Dossier d’architecture

Damien Chaudois

4 AL 1

2018

Intro

Ce dossier d'architecture décrit le logiciel Kontact, un projet fictif ayant pour but de répondre à la demande du professeur de design pattern de produire un logiciel contenant 5 patterns différents. Contrairement à un projet standard où l'architecture technique permet de répondre à un besoin, ici c'est l'inverse qui a été mis en place : l'objectif du projet a été décider pour permettre d'implémenter des techniques/patterns demandés. La portée de ce projet étant minime et insuffisante pour justifier un dossier d'architecture, certaines fonctionnalités décrites seront fictives et leur périmètre sera anticipé plutôt qu'analysé.

Table des matières

Enjeux, objectifs, contraintes

Comme énoncé plus haut, les objectifs de fonctionnalités sont déduits en fonction des demandes de réalisations techniques (6 design patterns), mais ceux-ci doivent être cohérents Il a donc été décidé de créer un logiciel de gestion de carnet d'adresses, nommé Kontact, puisqu'il semblait simple d'y appliquer des patterns génériques que l'on peut observer dans n'importe quel logiciel. L'objectif est donc d'implémenter 6 patterns :

- Le singleton
- Le composite
- Le prototype
- L'observer
- L'itérateur
- Le Template

Le sujet nous impose également de travailler en C#. La nature du logiciel, ses fonctionnalités, et les techniques additionnels au C# sont libres.

L'objectif ici n'est donc pas de coder pour satisfaire des besoins, mais de trouver des besoins pour satisfaire des contraintes techniques. Les fonctionnalités suivantes répondent ainsi aux contraintes correspondantes :

Le singleton	L'application enregistre ses données dans une base SQLite. L'accès à cette base se fait au travers d'une classe statique
Le composite	L'application est pensée pour être modulaire : il est possible de rajouter des tables à la base, il suffira alors de faire implémenter l'interface d'accès de données commune à tous les objets représentant les tables
Le prototype	Les contacts peuvent être dupliqués : pour permettre ceci l'objet de gestion de données du contact peut être cloné
L'observer	Les contacts sont symbolisés par un bouton. Lors d'un appui sur celui-ci, le panneau d'information apparaît. Si le nom ou le prénom est modifié, le bouton est mis à jour au travers du pattern observer
L'itérateur	Les prénoms et nom appartiennent à un contact mais ses traités comme des champs dans le panneau d'information. Pour permettre d'itérer sur ceux-ci, l'objet de gestion des contacts possède sa propre gestion d'itération grâce à son implémentation de IIterator
Le Template	L'import/export est géré par deux classes qui implémentent une interface commune qui permet d'importer et d'exporter sous différents formats, comme CSV ou JSON

Environnement

Kontact étant une application client lourd codé en C#, l'utilisateur doit répondre à certaines contraintes. En considérant qu'il faut utiliser un OS Microsoft :

Version de Windows	Installation du Framework .net 4.0 ou plus requis	Configuration minimum
Windows 7	Oui	1Go RAM 1GHz CPU
Windows 8 + 8.1	Non	1Go RAM 1GHz CPU
Windows 10	Non	1Go RAM 1GHz CPU Carte graphique avec DirectX 9

L'espace disque utilisé étant négligeable il n'est pas cité ici étant de l'ordre de 10Ko. Une connexion à internet n'est pas requise pour utiliser Kontact, seulement si on veut garder le client à jour.

