25/05/2020

Thomas Tissot

CFPT

Documentation Technique

Cadastre Parcelle mensuration

# Table des matières

[1. Table des matières 1](#_Toc42521197)

[2. Introduction 3](#_Toc42521198)

[3. Méthodologie 3](#_Toc42521199)

[4. Résumé du cahier des charges 3](#_Toc42521200)

[4.1 But de l’application 3](#_Toc42521201)

[4.2 Matériel et logiciel nécessaires 3](#_Toc42521202)

[5. Analyse fonctionnelle 3](#_Toc42521203)

[5.1 Interfaces 3](#_Toc42521204)

[5.1.1 Fonctions des menus 3](#_Toc42521205)

[5.1.2 Menu 4](#_Toc42521206)

[5.1.3 Page d’accueil 4](#_Toc42521207)

[5.1.4 Page modifier parcelle 5](#_Toc42521208)

[5.2 Liste des fonctionnalités disponibles 5](#_Toc42521209)

[5.3 Description des fonctionnalités 5](#_Toc42521210)

[5.3.1 Importer un fichier CSV 5](#_Toc42521211)

[5.3.2 Exporter un fichier CSV 6](#_Toc42521212)

[5.3.3 Filtrer les parcelles 6](#_Toc42521213)

[5.3.4 Trier les parcelles 6](#_Toc42521214)

[5.3.5 Sélectionner des parcelles 6](#_Toc42521215)

[5.3.6 Modifier une parcelle 7](#_Toc42521216)

[5.3.7 Créer des ordres 7](#_Toc42521217)

[6. Analyse Organique 8](#_Toc42521218)

[6.1 Technologie utilisé 8](#_Toc42521219)

[6.2 Environnement 8](#_Toc42521220)

[6.3 Modèle logique de données 9](#_Toc42521221)

[6.4 Description des tables 9](#_Toc42521222)

[6.4.1 Table cadparcellemensu 9](#_Toc42521223)

[6.4.2 Table geometre 10](#_Toc42521224)

[6.4.3 Table etats 10](#_Toc42521225)

[6.4.4 Table ordres 10](#_Toc42521226)

[6.5 Diagrammes de classe 11](#_Toc42521227)

[6.6 Description détailler des méthodes 11](#_Toc42521228)

[6.6.1 Class ConnexionDb 11](#_Toc42521229)

[6.6.2 Classe UtilCsv 11](#_Toc42521230)

[6.6.3 Classe UtilDataGrid() 12](#_Toc42521231)

[6.6.4 Classe UtilDb() 12](#_Toc42521232)

[6.6.5 Classe UtilLog() 13](#_Toc42521233)

[6.7 Outils externes 13](#_Toc42521234)

[6.7.1 Draw.Io 13](#_Toc42521235)

[6.7.2 GitHub 13](#_Toc42521236)

[7. Réalisation 13](#_Toc42521237)

[7.2 Arborescence du Projet 13](#_Toc42521238)

[8. Tests 13](#_Toc42521239)

[8.1 Tests unitaires 13](#_Toc42521240)

[8.1.1 ConnexionDbTest 13](#_Toc42521241)

[8.1.2 UtilDbTest 14](#_Toc42521242)

[UtilDataGridTest 15](#_Toc42521243)

[8.2 Tests manuels 15](#_Toc42521244)

[8.2.1 UtilDb 15](#_Toc42521245)

[8.2.2 UtilDataGrid 16](#_Toc42521246)

[9. Table des illustrations 16](#_Toc42521247)

# Introduction

Cette documentation a pour but de détailler en plusieurs partie mon application C# « Cadastre parcelle mensuration », réaliser dans le cadre du TPI (Travail pratique individuel) pour valider mon CFC d’informaticien.

Le premier jour de mon TPI j’ai reçu le cahier des charges de mon application par mail. Pour ma part je dois faire une application Windows Form qui permettra de visualiser, filtrer et modifier les données du cadastre importer avec un fichier CSV que mon application ajoutera dans la base de données. On peut également exporter les données de la base de données en fichier CSV.

# Méthodologie

J’ai choisi d’utiliser la méthodologie « WaterFall ». Celle-ci consiste une succession d’étapes, chacune d'entre elles menant à la suivante. Voici les 6 étapes de cette méthodologie :

* Exigences (analyse des besoins du mandant)
* Conception
* Planning
* Mise en œuvre
* Vérification
* Maintenance

Dans mon cas je ne vais pas utiliser la dernière étape dans le cadre mon projet.

