courses/redigez-des-documents-de-qualite-avec-latex

Compilation de conseils

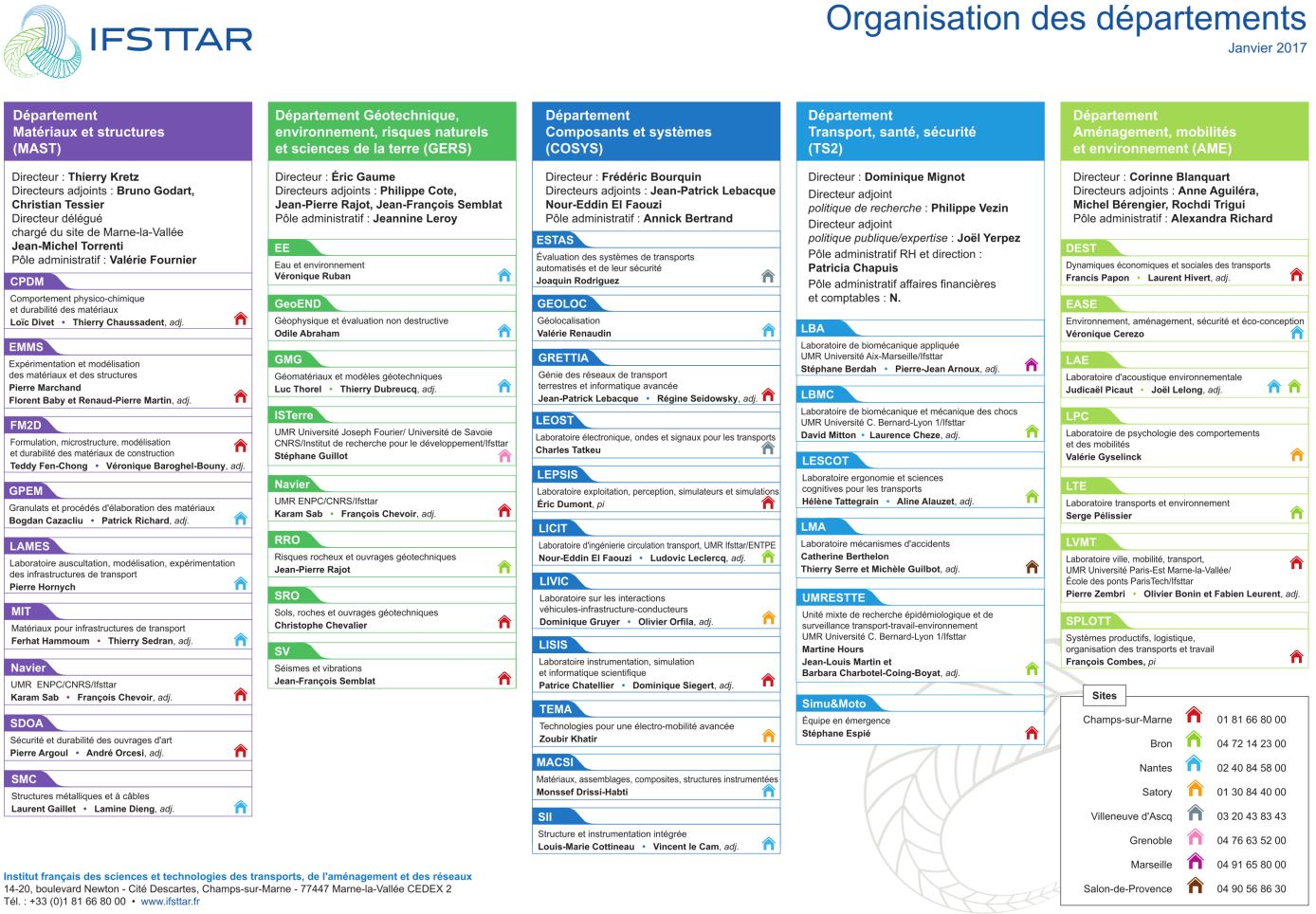
http://enicolashernandez.blogspot.fr/2011/03/redaction-dun-rapport-de-stage.html

Données chiffrées et ressources visuelles

http://www.ifsttar.fr/accueil/

Annexes

A. Organigramme des départements de l'IFSTTAR



Suivez le guide !