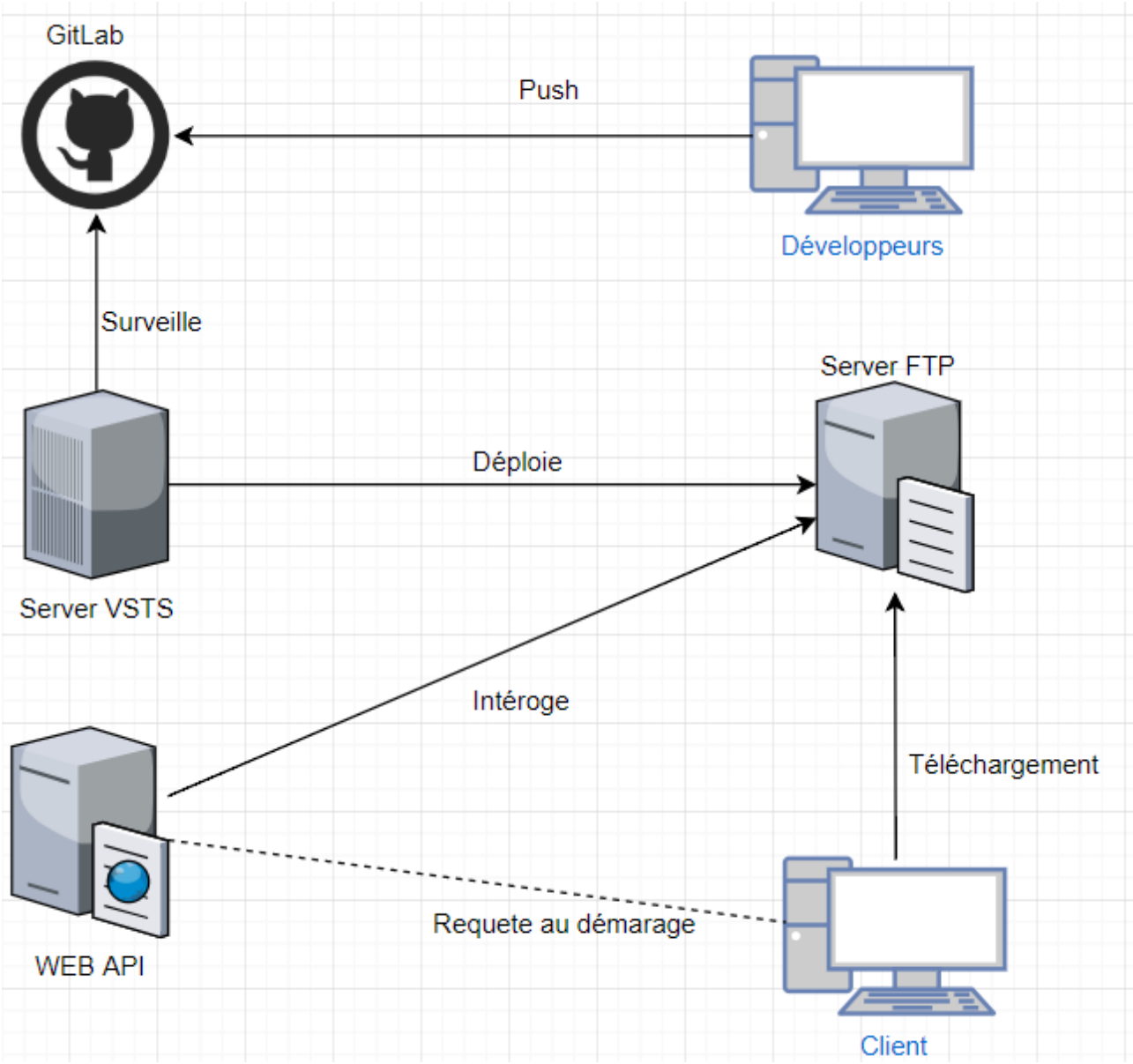
Déploiement

L'application a pour forme un fichier exécutable. L'installation se fait simplement en téléchargeant le client ou en copiant le fichier .exe. L'application ira créer la base de sauvegarde dans le dossier d'exécution si celui-ci n'existe pas déjà. Le déploiement se fait automatiquement via VSTS (la version Microsoft de Jenkins) : dès qu'un push est fait sur la branche master du repository du GitLab du serveur VSTS, celui-ci va immédiatement déclencher un pull sur le serveur FTP qui sera plus tard utilisé pour télécharger les éventuelles mises à jour.

Mise à jour

Le client possède un system de mise à jour pour que l'utilisateur conserve toujours la dernière version. Au démarrage, Kontact ira interroger un service web qui lui donnera la dernière version déployée. Si cette version est supérieure à la sienne, il proposera à l'utilisateur de télécharger et d'installer cette nouvelle version. Il ira donc copier le nouveau client depuis le serveur ftp et remplacera l'ancienne version par la nouvelle.