# Résumé du cahier des charges

## 4.1 But de l’application

Le but de l’application est de permettre à l’utilisateur d’importer des données a partir d’un fichier csv et de visualiser ces données. Il sera aussi possible de filtrer et trié les données du tableau, l’utilisateur pourra exporter les données en fichier CSV tout en gardant les filtres. Il pourra également modifier certains champs.

## 4.2 Matériel et logiciel nécessaires

* Ordinateur (PC)
* Visual Studio 2019 – Langage C#
* GitHub Desktop
* Serveur EasyPHP

# Analyse fonctionnelle

Cette section va traiter la partie visible de l’application, donc ce que verra l’utilisateur final.

## Interfaces

### 5.1.1 Fonctions des menus

* Fichier -> Importer CSV fichier renvoi sur un OpenFileDialog pour sélectionner le fichier que l’utilisateur veut importer.
* Fichier -> Exporter CSV renvoi sur un SaveFileDialog pour sélectionner l’emplacement ou l’utilisateur veut enregistrer le CSV.
* Aide -> Liste des raccourcis renvoie sur une page qui affiche tous les raccourcis clavier présent dans mon application.
* Aide -> A propos renoi sur la page a propos de mon application

### 5.1.2 Menu

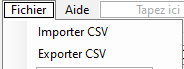


Figure - Menu Fichier

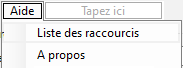


Figure - Menu Aide

### 5.1.3 Page d’accueil

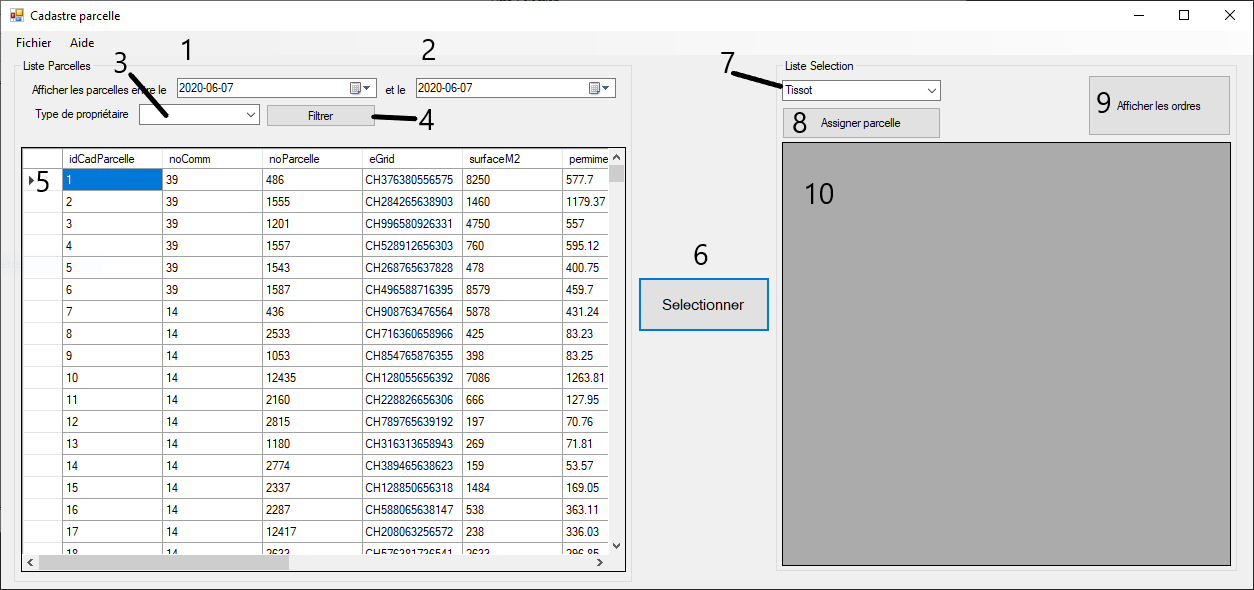


Figure - Page d'accueil

1. L’utilisateur peut sélectionner la date de début de plage pour filtrer les parcelles.
2. L’utilisateur peut sélectionner la date de fin de plage pour filtrer les parcelles.
3. L’utilisateur peut sélectionner le type de propriété pour filtrer les parcelles.
4. L’utilisateur clique sur **Filtrer** pour appliquer les filtres choisis. (Eléments numéro 1, 2 et 4) Et reclique dessus pour les annuler.
5. L’utilisateur voit les parcelles filtrer par l’utilisateur ou toute celles présentent dans la base de données.
6. L’utilisateur clique sur le bouton **Sélectionner** pour déplacer les parcelles sélectionner dans la liste Parcelle dans la liste Sélection.
7. L’utilisateur peut sélectionner le géomètre a qui l’utilisateur veut assigner les parcelles présente dans la liste Sélection.
8. L’utilisateur clique sur le bouton **Assigner parcelle** pour assigner les parcelles déplacer dans la liste Sélection au géomètre sélectionné dans l’élément numéro 7.
9. L’utilisateur clique sur le bouton **Afficher les ordres** pour ouvrir la page de visualisation des ordres présent dans la base de données.
10. Liste Sélection permet d’afficher toutes les parcelles que l’utilisateur à sélectionner.

