MANZANO Lilian *Groupe T*

PERE Brandon

**Rapport final - CDAA**

SUJET

Ce rapport porte sur une application créé en C++ en utilisant QT. Cette application est un gestionnaire de contacts (nom, prénom, téléphone, …), avec des fonctionnalités de recherches et d’ajout de notes.

**TABLE DES MATIÈRES**

[Fonctionnalités 3](#_Toc122026334)

[Fenêtre principale : 3](#_Toc122026335)

[Interface : 3](#_Toc122026336)

[Recherches : 3](#_Toc122026337)

[Barre d’outils : 4](#_Toc122026338)

[Fenêtre de détails de contact : 4](#_Toc122026339)

[Infos du contact : 4](#_Toc122026340)

[Interactions du contact : 4](#_Toc122026341)

[Fenêtre d’édition des tags : 5](#_Toc122026342)

[Fenêtre d’édition de contact : 5](#_Toc122026343)

[Fenêtre d’historique des modifications : 6](#_Toc122026344)

[Fenêtre de choix des dates : 6](#_Toc122026345)

[Indications à l’utilisateur : 6](#_Toc122026346)

[Fichiers et Dossiers 7](#_Toc122026347)

[Architecture de la base de données 8](#_Toc122026348)

[Diagramme de la base de données : 8](#_Toc122026349)

[Description de la base de données : 8](#_Toc122026350)

[Détails des classes 9](#_Toc122026351)

[Classes du premier jalon : 9](#_Toc122026352)

[Classes gérant la base de données : 10](#_Toc122026353)

[Classes gérant l’interface : 11](#_Toc122026354)

[Signaux et slots 12](#_Toc122026355)

[Fenêtre de sélection de date : 12](#_Toc122026356)

[Fenêtre de choix oui/non : 12](#_Toc122026357)

[Fenêtre principale : 12](#_Toc122026358)

[Widget Principal : 13](#_Toc122026359)

[Fenêtre d’historique : 14](#_Toc122026360)

[Fenêtre de détails du contact : 14](#_Toc122026361)

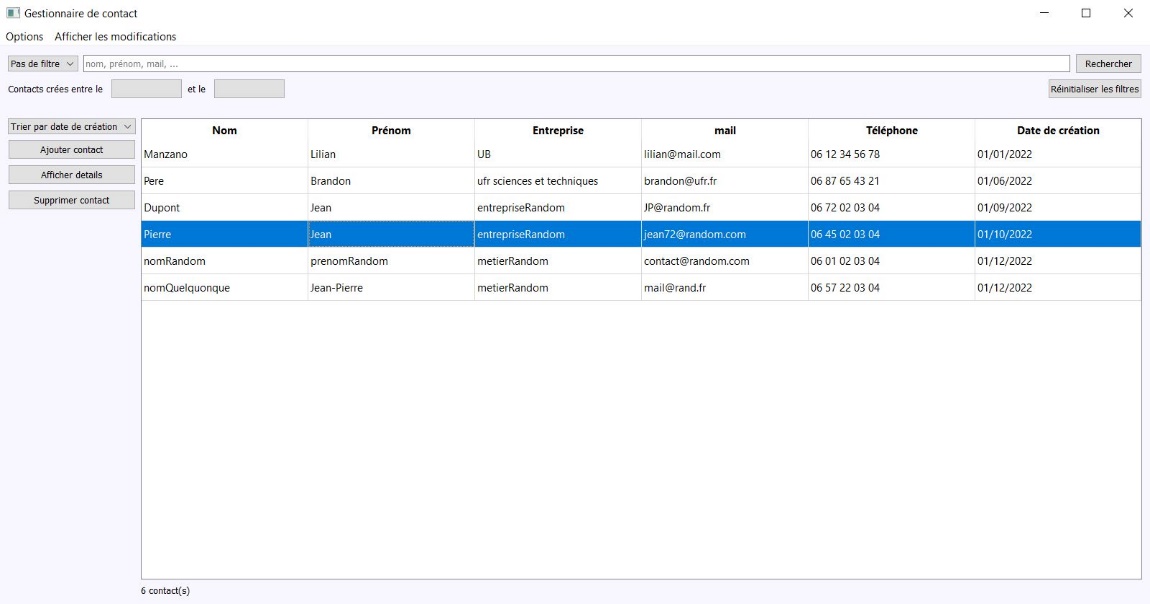
[Fenêtre d’édition de contact : 14](#_Toc122026362)

[Fenêtre d’édition de tags : 15](#_Toc122026363)

# Fonctionnalités

## Fenêtre principale :

### Interface :



La fenêtre principale affiche la liste de contacts et leurs caractéristiques dans un tableau, celui-ci permet de sélectionner une ligne correspondant à un contact.