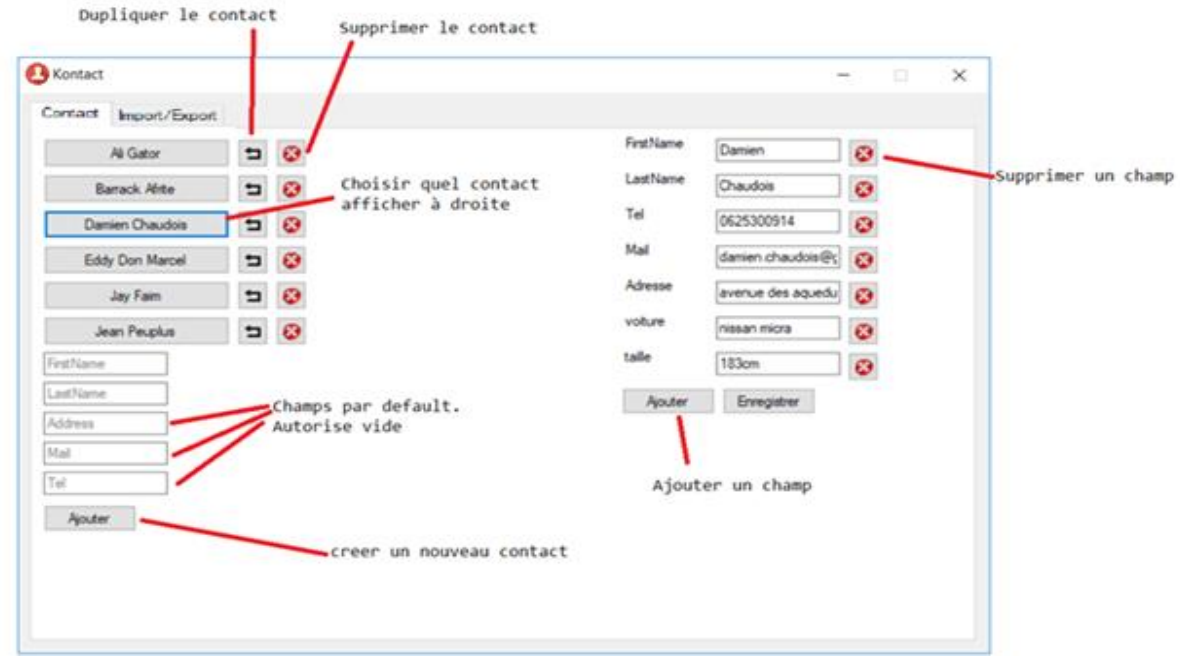
La stratégie de déploiement/Mise à jour peut être décrite comme suit :

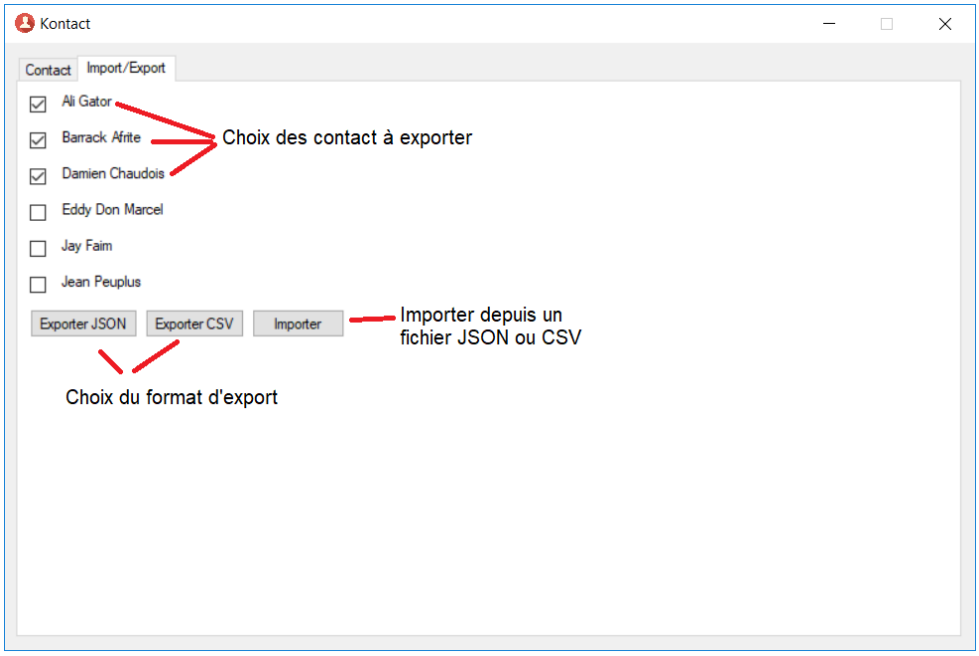


Le serveur web, VSTS, GitLab, et FTP sont tous situés sur le même serveur physique. Le serveur GitHub étant public, il est accessible depuis l'URL suivante :

https://github.com/chaudois/projetDesignPaternG16/

L'interface





Depuis l'interface, l'utilisateur peut effectuer les actions suivantes :