### 5.1.4 Page modifier parcelle

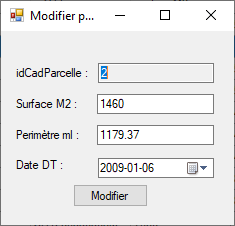


Figure - Page modification parcelle

Ce formulaire permet de modifier 3 valeur d’une parcelle pour valider les modification il suffit de cliquer sur le bouton « Modifier ». Le MessageBox ci-dessous s’affiche si la requête a été exécute avec succès.

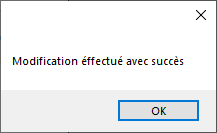


Figure - MessageBox modification parcelle

## 5.2 Liste des fonctionnalités disponibles

* Importer un fichier CSV
* Exporter un fichier CSV
* Filtrer des parcelles
* Trier les parcelles
* Sélectionner des parcelles
* Modifier une parcelle
* Créer des ordres

## Description des fonctionnalités

Les prérequis correspondent aux étapes nécessaires pour pouvoir accéder à la fonctionnalité. Les interfaces quand a elles correspondent à la vue sur laquelle les fonctionnalités sont disponible.

### 5.3.1 Importer un fichier CSV

**Prérequis :** Avoir extrait un fichier CSV avec l’outils d’extraction du cadastre genevois

**Interface :** Page d’accueil

L’utilisateur peut importer un fichier CSV, pour cela il doit cliquer sur le menu Fichier –> Importer un CSV. Ensuite un page de l’explorateur de fichier va s’ouvrir pour que l’utilisateur sélectionne le fichier CSV qu’il veut importer. Un fichier texte se créer lors de l’import qui répertorie les erreurs que l’import a rencontré.

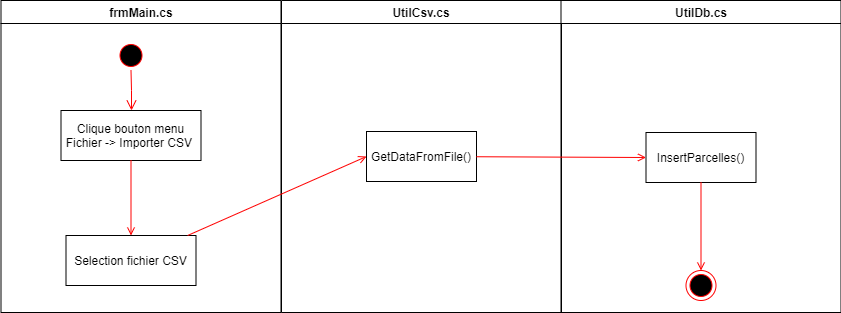


Figure - Diagramme Importer CSV

### 5.3.2 Exporter un fichier CSV

**Prérequis :** Avoir importer des parcelles dans la base de données

**Interface :** Page d’accueil

L’utilisateur peut exporter les parcelles présente dans le DataGridView, c’est pourquoi l’utilisateur peut filtrer ce qu’il veut exporter.

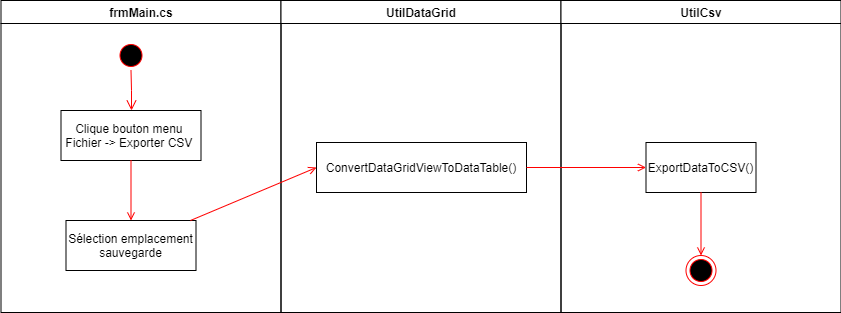


Figure - Diagramme Exporter CSV

### 5.3.3 Filtrer les parcelles

**Prérequis :** Avoir importer des parcelles dans la base de données

**Interface :** Page d’accueil

L’utilisateur peut filtrer les parcelles en choisissant une plage de date et un type de propriété

### 5.3.4 Trier les parcelles

**Prérequis :** Avoir importer des parcelles dans la base de données

**Interface :** Page d’accueil

L’utilisateur peut trier des parcelles en cliquant sur n’importe qu’elles colonnes présente dans toute les DataGridView, cela garde les filtre mis en place.