Ce manuel d'utilisation permet de s'orienter au sein de ce projet.
Il contient l'ensemble des remarques et consignes importantes à garder à l'esprit et à respecter.

Que puis-je faire à l'aide de ce projet ?

C'est un outil permettant de faciliter la création d'une base de données géo-référencées.

Il assiste certaines tâches. Il peut notamment se charger :

- de récupérer les données que l'utilisateur souhaite importer
- d'effectuer certains pré-traitements sur ces données
- de les importer dans une base "prête à requêter"

De plus, le projet contient des scripts d'installation permettant d'effectuer les actions préliminaires de adéquates de façon automatisée.
(pour systèmes *Linux* uniquement)

Avant de commencer

Arborescence du projet

Par souci de clarté / lisibilité, certains fichiers et dossiers ont été omis.
les éléments utiles --> **ont une description**

├─── README.pdf --> **fichier "lisez-moi"**
├─── documents

| ..... ├─── in
| ..... └─── out
| ..... ├─── reflexions
| ..... | ..... ├─── choix
| ..... | ..... └─── procedures
| ..... | ..... ├─── creation_fichier_import
| ..... | ..... | ..... └─── example.xlsx --> **fichier excel type**
| ..... | ..... ├─── miniconda
| ..... | ..... | ..... └─── desinstallation.pdf
| ..... | ..... ├─── postgresql
| ..... | ..... | ..... └─── desinstallation.pdf
| ..... | ..... └─── utilisation
| ..... | ..... └─── manuel_utilisateur.pdf --> **ce manuel**
| ..... └─── rendus
|
└─── src --> racine du code source
..... ├─── python
..... | ..... ├─── config.py
..... | ..... ├─── main.py --> **outil principal**
..... | ..... └─── packages
..... ├─── res
..... | ..... └─── gd.yml --> **env. conda à importer**
..... └─── scripts
..... ├─── create_venv.sh --> **regénère le venv en .yml**
..... ├─── excel2conf.py --> **excel vers json de config**
..... ├─── prepare_database.sh --> **installe** / **configure postgresql**
..... └─── prepare_environment.sh --> **prépare l'env. d'exécution**

Quelques indications et consignes

Ne pas altérer de quelque manière que ce soit (modification, suppression,

déplacement, création, etc.) la structure du projet dans le répertoire `src`. En effet ce dossier contient du code source et sa structure ainsi que ses fichiers et leurs contenus sont nécessaires au fonctionnement du programme. Ne pas y toucher, donc. *(sauf indication contraire)*

Prérequis

Pour une installation à partir de zéro

(À n'effectuer que la toute première fois.)

- un système d'exploitation *Linux*
- un système propre, c'est-à-dire à jour et fonctionnel *(comme pour tout logiciel)*
- des droits administrateur, "**root**", afin de pouvoir efffectuer des `sudo` lors de la phase d'installation
- une connexion internet

Pour l'exécution de l'outil seulement

- n'importe quel système d'exploitation (seul *Ubuntu 16.04 LTS* a été testé)
- avoir effectué les installations de la phase préparatoire (base de données **postgresql** + **environnement d'exécution**)
- éventuellement une connexion internet s'il faut récupérer des données distantes

Phase préparatoire

Préparation de la base

Si une ou plusieurs base(s) `postgresql` sont d'ores-et-déjà présente(s),

ignorer cette étape en passant directement à la suivante. S'adapter en conséquence.

Il ne doit y avoir aucune trace résiduelle d'une quelconque installation antérieure ou actuelle de postgresql/postgis (aller consulter `documents/out/reflexions/procedures/postgresql/desinstallation.pdf` et suivre les consignes pour le vérifier !).

Une fois ces vérifications effectuées, naviguer dans le projet pour se rendre dans le dossier qui regroupe l'ensemble des scripts (`src/scripts`).

Éventuellement, modifier le nom de la base ainsi que le mot de passe pour l'utilisateur `postgres` comme désiré.
Les deux variables concernées se situent en haut du script.