- Créer un nouveau contact
- Supprimer un contact existant
- Dupliquer un contact existant
- Rajouter et supprimer autant de champs que voulue par contact
- Exporter des contacts sélectionnés ou importer un fichier, au format CSV ou JSON

Les différents composants du logiciel

Le logiciel se décompose en quatre parties :

- La partie mise à jour via des services web
- L'exécutable
- La sauvegarde SQL
- L'Export/Import via des fichier CSV ou JSON

La partie web ayant été décrite plus haut (et totalement fictive), je me concentrerais ici sur les trois derniers points.

L'exécutable

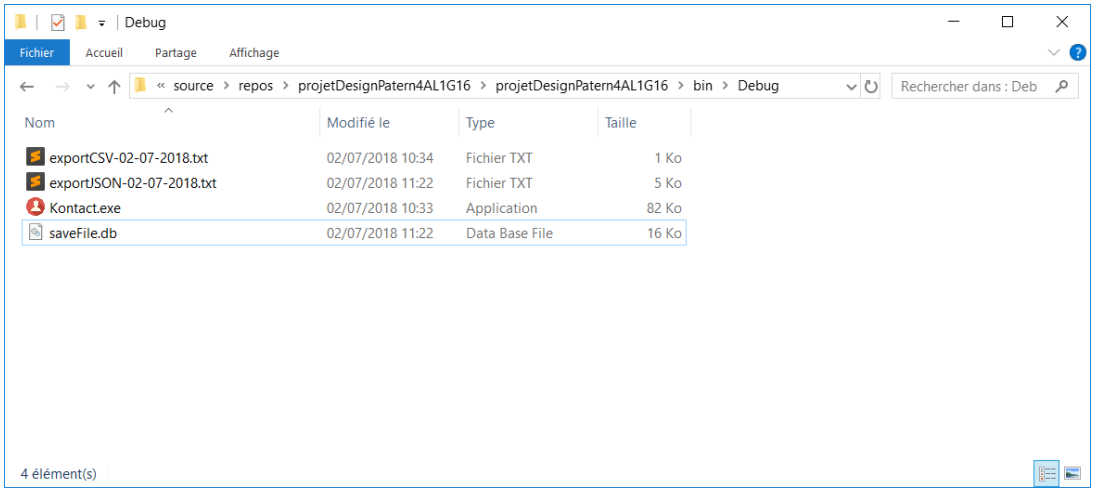
Le fichier .exe de l'application Kontact peut être placé partout sur l'ordinateur. Il requiert peu de RAM, compter 500 Ko par contact présent dans l'application. Il ne consomme de la puissance CPU que lors de l'appui sur les boutons et sa consommation est proportionnelle à la banque de contact.

La sauvegarde SQL

L'utilisation de l'application génère immédiatement un fichier de sauvegarde saveFile.db pour Sqlite3. À chaque démarrage de l'application, celle-ci s'y connecte ou le crée pour y sauvegarder toutes les informations requises. La taille du fichier est négligeable, compter entre 16 et 32 Ko.

Export/Import

L'application peut à tout moment permettre d'exporter des contacts présélectionnés au format voulu par l'utilisateur : CSV ou JSON. Dans ce cas, l'application va générer un fichier txt contenant toutes les informations des contacts et de leurs champs au format voulu dans le répertoire d'exécution et immédiatement l'afficher à l'utilisateur.



Justification des choix

Comme énoncé plus haut, les choix technologiques sont le plus souvent imposé et le gros des décisions concerne quelle fonctionnalité peut être mise en place pour implémenter une contrainte. Il reste cependant quelques libertés que je décrirai ici.

L'utilisation d'une WinForm

WinForm est le Framework de base offert par C# pour créer facilement des interfaces graphiques pour les applications client lourd. Elle permet de placer des contrôles via une interface graphique ou dynamiquement via l'exécution de code. Dans la mesure où le logiciel doit proposer des fonctionnalités qui permettent d'implémenter des patterns par définition « génériques », une architecture model-vue-Controller semblait ce qu'il y avait de plus adapté pour donner une opportunité à des contraintes d'apparaitre qui pouvaient être résolues avec des patterns adaptés.