On trouve en haut une barre de recherche et différents filtres pour trouver des contacts en fonctions de leurs propriétés (nom, prénom, entreprises, …), de leur date de création, ou encore des tags @todo et @date qu’ils possèdent.

En bas du tableau, on y voit indiqué le nombre de contacts correspondant à la recherche, si aucune recherche n’a été faite c’est donc le nombre total de contacts qui y est indiqué.

Enfin à gauche du tableau on trouve un sélecteur et 3 boutons. Le sélecteur permet de choisir un tri des contacts dans le tableau, soit par date de création, par ordre alphabétique ou alphabétique inverse, ces deux derniers tris prennent en compte le nom du contact. De plus il y a un bouton qui permet de supprimer le contact sélectionné, un bouton qui permet d’ouvrir la fenêtre d’édition afin d’ajouter un nouveau contact, et pour finir le troisième bouton permet d’ouvrir la fenêtre de détails du contact sélectionné.

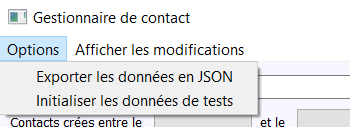
### Recherches :

La barre de recherche et les filtres peuvent tous être cumulés, cela permet ainsi de nombreuses possibilités :

* Il y a un sélecteur de filtres parmi lequel on peut choisir de ne pas en appliquer ou filtrer par nom, prénom, entreprise, mail, ou téléphone.
* Il y a aussi 2 boutons permettant de choisir une date de départ et de limite pour les dates de création des contacts, ainsi si les 2 dates sont sélectionnées on peut trouver des contacts crées sur une période précise.
* Dans la barre de recherche en plus des informations du contact, on peut renseigner des tags @todo ou @date, tout ce qui se trouvera après un tag @todo jusqu’au prochain tag sera considéré comme contenu du tag.

Par exemple si nous voulons tous les contacts crées après le 01/12/2022 ayant une adresse mail «.com» et possédant un tag @todo renseignant une réunion le 01/01/2023, nous pouvons faire :

### Barre d’outils :



Sur cette fenêtre principale on y trouve aussi une barre d’outils. Cette dernière possède un menu options qui permet d’initialiser des données de tests ou d’exporter les données de la base de données en format JSON (l’historique des modifications n’est pas exporté), le fichier «.json» apparait dans le dossier «data» crée par notre application près de l’exécutable. En plus de ce menu, il y a un autre bouton dans la barre d’outils qui permet d’ouvrir la fenêtre d’historique de toutes les modifications.

## Fenêtre de détails de contact :

### Infos du contact :



Cette fenêtre permet de montrer les détails du contact sélectionné dans le tableau de la page principale, on y voit toutes les informations du contact, ainsi que sa photo (l’image par défaut ci-dessus). On peut également y trouver un bouton qui ouvre une fenêtre pour modifier le contact, et un bouton qui ouvre une fenêtre de l’historique des modifications uniquement de ce contact.

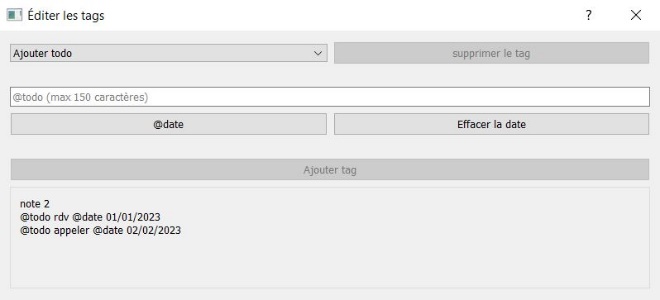
### Interactions du contact :

En seconde partie de la fenêtre on peut éditer les interactions de ce contact, il y a un sélecteur, s’il est sur le premier choix, l’on peut ajouter une interaction. Les autres choix de ce sélecteur sont en fait les interactions que possède le contact, en en sélectionnant une on peut choisir de la modifier, de la supprimer ou d’éditer ses tags (cela ouvrira une nouvelle fenêtre) . De plus on voit en dessous le contenu de l’interaction et ses tags.