Pour finir, lancer le script en suivant les indications qui apparaissent.
`sudo ./prepare_database.sh`

Préparation de l'environnement d'exécution

La commande `conda` (*anaconda* ou *miniconda*) ne doit pas être installée. Taper `conda --version` pour le vérifier.

Si `conda` est d'ores-et-déjà installé :

- obtenir la liste des environnements présents avec `conda info --envs`
- aller regarder le nom dans `src/res` du fichier en `.yml` , c'est le nom de l'environnement que l'on souhaite importer
- vérifier qu'un environnement de la liste ne porte pas déjà ce nom, sinon il va y avoir un conflit
- si c'est le cas, il faut changer le nom de l'environnement importé en changeant le nom du fichier `.yml` , ainsi que les occurrences de son

nom au sein du fichier lui-même (faire un rechercher/remplacer)
- importer cet environnement avec la commande `conda env create -f le_nouveau_nom.yml`
- l'activer en tapant `. activate le_nouveau_nom`

Sinon, il faut seulement lancer le script approprié :
`. ./prepare_environment.sh`

Phase d'exécution

Création d'une configuration d'import

Il y a deux manières de fabriquer un fichier de configuration d'import.

La première méthode, assez terre-à-terre, est de l'écrire à la main. Pour cela, faire l'étape **1** du lancement ci-dessous, puis modifier le fichier `/srv/geodata/configuration/data.conf.json`. Ce fichier, copié à partir de `src/res/data.conf.json`, contient une node **documentation**. Lire son contenu avant de commencer la rédaction du fichier de configuration.

La seconde méthode permet de renseigner les imports d'une manière plus commode : renseigner les imports dans un fichier excel. Suivre la procédure suivante :

- prendre pour point de départ le fichier type `documents/out/reflexions/procedures/creation_fichier_import/example.xlsx`
- le modifier en calquant sa syntaxe et son formatage
  attention à la respecter attentivement
  écueils les plus courants :
  - cellules vides pas réellement vides " "

- saut de ligne au sein d'une cellule
- séparateur en trop (, ou ;)

- lancer le script de conversion comme suit
  `src/scripts/excel2conf.py /path/absolu/excel.xlsx`
  qui va générer un fichier en `.data.conf.json`
  dans le même répertoire que le fichier excel source
- l'ouvrir pour vérifier et corriger
  en effet certaines fautes sont difficilement visibles dans le fichier excel
  exemple : un saut de ligne provenant d'un copier/coller
  une fois transcrit en JSON, les erreurs se remarquent plus facilement

Lancement

1. première exécution "à vide" afin de générer l'arborescence de travail (par défaut dans `/srv`)
2. fabriquer un fichier de configuration : voir "Préparation au lancement"
3. le déplacer dans `/srv/geodata/configuration/` et le nommer `data.conf.json` Éventuellement conserver le `data.conf.json` existant : le déplacer et/ou le renommer.
4. exécution de l'outil
   - aller dans `src/python`
   - si la commande `which python` n'indique pas `nom_user/miniconda/envs/nom_environnement/bin/python` alors activer l'environnement virtuel conda `. activate nom_environnement`
   - lancer le programme `./main.py`

Pour importer à nouveau des données, deux choix :

- réitérer les étapes **1**, **2** et **3** ci-dessus
- réitérer l'étape **1** puis ajouter dans le `data.conf.json` déjà en place les imports du fichier fabriqué

Vérifications

Beaucoup d'erreurs de causes multiples sont susceptibles de survenir pendant l'exécution. Il est donc souvent nécessaire d'effectuer de multiples tentatives pour une même donnée à importer. En effet, à chaque essai et jusqu'à réussite, il faut modifier la configuration associée à cette donnée. Pour cela se référer aux messages de journalisation affichés dans le terminal.

La liste des données ayant été traitées complètement, c'est-à-dire importées en base avec succès, se trouve dans `/srv/geodata/configuration`. Cette liste n'a pas vocation a être seulement lue. En effet, elle peut être modifiée, par exemple pour réitérer un import déjà effectué. Le programme se base sur ce fichier pour déterminer si une donnée a été importée précédemment avec succès. Ainsi il suffit de supprimer la ligne correspondant au chemin de la donnée à importer à nouveau.