### 5.3.5 Sélectionner des parcelles

**Prérequis :** Avoir importer des parcelles dans la base de données

**Interface :** Page d’accueil

L’utilisateur peut sélectionner des enregistrements dans le DataGridView en cliquant sur une ligne ou plusieurs en maintenant la touche contrôle ensuite en cliquant sur le bouton sélectionner les enregistrements se déplaceront dans la DataGridView de droite.

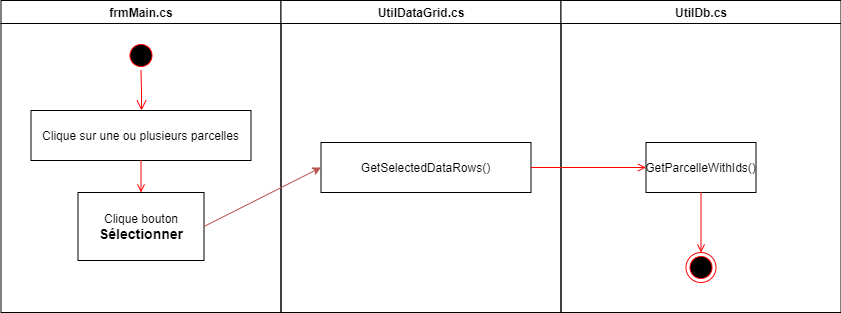


Figure - Diagramme Sélectionner parcelle

### 5.3.6 Modifier une parcelle

**Prérequis :** Avoir importer des parcelles dans la base de données

**Interface :** Page modifier parcelle

L’utilisateur peut modifier en double cliquant sur une parcelle dans la liste, cela ouvrira la page modifier. L’utilisateur a la possibilité de modifier la surface m2, le périmètre ml et la dateDt.

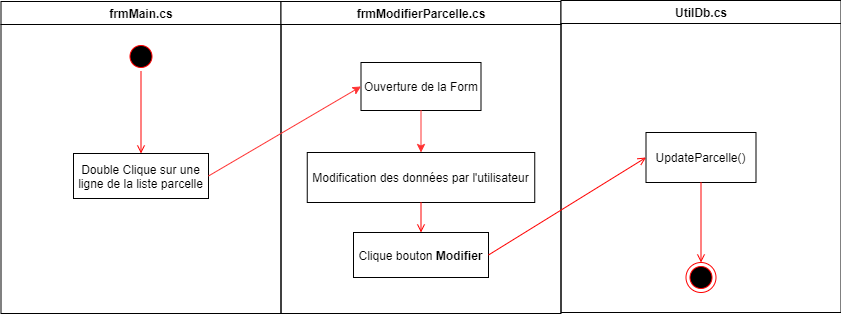


Figure - Diagramme Modification parcelle

### 5.3.7 Créer des ordres

**Prérequis :** Avoir sélectionner une ou plusieurs parcelle

**Interface :** Page d’accueil

L’utilisateur peut créer des ordres en sélectionnant l’opérateur désirer et en cliquant sur « Assigner parcelle ». Cela créera un ordre par parcelle présente dans la liste sélection.

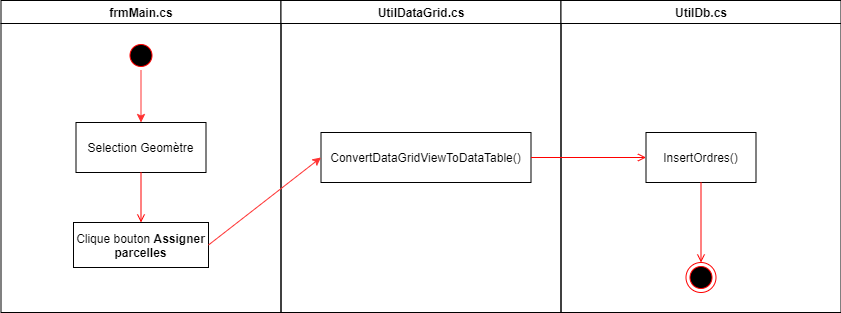


Figure - Diagramme Créer ordres

# Analyse Organique

Cette partie de la documentation portera sur le fonctionnement interne de l’application, cette partie est vu exclusivement par le développeur. Pour que cette partie soit le plus clair possible je vais séparer les méthodes par class.