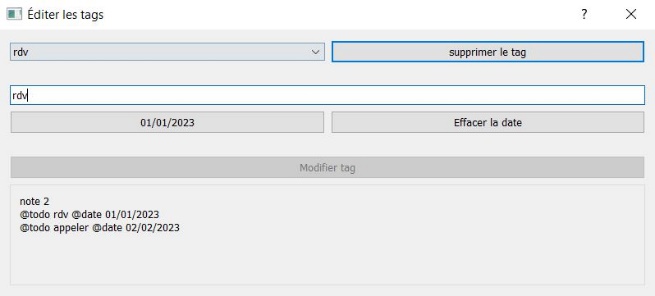
Une image contenant texte

Description générée automatiquement

## Fenêtre d’édition des tags :

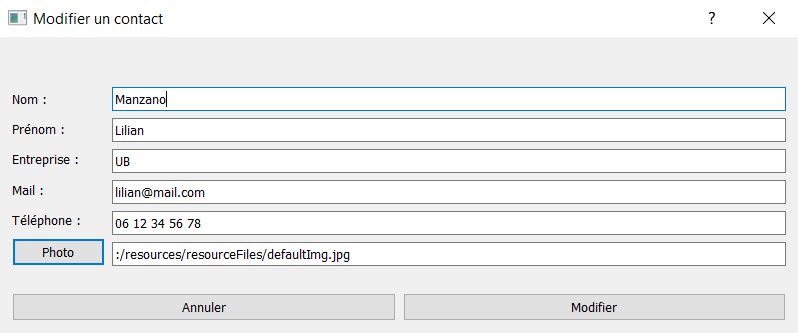


L’édition des tags est similaire à l’édition des interactions, il y a un sélecteur dont le choix par défaut est d’ajouter un nouveau tag, les autres choix sont les tags déjà existants que l’on peut modifier. Il faut au minimum renseigner un contenu qui sera la tag @todo, et il est possible de renseigner une date qui sera donc le @date. On voit en bas l’interaction concernée et tous les tags, à noter que ce soit ici ou dans la fenêtre de détails du contact, les tags sont triés par date.



## Fenêtre d’édition de contact :

Comme nous avons vu précédemment, il y a 2 cas où la fenêtre d’édition de contact s’affiche, lorsque nous voulons crée un nouveau contact depuis la fenêtre principale, ou quand nous voulons en modifier un depuis la fenêtre de détails d’un contact. Pour ces deux utilisations, c’est la même fenêtre, celle-ci est juste prérempli avec les informations du contact que nous voulons le modifier.