L'utilisation de SQLite3

Pour permettre d'implémenter le pattern singleton, la fonctionnalité la plus simple sur laquelle appliqué ce pattern semblait être la persistance des données via une base de données SQL. À partir de là la question qui s'est posé est quelle version-distribution de SQL utilisé. Puisque l'application ne partage pas ses données interne avec une autre application, il n'est pas nécessaire de faire appel à un serveur SQL, local ou distant. De plus, l'installation doit être le plus simple possible pour le client, il ne doit pas avoir besoin d'installer un autre logiciel du type WAMP etc.… C'est exactement ce que propose SQLite, qui permet de créer un fichier unique possédant toutes les tables voulues pour un poids minimum. Toute la structure, procédure stocké et données sont stockées dans un seul fichier qui peut être recréé à la volée par le client sans que l'utilisateur n'ait besoin d'avoir installé quoi que ce soit au préalable.

L'export/import via des fichier texte au format CSV/JSON

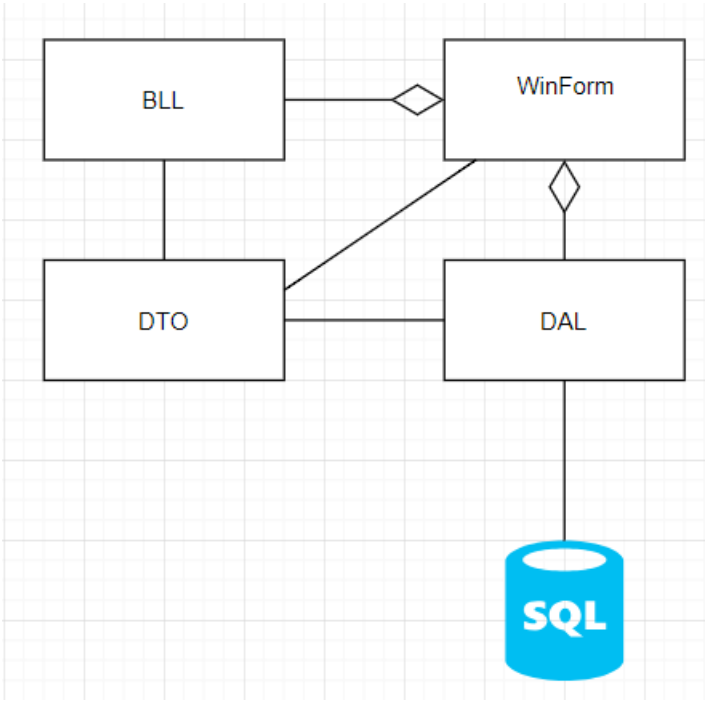
Dans l'espoir de rajouter des fonctionnalités qui sont susceptibles d'implémenter des patterns, il a été décidé de proposer l'export/import des contacts via l'interface. Bien qu'il soit déjà possible de transmettre toutes les données en faisant circuler le fichier .db qui stocke la base de données SQLite, cette fonctionnalité permet de choisir quels contacts exporter, en plus de permettre d'importer des contacts en plus de ceux déjà présents. L'extension de fichier utilisée est le txt pour que n'importe qui puisse ouvrir le fichier avec son éditeur de texte favori, et le format CSV/JSON permet à des développeurs de facilement reprendre le logiciel puisqu'il s'agit de formats standards et bien connus., en plus de permettre à d'autres applications de potentiellement interagir avec (exemple : google contact)

Description du logiciel

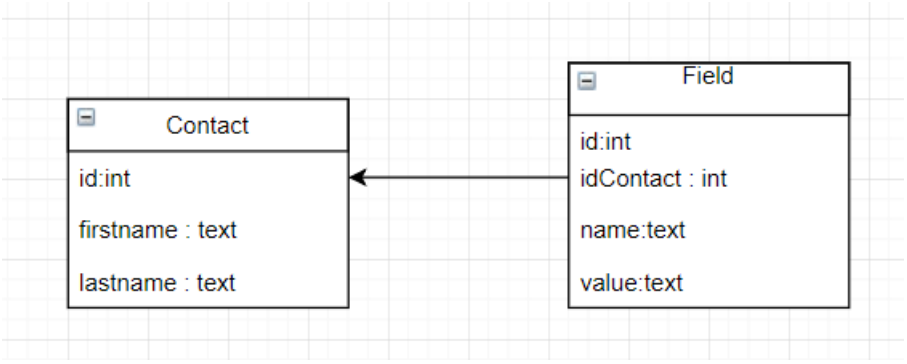
Le code

Il se découpe en 4 parties :