## 6.1 Technologie utilisé

* C#
* MySql

## 6.2 Environnement

Pour le développement de mon application, j’ai utilisé Visual Studio Community 2019.

J’utilise EasyPhp 14.1 comme serveur pour héberger ma base de données. Et pour créer et modifier ma base de données j’ai utilisé PhpMyAdmin qui est fourni directement dans EasyPhp.

J’ai sauvegardé mon projet chaque soir sur mon disque dur personnel et j’ai également utilisé GitHubDesktop.

Pour la documentation, j’ai utilisé Microsoft Word et Microsoft Excel.

En se qui concerne les tests j’ai créé un projet de test dans ma solution Visual Studio qui me permet de faire des tests unitaires.

## 6.3 Modèle logique de données

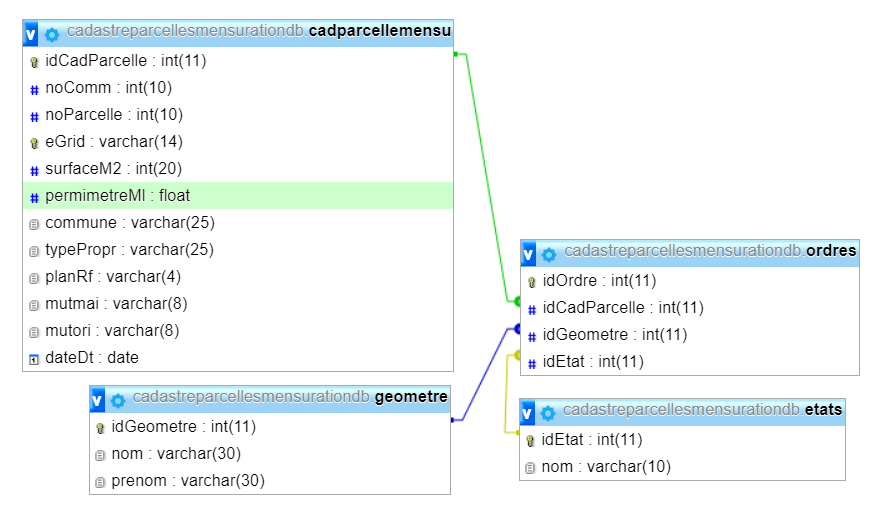


Figure - Modèle logique de données

Toutes les tables ont été pensé et créer par moi-même, en relisant plusieurs fois mon cahier des charges j’ai trouvé que c’était la façon la plus simple et claire pour moi.

La taille de certains champs sont très precis (par exemple le eGrid) car sur le site du cadastre genevois j’ai réussi a trouver la longueur exacte ou maximum de plusieurs champs. Malheureusement certains champs n’étais pas précise donc j’ai assigné la taille moi même.

## 6.4 Description des tables

### 6.4.1 Table cadparcellemensu

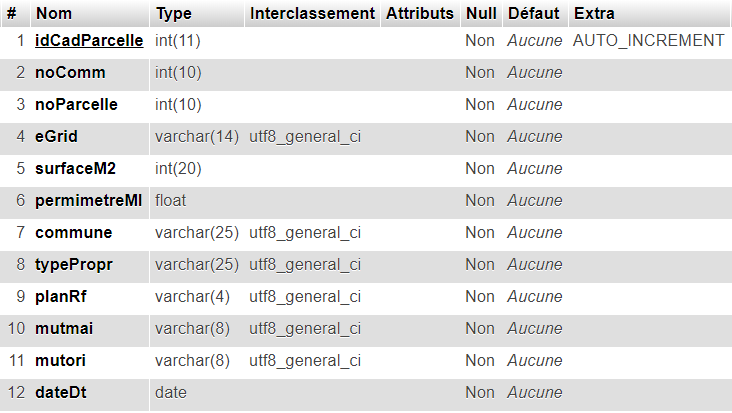


Figure - Table cadparcellemensu

Cette table contient toutes les parcelles importer par l’utilisateur. Les champs ont des nom en peu spécial mais garder les même noms que dans le fichier CSV est pour la chose la plus simple a faire pour que je m’y retrouve facilement.