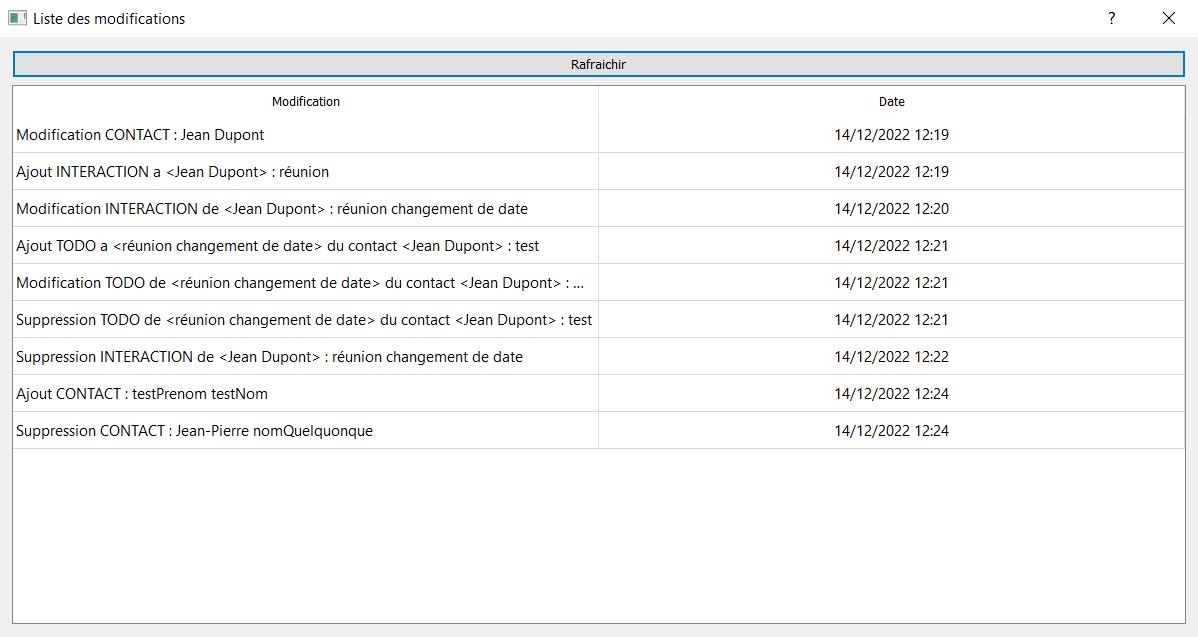


Pour éditer un contact il faut remplir obligatoirement les champs nom, prénom, entreprise, mail et téléphone. La photo est facultative, si aucun lien n’est renseigné ou si celui-ci est erroné, l’image par défaut sera associée au contact, en revanche quand le lien vers une image est bon (image existante et sous le format JPG, JPEG, ou PNG), l’image est copié dans le dossier «img» crée par notre programme près de l’exécutable, ainsi l’utilisateur peut modifier l’emplacement ou supprimer l’image d’origine sans impacter l’application.

## Fenêtre d’historique des modifications :

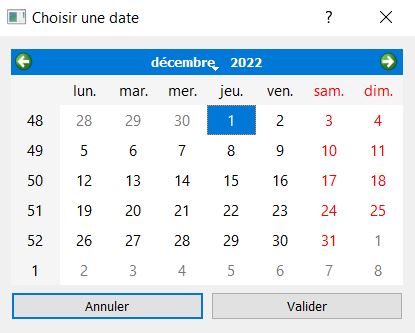
Comme vu précédemment la fenêtre d’historique des modifications s’ouvre dans 2 cas, on peut voir l’historique d’un contact en particulier depuis la fenêtre de détails de celui-ci, c’est-à-dire sa création, ses modifications, mais aussi tout ce qui concerne ses interactions et tags.

Et depuis la page principale on peut choisir de voir l’historique global, c’est-à-dire toutes les informations vues juste avant pour tous les contacts, mais aussi les suppressions des contacts qui n’existent plus.



Si des modifications sont faites pendant que cette page est ouverte, le tableau n’est pas actualisé automatiquement pour les afficher, il y a en haut de la fenêtre un bouton pour rafraichir et les afficher sans avoir besoin de fermer et rouvrir la fenêtre.

## Fenêtre de choix des dates :

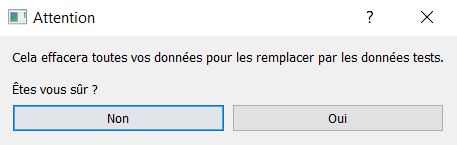
Pour choisir des dates que ce soit pour les filtres de recherches où les tags @date, ce sont des boutons qui ouvre une fenêtre de dialogue avec un calendrier où il est possible de sélectionner une date, cela nous permet notamment de ne pas avoir à vérifier le format de la date contrairement à un champ d’édition

## Indications à l’utilisateur :

Pour faciliter l’utilisation de notre application, tout au long de l’utilisation de celle-ci nous informons l’utilisateur de nombreuses informations grâce à des message rouges pour une erreur, un message vert pour les réussites, et une boîte de dialogue posant une question avec oui ou non comme réponse. Par exemple :







# Fichiers et Dossiers

En plus du fichier «.pro», et des sources «.h» et «.cpp», notre projet possède un dossier «resourceFiles» et un fichier «resource.qrc» renseignant tout ce que contient ce dossier. Ce dossier possède :