- La vue dans le projet WinForm
- Les model dans le projet DTO (Data transfert Object)
- Le module d'import/export (Business Logic Layout)
- Et l'accès aux données dans la DAL (Data Access Layout)



La base de données SQLite contient deux tables :



Et sont représentés par deux DTO et deux objets SQL qui permettent d'accéder à leurs données. Lors du chargement de la fenêtre, celle-ci est remplie avec autant de ligne de contact qu'il y a de contact dans la base. Au démarrage de l'application, si la base n'existe pas, elle est créée ainsi que ces tables.

WinForm

Le projet principal de l'application est WinForm qui contient deux classes : Program qui contient le Main et donc le point d'entrée de l'application, et MainWindow, qui est instancié dans la Main (). Le gros du code de l'application est contenu dans la classe MainWindows, et c'est dans cette classe que sont accessibles tous les éléments graphiques de la fenêtre. Cette class est découpé en deux fichiers grâce au mot-clé « partial ». Le premier fichier MainWindow.Designer.cs contient la déclaration et la configuration de base des contrôles de la fenêtre, et le fichier MainWindow.cs contient la définition de la fonction InitializeComponent qui est appelé dans le constructeur de la classe et donc appelé au démarrage de l'application. Une fois l'application démarré, la classe MainWindow va, au travers de la fonction InitializeComponent, pouvoir attacher des comportements à des évènements. Par exemple, la navigation entre les deux onglets déclenche un évènement qui va aller charger chacun le FlowLayoutPanel contenu dans l'onglet choisi, en faisant appel aux données de la base à chaque fois.

Le control FlowLayoutPanel est un conteneur de control qui a été choisis puisqu'il mimique le comportement du DOM HTML : chaque élément est placé à la suite, peut import la valeur de l'attribut « location » du control. Cela permet de facilement placer des control dynamiquement sans s'inquiéter de leur position dans le layout. À partir de là, l'algorithme de chargement des données est très simple :