C. Cahier des charges

Cahier des charges

Stage de fin de DUT à l'Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux

IUT Informatique de Nantes

Rémi TAUNAY

5 mai 2017

Table des matières

Première partie

Présentation générale du problème

1 Projet

1.1 Finalité

L'objectif de ce projet est de créer une base de données géo-référencées. Cette base est destinée à constituer un rassemblement volumineux de données géographiques sur le territoire. Les données sont rapatriées de sources diverses puis traitées automatiquement. Enfin, elle sont importées dans la base.

À des fins d'études recroisant ces données, cette base est vouée à être un source de données commune à de multiples utilisateurs. Dans cette optique, cette base sera munie d'un ensemble de requêtes basiques, c'est-à-dire de filtres larges permettant une utilisation facilitée de ces données par des logiciels de traitements, en aval.

1.2 Problématique

À l'IFSTTAR, certaines recherches requièrent la manipulation de données géographiques : statistiques, recroisements, simulations. Actuellement le regroupement, stockage, filtrage, et partage de ces données est réalisé manuellement. Par conséquent, les mêmes manipulations sont souvent répétées, ce qui constitue une perte d'efforts et de temps.

1.3 Énoncé du besoin

Le travail à effectuer vise à créer un serveur de données centralisé en important automatiquement des données, en les pré-traitant, puis en les rapatriant dans la base.

L'utilisateur doit pouvoir définir l'ensembles paramètres d'exécution avant le lancement. Ainsi, sans modifier le code source, il doit être en mesure de spécifier les éléments énoncés ci-dessous.

comportement : réaction du programme à une situation donnée (exemple : niveau de verbosité)

structure de la base : tables, attributs, clés, relations, contraintes

données : source des données et traitements attendus (exemple : spécifier la table qui va accueillir un ensemble de données)

L'ensemble de ces spécifications s'effectueront à travers des fichiers de configuration, de format JSON. Toute interaction entre le logiciel et l'utilisateur s'effectuera par ces fichiers. Ainsi, l'utilisateur n'aura pas besoin d'altérer le code source pour contrôler le comportement du logiciel.

De plus, seuls les paramètres absolument nécessaires seront à préciser, comme par exemple la structure de la base si l'utilisateur souhaite la créer. L'utilisateur aura bien entendu la liberté de redéfinir les autres paramètres au cas où leurs valeurs par défaut ne lui conviennent pas.