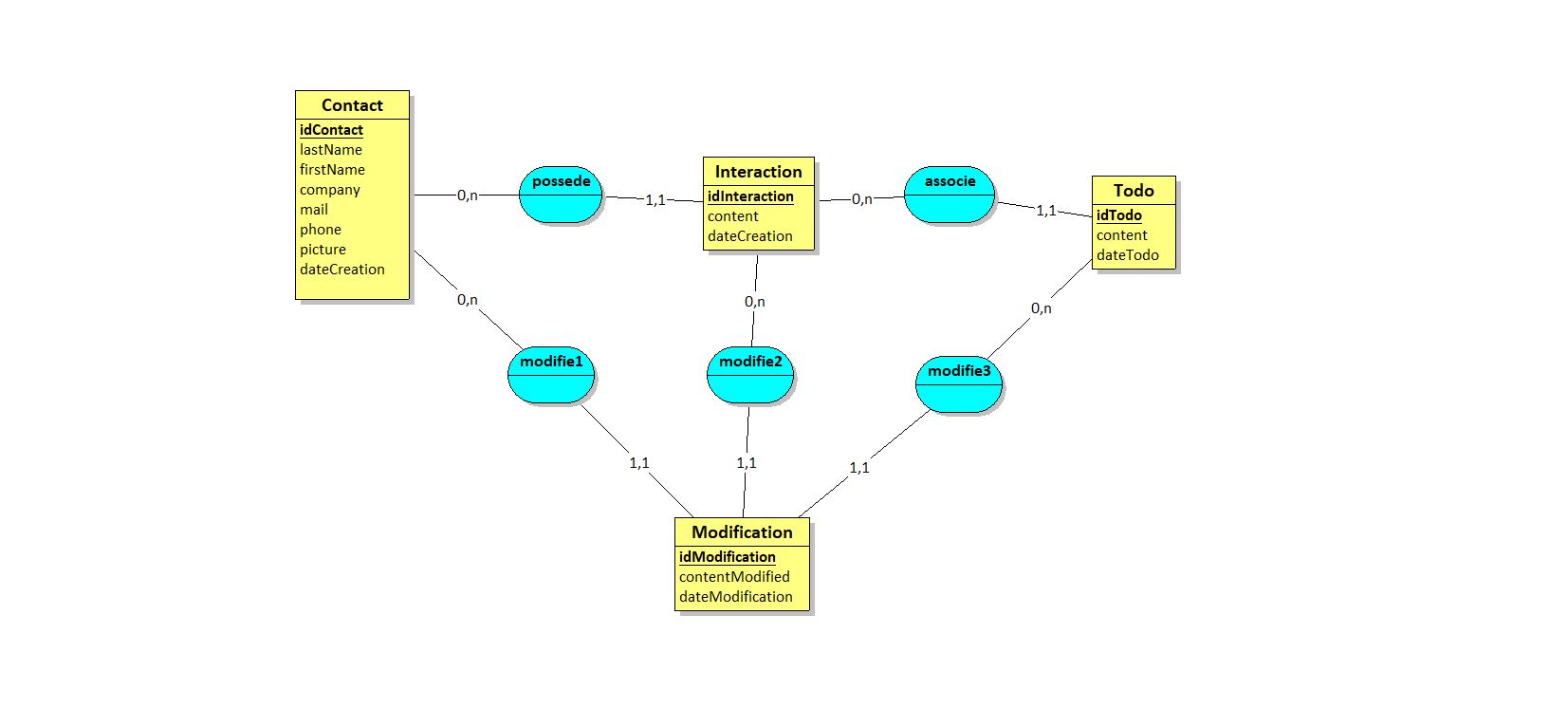
* L’image par défaut d’un contact.
* Un fichier de style «style.qss» que nous n’avons pas encore vraiment rempli, mais que nous avons mis pour éventuellement perfectionner le projet par la suite.
* Le fichier «dataTest.sql» qui contient des requêtes d’insertions pour initialiser les données de tests.
* Les 4 fichiers qui contiennent les instructions pour créer les 4 tables de notre base de données : «initContact.sql», «initInteraction.sql», «initTodo», «initModification».

En plus de cela notre programme va quand il va être lancé créer 2 dossiers près de l’exécutable :

* «data» qui va contenir la base de données SQLite et les éventuelles fichier JSON lors d’un export.
* «img» qui va contenir les images copiée lorsque l’utilisateur choisi sur son ordinateur une photo pour un contact.

# Architecture de la base de données

## Diagramme de la base de données :



## Description de la base de données :

Notre base de données n’a pas évolué depuis le premier jalon, elle possède toujours 4 tables qui sont respectivement :

* **Contact** (idContact, lastName, firstName, company, mail, phone, picture, dateCreation)
* **Interaction** (idInteraction, content, dateCreation, *idContact*)
* **Todo** (idTodo, content, dateTodo, *idInteraction*)
* **Modification** (idModification, contentModified, dateModification, *idContact*, *idInteraction*, *idTodo*)

# Détails des classes

## Classes du premier jalon :

Lors du premier jalon, nous avons décrit différentes classes objets ne dépendant pas de QT, celles-ci sont : Date, Contact, Interaction, Todo, ListContact, ListInteraction, ListTodo.

Ces classes ont peu évolués, c’est pour cela que si vous souhaitez plus d’informations, nous vous invitons à vous référer au rapport précédent.