### 6.4.2 Table geometre

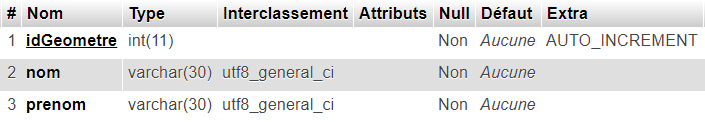


Figure - Table geometre

Cette table contient le géomètre à qui on peut assigner une ou plusieurs parcelles. Tous les enregistrements dans cette table ont été ajouter manuellement par moi.

### 6.4.3 Table etats

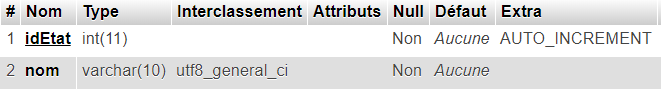


Figure - Table etats

Cette table contient 2 états diffèrent qui me permettront de savoir si un ordre est en cours ou terminé.

### 6.4.4 Table ordres

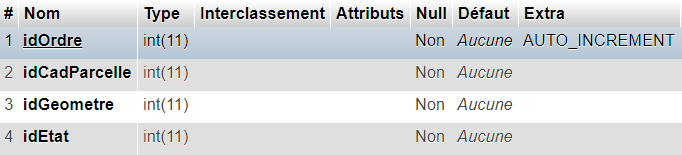


Figure - Table ordres

Cette table contient tous les ordres crée par l’utilisateur. Les champs idCadParcelle, idGeometre et idEtat sont les 3 des clés étrangère récupérer dans mes 3 autre tables.

## 6.5 Diagramme de classes

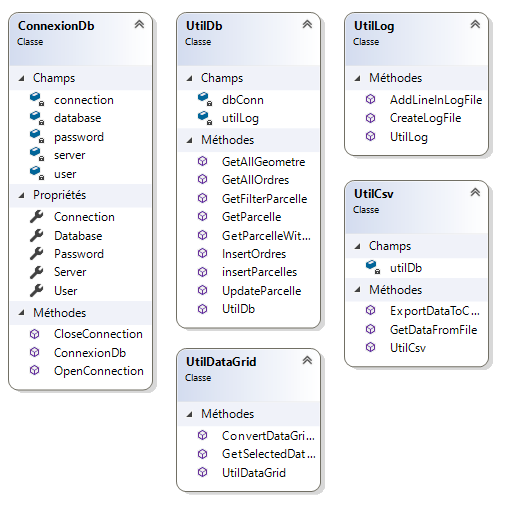


Figure - Digramme de classes

## 6.6 Description détailler des méthodes

### 6.6.1 Class ConnexionDb

Cette classe permet d’ouvrir et de fermer la connexion avec la base de données.

Pour être instanciée elle a besoin de plusieurs paramètres :

* L’adresse du serveur (Dans mon cas 127.0.0.1)
* Le nom de la base de données
* Le nom de l’utilisateur
* Le mot de passe de l’utilisateur

#### Méthode OpenConnection()

Cette méthode publique a pour but d’ouvrir la connexion avec la base de données. Elle retourne un bool pour savoir si l’application a rencontré un bug lors de la connexion.

#### Méthode CloseConnection()

Cette méthode publique a pour but de fermer la connexion avec la base de données. Elle retourne un bool pour savoir si l’application a rencontré un bug lors de la déconnexion.

### 6.6.2 Classe UtilCsv

Cette classe contient toutes les méthodes que j’utilise pour importer ou exporter des fichier CSV.

#### Méthode GetDataFromFile()

Cette méthode publique a pour but de créer une DataTable à partir d’un fichier CSV. Elle prend comme paramètre le chemin d’accès du fichier csv que l’utilisateur a sélectionné.

#### Méthode ExportDataToCSV()

Cette méthode publique a pour but de convertir une DataTable en fichier CSV. Elle prend comme paramètres une DataTable qui contient les informations des parcelles et un string qui contient le chemin d’accès où le fichier CSV va s’enregistrer.

### 6.6.3 Classe UtilDataGrid()

Cette classe permet de gérer les DataGridView présente dans mon application.

#### Méthode ConvertDataGridViewToDataTable()

Cette méthode publique a pour but de convertir les parcelles présente dans mon DataGridView en DataTable.

#### Méthode GetSelectedDataRows(DataGridView grid)

Cette méthode publique récupère en paramètre le DataGridView parcelle, elle a pour but de retourner tous les ids des parcelles sélectionner par l’utilisateur.

### 6.6.4 Classe UtilDb()

Cette classe contient toutes les requêtes SQL.