Récupérer la liste des contacts

Pour chaque contact

    Créer un bouton avec le nom du contact en texte

    L'ajouter au FlowLayoutPanel

    Ajouter le bouton supprimer et dupliquer

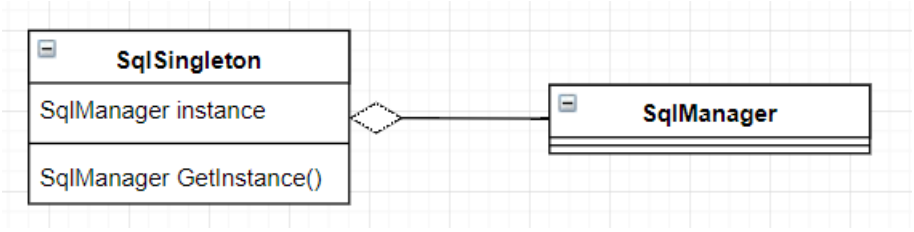
    Sauter une ligne

Recommencer

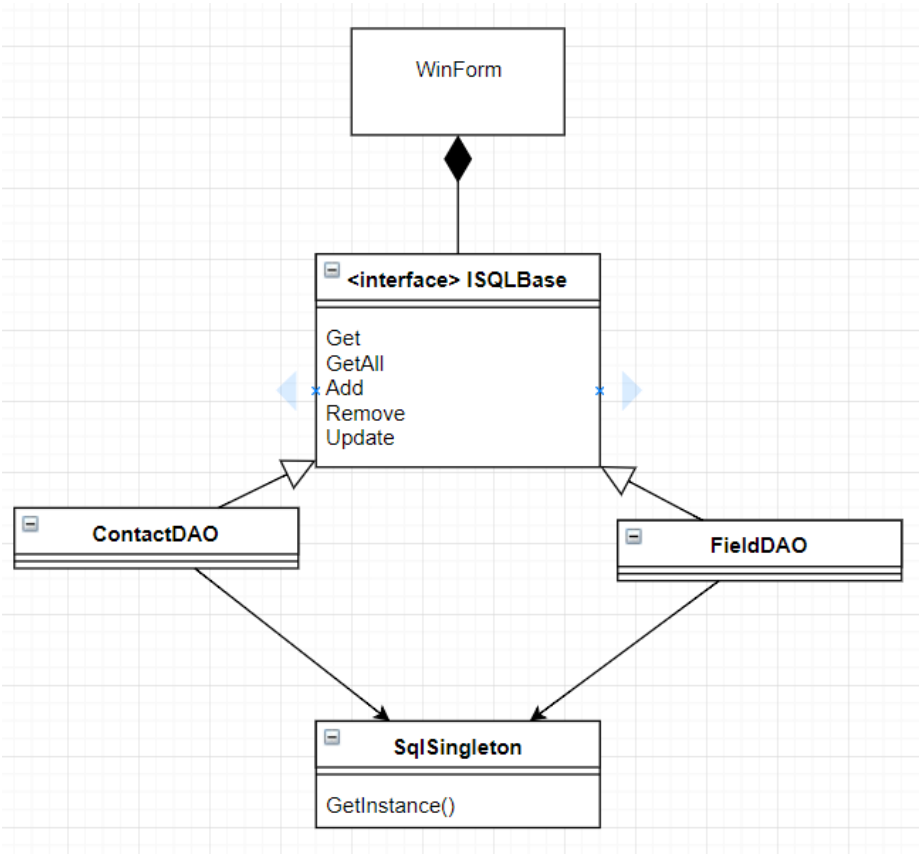
À la création de chaque bouton représentant un contact, une fonction lui est attribuée si on click dessus. La fonction en question va afficher le panneau latéral avec toutes les informations du contact concerné. Le pattern Observer est mis en place pour que si le nom du contact soit changé, il ne soit pas nécessaire de recharger toute la page, puisque la référence du bouton de ce contact a été passé en paramètre et il suffit donc de changer sa propriété « Text »

Data Access Layout

L'accès aux données se fait via le projet « DAL ». Celui-ci fait intervenir deux patterns : le singleton et le composite. La classe static SqlSingleton possède une instance de la classe SqliteManager. Cette classe possède un constructeur qui va exécuter les requêtes SQL contenues dans ses variables internes, qui contienne elles-mêmes la structure des tables de la base SQLite. Ainsi dès le démarrage de l'application, celle-ci va instancier immédiatement la classe SqlManager, qui va elle-même exécuter les scripts de création de table qui vont d'abord vérifier si lesdites tables n'existes pas déjà.



L'accès à la base se fait au travers des objets « Data Access Object », qui implémentent tous les deux l'interface ISQLBase. Cette interface s'assure que tous les DAO possèdent les mêmes fonctionnalités au regard des données qu'ils traitent. Pour accéder aux données, ces deux classes font appel au singleton pour obtenir l'instance de l'accès à la base de données. Cette architecture a été choisis pour permettre de rapidement et facilement rajouter des tables dans la base à l'avenir.