Les quelques changements réaliser sur celles-ci sont :

* Les méthodes ‘toString()’ de ces classes ont changés pour mieux convenir à nos besoins.
* De nouvelles méthodes dans la classe Date permettant d’en comparer plusieurs objets, notamment utile pour le tri par date de création des contacts.
* Les classes de listes qui étaient toutes similaires possèdent désormais des méthodes propres à elles, comme par exemple les méthodes de tri par ordre alphabétique d’une liste de contact.

## Classes gérant la base de données :

Pour gérer la base de données, nous avons deux classes :

* **DatabaseManagement** qui va réaliser les tâches qui vont porter sur toutes la BDD, comme l’ouvrir, créer ses tables si nécessaire, initialiser les données de tests, en exporter tout le contenu en format JSON, …
* **DatabaseCRUD** qui elle possède des méthodes pour réaliser toutes les requêtes portant uniquement sur certains éléments de la BDD, comme les requêtes d’insertion, de modification, de suppression, ou de sélection ; CRUD signifiant create, read, update, et delete.

Grâce à ces méthodes, toutes les modifications demander par l’utilisateur comme créer un contact, le modifier, ou encore le supprimer seront faites directement et simplement dans la base de données, ainsi si jamais un problème survient, les modifications ne seront pas perdues.

De plus pour obtenir des listes, notamment lors de la recherche de contact, nous utilisons une requêtes SQL, ce qui est plus simple que de créer nos propres fonctions sur nos objets de listes.

Une image contenant texte

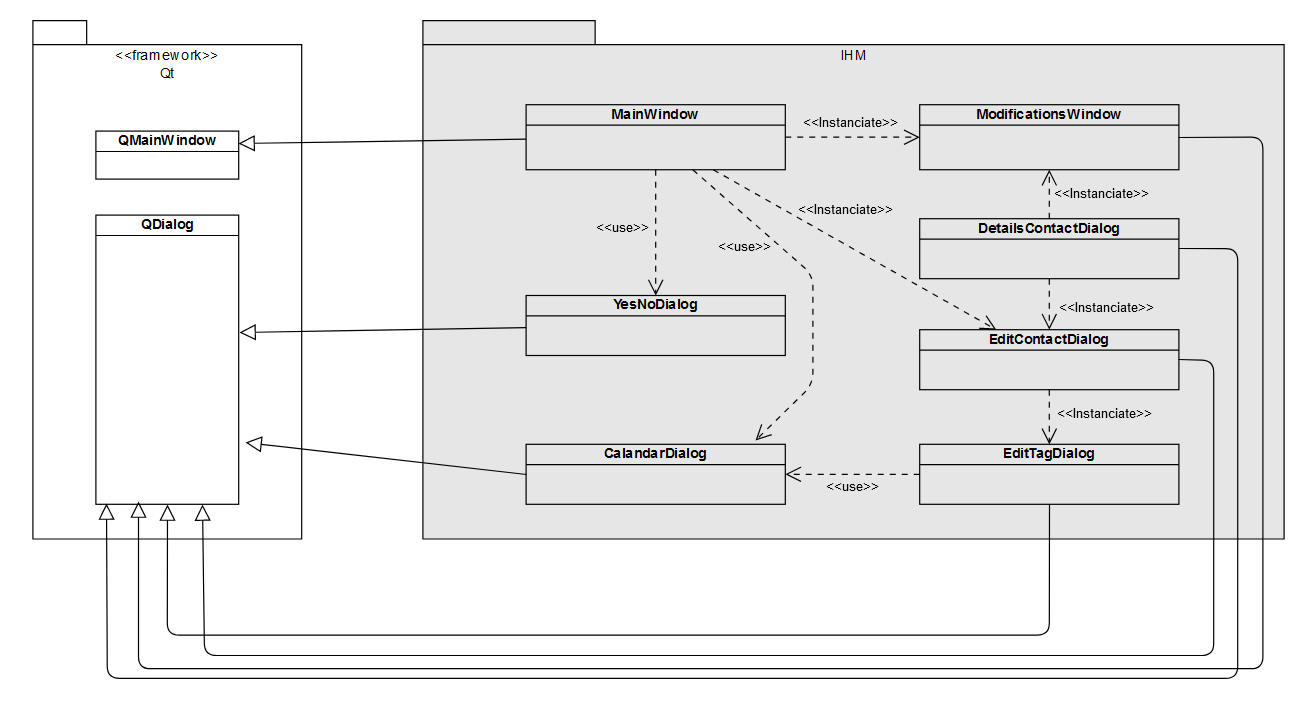
Description générée automatiquement

## Classes gérant l’interface :

Toutes les autres classes servent à gérer l’interface :