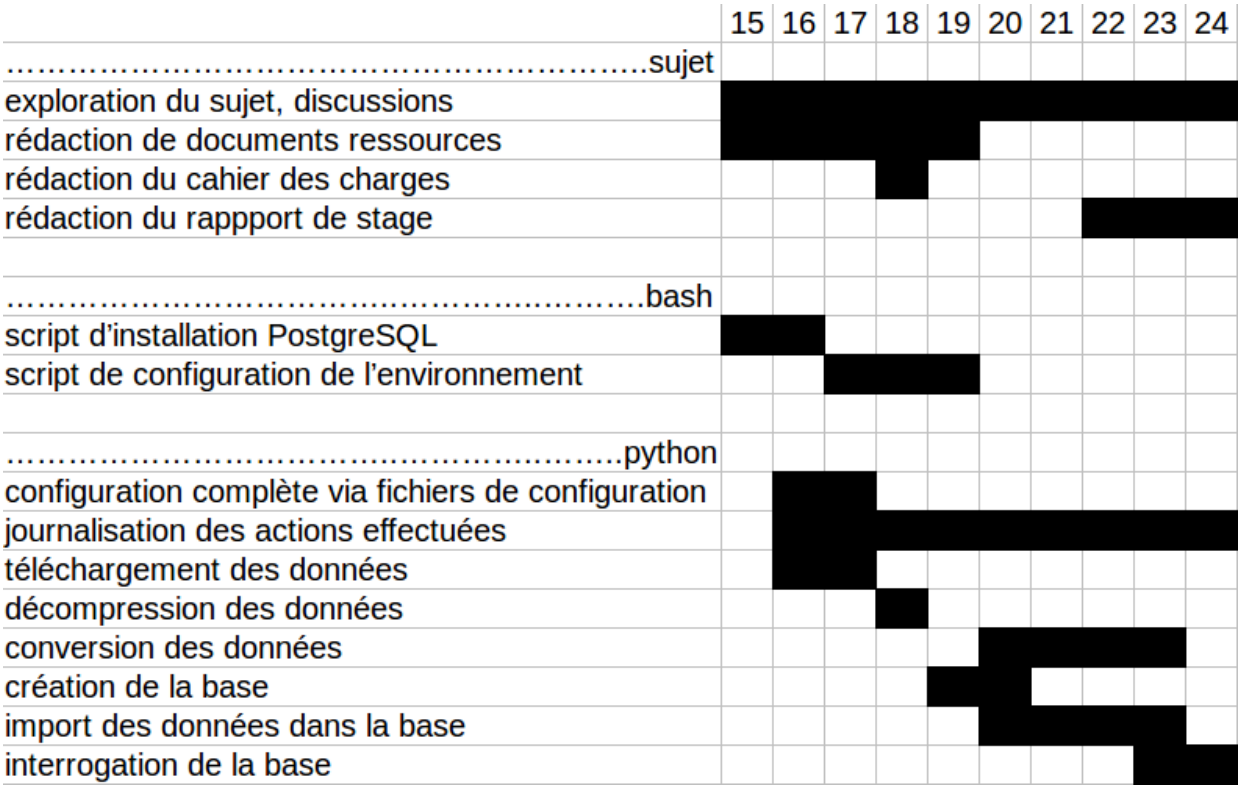
1.4 Solution apportée

Pour répondre au besoin énoncé ci-dessus, une base de données centralisée ainsi que des procédures d'importation, de nettoyage, et de traitement de données sont crées. L'ensemble de ces solutions permet notamment :

- un stockage massif performant de données
- l'élimination des tâches de pré-traitement : récupération, nettoyage, uniformisation, recherche de liens (clés & relations)
- la mutualisation des données, rendues accessibles à de multiples utilisateurs
- la mise à disposition de requêtes basiques de sélection avec application de filtres (attributaires, géographiques)

1.5 Planification

Diagramme de Gantt :



2 Contexte

2.1 Études déjà effectuées

Marc Noël a d'ores-et-déjà effectué un travail préliminaire sur les données et établi les liens existants. En effet, en parcourant et effectuant des traitements sur ces données il a pu conçevoir le schéma de la base de données.

C'est ce schéma qu'il revient ici d'implémenter, en discutant des adaptations mineures éventuelles. Ce

schéma se veut proche des données d'entrée. De cette manière, l'intégration des données est facilitée.

2.2 Nature des prestations demandées

En premier lieu, le développement de deux scripts shell (`bash`). Ces scripts, non interactifs, vont venir préparer l'exécution ultérieure du code `Python`. En effet, ils se chargent de d'installer et configurer automatiquement l'ensemble des composants de l'environnement. Le premier se charge d'installer et de configurer la base `PostgreSQL`. Le second se charge de préparer l'exécution du code `Python` en installant les dépendances (les packages) auxquels il fait appel.

En second lieu, le développement d'un script `Python` chargé de manipuler les données. Ce dernier doit effectuer un ensemble de traitements tels que précisés par l'utilisateur.

2.3 Parties concernées par le déroulement du projet et ses résultats

En ce qui concerne l'IUT informatique de Nantes, Monsieur Loïg Jezequel, est chargé en tant qu'enseignant référent de suivre et évaluer ce stage. Ce suivi est réalisé via un message électronique envoyé en chaque fin de semaine, ainsi qu'une visite sur le lieu de stage. De même, plusieurs rendus sont à fournir tel qu'un résumé du sujet, ce cahier des charges, ainsi que le rapport de stage.

En ce qui concerne la structure d'accueil, l'IFSTTAR, Pascal Gastineau et Pierre Hankach jouent le rôle de tuteurs. Ils seront également les commanditaires et les clients du développement effectué.

Enfin, une soutenance ponctue le stage. Le jury sera alors composé de Pierre Hankach, Loïg Jezequel, ainsi qu'un second professeur de l'IUT.