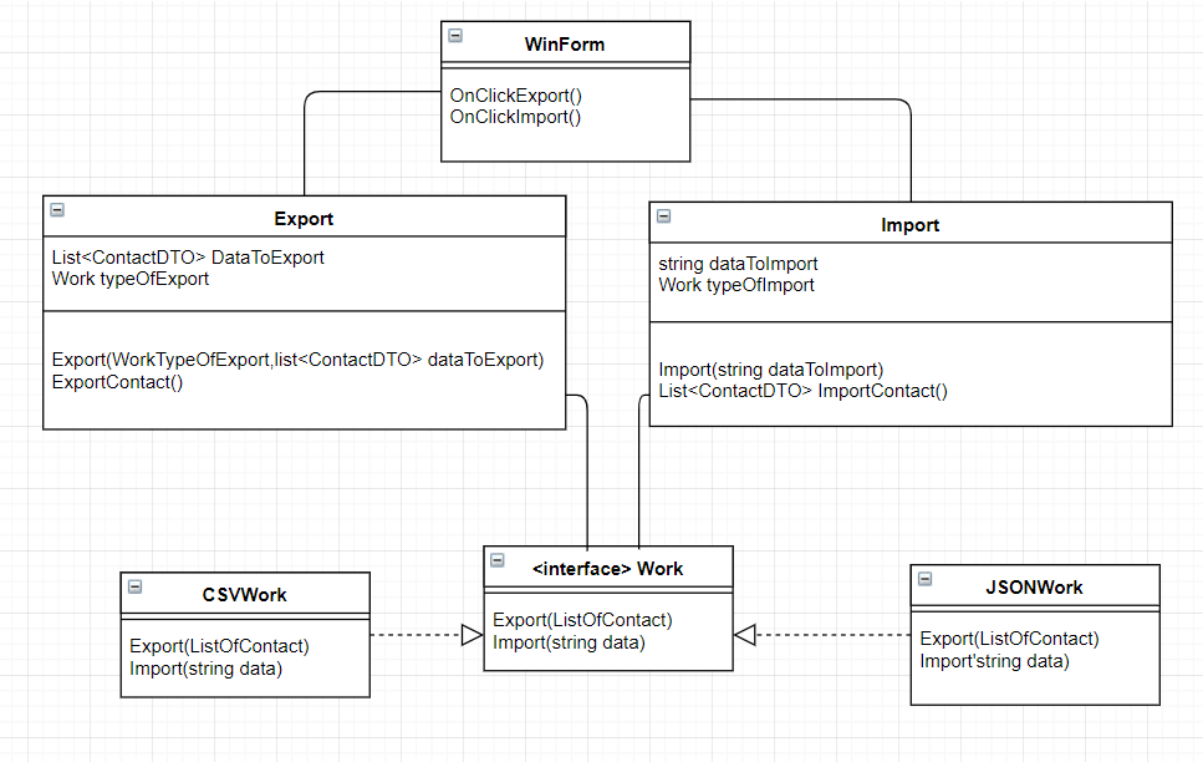


Data Transfert Object

Les DTO représentent la partie Model du MVC. Ce sont eux qui sont utilisés par l'application pour faire circuler les données de la base : leurs propriétés sont les mêmes que les champs des tables qu'ils représentent. Ils sont utilisés par tous les composants de l'application, et leur permettent de communiquer avec un modèle de données communes.

Business Logic Layout

L'appuie sur les boutons exporter ou importer déclenche une action. Le code de cet événement est contenu dans les classes de la BLL. Les différentes classes (Business Object) sont organisées autour du pattern « Template »



Ressources

L'application utilise trois types de ressources pour fonctionner correctement :

- Le fichier de base de données
- Les images
- Les packages NuGet

Le fichier de base de données

En ce qui concerne la base de données, le fichier est créé s'il n'existe pas, sa corruption ou sa suppression n'entraine donc pas une mal fonction de l'application, mais toutes les données sauvegardées seront perdues. Il est possible de les récupérer via un import si un exporta été fait au préalable sans pour autant supprimer les nouveaux contacts qui auront été rajouté entre-temps. Si le fichier est déplacé, il sera considéré comme perdu et un nouveau sera recréé. Le fichier pèse 16Ko par défaut, et il faut excéder les 356 contacts pour atteindre 32Ko

Les images

Les icones que Kontact utilise sur ces boutons et sur sa fenêtre sont des images au format .jpg insérées dans le projet et compilé avec lui, elles sont donc embarquées dans l'exécutable. Les images sont insérées via l'interface graphique de Visual Studio, et placé dans le dossier « ressource » du projet WinForm. Elles sont par la suite récupérées via la méthode static de la classe interne (accessible uniquement aux autres classes de l'assembly/projet) « Ressource » du même nom que la ressource :

```
boutonRemove.BackgroundImage=projetDesignPatern4AL1G16.Properties.Resources.Remove_icon;
```

Les packages NuGet

L'application utilise des librairies pour l'aider dans le traitement de deux fonctionnalités : l'accès à la base SQLite et la transformation de chaine de caractère au format JSON vers un objet C#. Dans Visual Studio le gestionnaire de dépendances par défaut est NuGet : il est lié à un magasin en ligne de toutes les librairies développées par la communauté et déposées sur la boutique Microsoft. Il est ainsi très simple de télécharger et d'intégrer au projet différent composant qui peut facilement être mis à jour au travers de Visual. Dans le cas de Kontact, deux Packages sont utilisés : NewtonSoft.Json version 11.0.2 pour le parsing de JSON à Objet, et Data. SQLite version 1.0.108 pour l'accès à la base SQLite3.