* **MainWindow** hérite de **QMainWindow**, et décrit la fenêtre principale avec sa barre d’outils, son widget central est un objet de **MainWidget**.
* **MainWidget** hérite de **QWidget** décrit l’interface de la fenêtre principale.
* **MessageLabel** hérite de **QLabel**, et est le message qui est écrit dans le cas de réussite ou d’échec, cela nous permet de l’utiliser comme un **QLabel**, mais en ayant rajouter une énumération et méthode pour définir son style (rouge, vert, ou rien), ce message est utilisé comme n’importe quel label par nos autres classes.
* **YesNoDialog** hérite de **QDialog**, et défini les fenêtres de question simple dans notre projet, son constructeur possède le titre et la question que nous voulons pour celle-ci, cela permet d’en créer facilement peu importe le contexte.
* **EditContactDialog** hérite de **QDialog**, et est la fenêtre d’édition de contact, celle-ci possède 2 constructeurs dont l’un où un contact doit être renseigner, suivant le constructeur utilisé la fenêtre s’adapte soit pour créer un contact ou alors pour le modifier (changement titre de la fenêtre, changement texte du bouton, remplissage avec les données du contact, …).
* **ModificationsWindow** hérite de **QDialog**, et décrit la fenêtre d’historique des modifications, comme la classe précédente celle-ci possède 2 constructeurs dont l’un avec un contact, ceci afin de s’adapter au contexte dans lequel elle est appelée et afficher éventuellement uniquement l’historique d’un contact.
* **DetailsContactDialog** hérite de **QDialog**, et décrit la fenêtre de détail du contact et d’édition d’interaction.
* **EditTagDialog** hérite de **QDialog**, et défini la fenêtre d’édition des tags.

Toutes ces classes possèdent un attribut **DatabaseCRUD** quand nécessaire, ainsi elles n’ont pas à réaliser elles-mêmes les requêtes à la base de données. Leurs seules tâches sont de décrire les différents éléments d’interfaces comme boutons, champs d’édition, … ; modifier leur propriétés et placement dans certains cas, par exemple rendre le bouton «Supprimer le contact» utilisable uniquement si un contact est sélectionné ; ouvrir d’autres fenêtres de dialogue qui sont dans ses propres attribut ; ou appeler des méthodes de **DatabaseCRUD** sous certaines conditions, par exemple appeler une méthode pour créer le contact uniquement si tous les champs d’éditions sont remplis et si le bouton «Ajouter» est cliquer.



# Signaux et slots

Pour communiquer et réaliser des actions en fonction de leurs inputs, nos classes d’interfaces possèdent des signaux et des slots. Les connexions entre ceux-ci sont directement réalisées au sein de nos fenêtres, et non pas dans le ‘main’, cela car nous créons de nouvelles fenêtres de dialogues à chaque fois qu’il en est demandé une, ainsi nous pouvons faire les connexions avec celle-ci dès qu’elle est créée.

Ici nous allons voir ces signaux et slots à travers les connexions faites dans chaque fenêtre.

## Fenêtre de sélection de date :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton « Annuler » | clicked | this | cancelCloseDialog | Le signal emitClose(QDate \*) est émis dans le slot |
| Bouton « Valider » | clicked | this | validateCloseDialog | Le signal emitClose(nullptr) est émis dans le slot |

## Fenêtre de choix oui/non :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton « Oui » | clicked | this | noCloseDialog | Le signal emitClose(true) est émis dans le slot |
| Bouton « Non » | clicked | this | yesCloseDialog | Le signal emitClose(false) est émis dans le slot |

## Fenêtre principale :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton d’option  « Exporter les données en JSON » | triggered | this | createJSON |  |
| Bouton d’option  « Initialiser les données test » | triggered | this | askInitDataTest |  |
| this | emitUpdateContact | this->mainWidget | searchContacts | Signal envoyé pour actualiser le tableau de contact quand les données tests ont été initialiser dans la BDD |
| Bouton d’option « Afficher les modifications » | triggered | this | openModificationsWindow |  |
| Fenêtre pour valider l’initialisation des données tests | emitClose(bool) | this | closeYesNoDialog(bool) | Connexion créée uniquement une fois la fenêtre crée, un paramètre booléen est passé pour connaître le choix fait par l’utilisateur |