Deuxième partie

Détail technique du besoin

3 Étapes préliminaires : scripts shell

3.1 Installation et configuration de `PostgreSQL`

En premier lieu, le rôle de ce script est d'installer `PostgreSQL` et l'extension `PostGIS`.

En second lieu, ce script a pour rôle de configurer `PostgreSQL`. Cela est réalisé via l'édition automatisée de fichiers de configuration, et permet l'obtention des fonctionnalités suivantes :

compatibilité : support UTF-8 pour la base `template1`

rôles : création de deux rôles (groupes) distincts permettant l'un la lecture et l'autre écriture ; ils seront hérités par les utilisateurs créés manuellement à posteriori

création : création de la base qui recevra les données

sécurité : ajout d'un mot de passe pour l'utilisateur `postgres`

journalisation : connexions et déconnexions

authentification définition de la politique de sécurité

connexions : autorisation des connexions à distance

3.2 Préparation de l'environnement d'exécution

Le rôle de ce script est de préparer l'environnement d'exécution. Sa tâche principale est d'effectuer une installation de `miniconda` et d'importer un l'environnement adéquat. Cet environnement aura été préalablement exporté et conservé dans l'arborescence du projet.

4 Applicatif `Python`

4.1 Description fonctionnelle

Le code `Python` devra fournir les fonctionnalités suivantes :

- configuration complète via fichiers de configuration
- journalisation des actions effectuées
- téléchargement des données
- décompression des données
- conversion des données
- création de la base
- import des données dans la base
- interrogation de la base

4.2 Modularité, découpage en sous-ensembles

Le découpage en sous-ensembles s'articule autour de modules. Chaque module regroupe un ensemble logique de fonctionnalités. La figure suivante montre l'arbre décrivant une structure possible du projet.

```
.
├── config.py
├── main.py
└── packages
    ├── database
    │   ├── creation
    │   │   └── create.py
    │   ├── filling
    │   │   └── fill.py
    │   └── querying
    │       └── query.py
    ├── data_cleaning
    │   └── convert.py
    ├── data_retrieving
    │   ├── download.py
    │   └── extract.py
    └── utils
        ├── json.py
        ├── log.py
        ├── path.py
        └── string.py
```

Cette structure n'est pas définitive. En effet, elle est sujette à être étoffée ou redécoupée si nécessaire. Cependant, le découpage montrera toujours clairement les différentes tâches, de manière à les séparer. En effet les modules de plus haut niveau correspondent aux tâches principales de l'application.

D. Journal de bord

lundi 10/04/2017

- Visite du bâtiment
- Formalités administratives
- Installation environnement de travail (xfce, environnemment bash). Création mini-serveur samba pour échange facile de documents avec Marc

mardi 11/04/2017

- Collecte/lecture de tutoriels et documentation sur postgres
- Installation manuelle de postgres et premiers tests

mercredi 12/04/2017

- Réflexions sur les configurations post-installation à effectuer au sein du script
- Début d'un script bash de configuration de postgres

jeudi 13/04/2017

- Avancement sur le script et lecture doc postgres

vendredi 14/04/2017

- IFSTTAR fermé

lundi 17/04/2017

- IFSTTAR fermé

mardi 18/04/2017

- Petite réunion autour du script bash terminé. Questions/Réponses sur celui-ci et complétion de la documentation. Explication de mon script en détail

mercredi 19/04/2017

- Longue discussion avec Marc sur les intérêts de la création de cette base. Exemple "bureau de vote" sur des logiciels de traitement. Problématiques inhérentes aux traitements de données géo-référencées
- Rédaction d'un résumé du sujet de stage et de mes premières impressions
- Apprentissage du modèle de données créé par Marc

jeudi 20/04/2017

- Ajout d'une section "Troubleshooting" après la démarche d'installation qui donne des commandes bash utiles
- Python création de fonctions utilitaires de logs

vendredi 21/04/2017

- Passage de l'embryon de projet python sous PyCharm
- Lecture de PEPs

lundi 24/04/2017

- dev. début téléchargement

mardi 25/04/2017

- dev. téléchargement des archives

mercredi 26/04/2017

- Réflexions/discussions sur les interactions utilisateur
- Début de la création d'un "système de configuration"