#### Méthode GetParcelle()

Cette méthode publique a pour but de récupérer toute la parcelle présente dans la base de données et les convertis en type DataSet.

#### Méthode GetFilterParcelle()

Cette méthode publique récupère en paramètre une date de début et une de fin préalablement convertie en string ainsi qu’un type de propriété, elle a pour but de récupéré toute les parcelles qui corresponde au filtre dans la base de données.

#### Méthode InsertParcelles()

Cette méthode publique a pour but de récupérer en paramètre une DataTable qui contient les parcelle importer par l’utilisateur et les insérer dans la base de données.

#### Méthode GetParcelleWithIds()

Cette méthode publique a pour but de récupérer en paramètre une liste d’ids qui correspond à chaque parcelle sélectionner par l’utilisateur dans le DataGridView. Ensuite cette méthode fait un select sur la base de données pour récupérer chaque parcelle correspondant au ids de la liste.

#### Méthode GetAllGeometre()

Cette méthode publique a pour but de récupérer tous les Géomètres présents dans la table Géomètre.

#### Méthode GetAllOrdres()

Cette méthode publique a pour but de récupérer tous les ordres présents dans la table Ordres.

#### Méthode InsertOrdres()

Cette méthode publique récupère en paramètre un idGeomtre ainsi qu’une DataTable préalablement convertis de puis le DataGridView Selection, Elle a pour but de créer un ordre par parcelle présente dans la DataTable.

#### Méthode UpdateParcelle()

Cette méthode publique récupère en paramètre un id, une date, le périmètre et la surface, elle a pour but de modifier la parcelle qui correspond a l’id récupérer en paramètre.

### 6.6.5 Classe UtilLog()

Cette classe contient toutes les classes utiles à créer et écrire dans le fichier texte log a chaque import de CSV.

#### Méthode CreateLogFile()

Cette méthode a pour but de récupérer en paramètre une liste de string qui contient les erreurs lors de l’import ainsi qu’une variable de type int qui contient le nombre total d’erreur. Ensuite cette méthode créer un fichier texte dans le répertoire du projet et ajoute le nombre d’erreur ainsi que chaque erreur que l’import a rencontrée.

#### Méthode AddLineInLogFile()

Cette méthode a pour but de récupérer en paramètre un StreamWriter qui permettra d’écrire dans le fichier log ainsi qu’une variable de type string qui contient la ligne à écrire dans le fichier log.

## 6.7 Outils externes

### 6.7.1 Draw.Io

Draw.io est un site web qui permet de créer des diagrammes en tout genre. Dans le cadre de mon projet j’ai fait des digrammes de séquences.

### 6.7.2 GitHub

GitHub est un service web d’hébergement et de gestion de développement de logiciel, utilisant le logiciel de gestion de versions Git.

J’ai utilisé cet outil pour sauvegarder mon projet régulièrement. En cas de bug bloquant cet outil m’aurait permis de revenir sur une version antérieur propre et sans bug.

# Réalisation

## 7.2 Arborescence du Projet

# Tests

La solution contient un projet de tests unitaires. C’est pourquoi toute les classes sont publiques. Dans certain cas les tests unitaire étais complique étant donné que je n’ai pas un environnement dédié aux tests certaines méthodes ont été tester manuellement.

## 8.1 Tests unitaires

### 8.1.1 ConnexionDbTest

#### Test constructeur

[TestMethod]

public void ConstructorShouldCreateConnexionDbCorrectly()

{

ConnexionDb conn = new ConnexionDb("localhost", "CadastreParcellesMensurationDb", "tissott", "Super");

ConnexionDb target = conn;

Assert.IsNotNull(target);

Assert.AreEqual(target.Server, "localhost");

Assert.AreEqual(target.Database, "CadastreParcellesMensurationDb");

Assert.AreEqual(target.User, "tissott");

Assert.AreEqual(target.Password, "Super");

}

#### Test OpenConnection()

[TestMethod]

public void OpenConnectionShouldReturnTrue()

{

ConnexionDb conn = new ConnexionDb("localhost", "CadastreParcellesMensurationDb", "tissott", "Super");

bool target = conn.OpenConnection();

Assert.AreEqual(target, true);

}

#### Test CloseConnection()

[TestMethod]

public void CloseConnectionShouldReturnTrue()

{

ConnexionDb conn = new ConnexionDb("localhost", "CadastreParcellesMensurationDb", "tissott", "Super");

conn.OpenConnection();

bool target = conn.CloseConnection();

Assert.AreEqual(target, true);