## Widget Principal :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton pour choisir une première date de filtre | clicked | this | openFirstCalendarDialog |  |
| Bouton pour choisir une seconde date de filtre | clicked | this | openSecondCalendarDialog |  |
| Bouton « Rechercher » | clicked | this | searchContacts |  |
| Bouton « Réinitialiser les filtres » | clicked | this | resetFilters |  |
| ComboBox pour choisir un tri | currentIndexChanged | this | updateTable |  |
| Bouton « Ajouter contact » | clicked | this | openCreateContactDialog |  |
| Bouton « Afficher détails » | clicked | this | openDetailsContactDialog |  |
| Bouton « Supprimer contact » | clicked | this | deleteContact |  |
| Tableau des contacts | cellClicked | this | enableDeleteDetailsButton |  |
| Fenêtre de dialogue pour sélectionner la première date de filtre | emitClose(QDate \*) | this | closeSecondCalendarDialog  (QDate \*) | Connexion créée uniquement une fois la fenêtre crée, un paramètre QDate est passé pour récupérer la date sélectionnée |
| Fenêtre de dialogue pour sélectionner la seconde date de filtre | emitClose(QDate \*) | this | closeSecondCalendarDialog  (QDate \*) | Connexion créée uniquement une fois la fenêtre crée, un paramètre QDate est passé pour récupérer la date sélectionnée |
| Fenêtre de création de contact | emitClose(Contact\*, bool) | this | editContact(Contact\*, bool) | Connexion créée uniquement une fois la fenêtre crée, des paramètres Contact et booléen sont passés pour l’ajouter dans le tableau et afficher un message d’erreur ou réussite |
| Fenêtre de détails du contact | emitModifyContact(Contact\*, bool) | this | editContact(Contact\*, bool) | Connexion créée uniquement une fois la fenêtre crée, des paramètres Contact et booléen sont passés pour le mettre à jour dans le tableau et afficher un message d’erreur ou réussite |

## Fenêtre d’historique :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton « Rafraichir » | clicked | this | refreshTable | Seul une de ces 2 connexion est faite en fonction du constructeur utilisé |
| Bouton « Rafraichir » | clicked | this | refreshTableByContact |

## Fenêtre de détails du contact :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton « Modifier le contact » | clicked | this | openModifyContactDialog |  |
| Bouton « Afficher les modifications » | clicked | this | askInitDataTest |  |
| ComboBox pour choisir une interaction | currentIndexChanged | this | updateInputInteraction |  |
| Bouton « Supprimer l’interaction » | clicked | this | removeInteraction |  |
| Bouton « Éditer les tags » | clicked | this | openEditTagDialog |  |
| Champ d’édition pour l’interaction | textChanged | this | updateEditInteractionButton |  |
| Bouton « Ajouter/Modifier l’interaction » | clicked | this | addModifyInteraction |  |
| Fenêtre de modification de contact | emitClose(Contact\*, bool) | this | editContact(Contact\*, bool) | Connexion créée uniquement une fois la fenêtre crée, des paramètres Contact et booléen sont passés pour le mettre à jour dans l’affichage et emettre un nouveau signal emitModifyContact(Contact\*, bool) à la fenêtre principal pour également la mettre à jour |
| Fenêtre d’édition de tags | emitUpdateTag | this | updateTag |  |

## Fenêtre d’édition de contact :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| Bouton « Photo » | clicked | this | openFileDialog |  |
| Champ d’édition pour le lien de la photo | textChanged | this | checkPathPicture |  |
| Bouton « Annuler » | clicked | this | close |  |
| Bouton « Ajouter/Modifier » | clicked | this | editContact | Le signal emitClose(Contact, bool) est émis dans le slot |

## Fenêtre d’édition de tags :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Émetteur | Signal | Destinataire | Slot | Détails |
| ComboBox pour choisir un tag | clicked | this | updateInputTodo |  |
| Bouton « Supprimer le tag » | clicked | this | deleteTodo | Le signal updateTodo est émis dans le slot pour mettre à jour l’affichage de la fenêtre de détails |
| Champ d’édition du tag | textChanged | this | updateEditTodoButton |  |
| Bouton « @date » | clicked | this | openCalendarDialog |  |
| Bouton « Effacer la date » | clicked | this | removeDate |  |
| Bouton « Ajouter/modifier tag » | clicked | this | addModifyTodo | Le signal updateTodo est émis dans le slot pour mettre à jour l’affichage de la fenêtre de détails |
| Fenêtre pour choisir la date | emitClose(QDate\*) | this | closeCalendarDialog(QDate\*) | Connexion créée uniquement une fois la fenêtre crée, un paramètre QDate est passé pour récupérer la date sélectionnée |