jeudi 27/04/2017

- Système de configuration "intelligent" terminé
- Lecture de doc sur la librairie GDAL

vendredi 28/04/2017

- Nouveau script bash qui sépare la configuration de l'environnement pour python de celle puremment pour postgres

lundi 01/05/2017

- IFSTTAR fermé

mardi 02/05/2017

- Apprentissage de conda (miniconda sur dépôt conda-forge), environnements virtuels, pip

mercredi 03/05/2017

- Réunion : chacun présente son avancement
- Discussions sur ce qu'il y a à faire en priorité et sur les questions à creuser
- Décompression des archives téléchargées

jeudi 04/05/2017

- Lecture partielle du tutoriel SDZ/OC LaTeX
- Recherche et installation d'outils qui me conviennent pour éditer du LaTeX
- Squelette LaTeX cahier des charges

vendredi 05/05/2017

- Rédaction du cahier des charges

lundi 08/05/2017

- IFSTTAR fermé

mardi 09/05/2017

- Scripts bash pour conda terminés. 1 supplémentaire, standalone pour créer un venv et l'exporter
- Décompression via autodétection du type MIME

mercredi 10/05/2017

- Conception et réalisation : retours et réajustements sur le fichier de configuration des imports

jeudi 11/05/2017

- Renommage des archives téléchargées avec leur extension appropriée
- Tests et corrections mineures

vendredi 12/05/2017

- Modularisation en fonction des URI
- Conception algorithme de recherche pré-conversion

lundi 15/05/2017

- Implémentation de l'algorithme
- Tests

mardi 16/05/2017

- Finalisation de l'algorithme
- Discussions et décisions : entrée utilisateur (excel + script standalone de conversion)
- Retrait des modes automatiques dans le fichier de configuration des imports

mercredi 17/05/2017

- Distinguer les fichiers essentiels lors de la phase de tri/conversion
- Amélioration de la qualité des logs

jeudi 18/05/2017

- Lectures sur le mot-clé yield en python et les concepts rattachés
- Lecture sur les libraires (fiona, ogr2ogr) et sur les technos qui tournent autour
- Début parsage des données d'entrée avec Fiona

vendredi 19/05/2017

- 24h des IUT

lundi 22/05/2017

- Essais de modifications structurelles avec Fiona

mardi 23/05/2017

- Réunion discussion pour terminer la spécification du fonctionnement des fichiers de configuration
- Algorithme non testé de modifications structurelles des entrées en fonction des configurations

mercredi 24/05/2017

- Modifications structurelles selon config terminées et testées
- Clarification des logs

jeudi 25/05/2017

- IFSTTAR fermé

vendredi 26/05/2017

- IFSTTAR fermé

lundi 29/05/2017

- Conversion du excel vers fichier de configuration

mardi 30/05/2017

- Prise de connaissance du fonctionnement des schémas sous PostgreSQL
- Réalisation de la gestion automatique des schémas

mercredi 31/05/2017

- Résolution d'un problème avec les schémas
- Requêtes CREATE et INSERT fonctionnelles

jeudi 01/06/2017

- Tests et et listage/résolution des problèmes rencontrés
- Détermination des améliorations sur la structure, le nommage, et la documentation à effectuer

vendredi 02/06/2017

- Rédaction d'un manuel d'utilisation

lundi 05/06/2017

- IFSTTAR fermé

mardi 06/06/2017

- Début de patron LaTeX pour rapport

mercredi 07/06/2017

- Premier jet de structure sémantique du rapport

jeudi 08/06/2017

- Rédaction du rapport

vendredi 09/06/2017

- Rédaction du rapport

lundi 12/06/2017

- Rédaction du rapport

mardi 13/06/2017

- Rédaction du rapport

mercredi 14/06/2017

- Rédaction du rapport

jeudi 15/06/2017

- Relecture rapport et corrections mineures (numérotation, typos)
- Réalisation des diapositives pour la présentation

vendredi 15/06/2017

- Présentation du stage à l'IFSTTAR
- Révision des diapositives