}

### 8.1.2 UtilDbTest

#### Test constructeur

[TestMethod]

public void ConstructorShouldCreateUtilDbCorrectly()

{

UtilDb target = new UtilDb();

Assert.IsNotNull(target);

}

#### Test GetParcelle()

[TestMethod]

public void GetParcelleShouldReturnAllParcelle()

{

UtilDb utilDb = new UtilDb();

DataSet target = new DataSet();

target = utilDb.GetParcelle();

Assert.IsNotNull(target.Tables[0]);

Assert.AreEqual(target.Tables[0].Columns.Count, 12);

}

#### Test GetFilterParcelle()

[TestMethod]

public void GetFilterParcelleShouldReturnParcelle()

{

UtilDb utilDb = new UtilDb();

DataSet target = new DataSet();

target = utilDb.GetFilterParcelle("2000-01-01","2020-01-01","privé");

Assert.IsNotNull(target.Tables[0]);

foreach (DataRow item in target.Tables[0].Rows)

{

Assert.AreEqual(item["typePropr"], "privé");

}

}

#### Test GetParcelleWithIds()

[TestMethod]

public void GetParcelleWithIdsShouldReturnParcelleWithSameId()

{

UtilDb utilDb = new UtilDb();

List<int> listId = new List<int>();

listId.Add(152);

DataSet target = utilDb.GetParcelleWithIds(listId);

Assert.IsNotNull(target.Tables[0]);

Assert.AreEqual(target.Tables[0].Rows[0].ItemArray[0], 152);

}

#### Test GetAllGeometre()

[TestMethod]

public void GetAllGeometreShouldReturnAllGeometre()

{

UtilDb utilDb = new UtilDb();

DataSet target = new DataSet();

target = utilDb.GetAllGeometre();

Assert.IsNotNull(target.Tables[0]);

Assert.AreEqual(target.Tables[0].Columns.Count, 3);

}

#### Test GetAllOrdres()

[TestMethod]

public void GetAllOrdresShouldReturnAllOrdres()

{

UtilDb utilDb = new UtilDb();

DataSet target = new DataSet();

target = utilDb.GetAllOrdres();

Assert.IsNotNull(target.Tables[0]);

Assert.AreEqual(target.Tables[0].Columns.Count, 4);

}

### UtilDataGridTest

#### Test constructeur

[TestMethod]

public void ConstructorShouldCreateUtilDataGridCorrectly()

{

UtilDataGrid utilDataGrid = new UtilDataGrid();

Assert.IsNotNull(utilDataGrid);

}

#### Test ConvertDataGridViewToDataTable()

[TestMethod]

public void ConvertDataGridViewToDataTableShouldReturnDataTable()

{

UtilDataGrid utilDataGrid = new UtilDataGrid();

DataGridView dataGridView = new DataGridView();

DataTable target = utilDataGrid.ConvertDataGridViewToDataTable(dataGridView);

Assert.IsNotNull(target);

}

## 8.2 Tests manuels

### 8.2.1 UtilDb

#### InsertParcelle()

**Résultat attendu :** Ajout des parcelles dans la base de données.

**Résultat :** Ok

#### InsertOrdres()

**Résultat attendu :** Ajout des ordres dans la base de données.

**Résultat :** Ok

#### UpdateParcelle()

**Résultat attendu :** Modification de certains champs d’une parcelle.

**Résultat :** Ok

### 8.2.2 UtilDataGrid

#### GetSelectedDataRows()

**Résultat attendu :** Retourne une liste contenant les ids des parcelles.

**Résultat :** Ok

# Table des illustrations

[Figure 1 - Menu Fichier 4](#_Toc42508054)

[Figure 2 - Menu Aide 4](#_Toc42508055)

[Figure 3 - Page d'accueil 4](#_Toc42508056)

[Figure 4 - Modèle logique de données 7](#_Toc42508057)

[Figure 5 - Table cadparcellemensu 8](#_Toc42508058)

[Figure 6 - Table geometre 8](#_Toc42508059)

[Figure 7 - Table etats 8](#_Toc42508060)

[Figure 8 - Table ordres 9](#_Toc42508061)

[Figure 9 - Diagramme Importer CSV 11](#_Toc42508062)

[Figure 10 - Diagramme Exporter CSV 11](#_Toc42508063)

[Figure 11 - Diagramme Sélection parcelle(s) 11](#_Toc42508064)

[Figure 12 - Diagramme Création ordre(s) 12](#_Toc42508065)

[Figure 13 - Diagramme Modifier parcelle 12](#_Toc42508066)