Compte rendu final - Projet CDAA

Sener Betul
Dusoleil Yoan

Décembre 2022

Contents

1 Introduction et utilisation de l'application

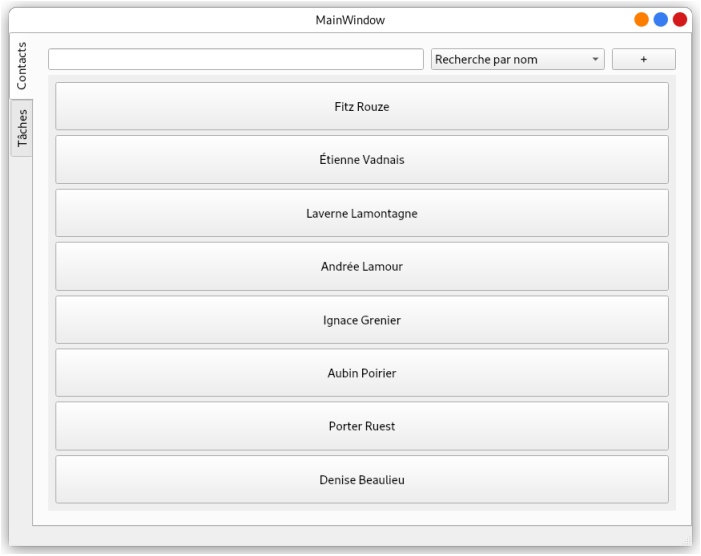
1.1 Compilation et exécution

Tout d'abord, un simple `qmake Projet-CDAA.pro` suivi d'un `make` dans le terminal suffit pour compiler l'application. Une fois l'application compilée, elle génère un exécutable nommé `projetCDAA`.
Si les fichiers sources sont compilés depuis un IDE, l'exécutable risque d'être placé dans un dossier autre que la racine du projet. Les chemins dans le code étant relatif au répertoire d'éxecution du programme, l'application n'arrivera alors pas à trouver le fichier .sqlite situé dans [emplacement du dossier du projet]/data/data.sqlite et une fenêtre d'alerte apparaîtra alors, demandant à l'utilisateur de sélectionner l'emplacement du fichier. Les photos peuvent aussi avoir un problème de chargement pour les mêmes raisons.

1.2 Importer/Exporter un fichier JSON

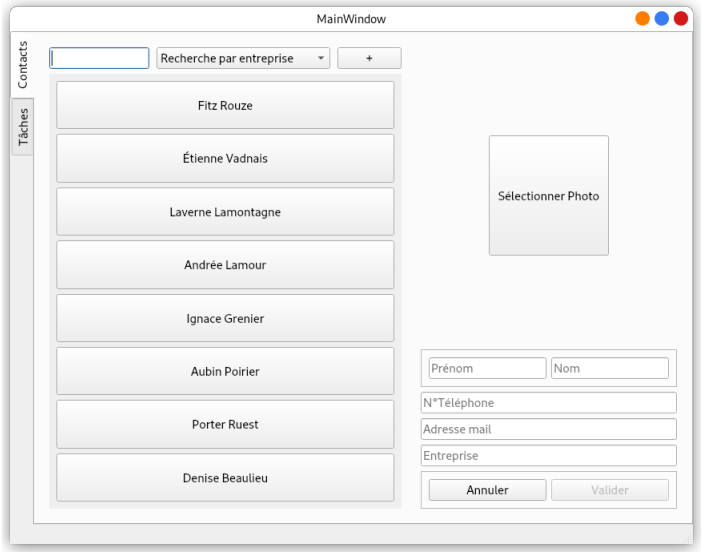
Le menu "Fichier" disponible sur la barre de menu en haut à gauche de l'interface permet d'importer une liste de contact depuis un fichier JSON. On peut aussi exporter la liste de contact actuelle dans un fichier JSON.

1.3 Accueil de l'application



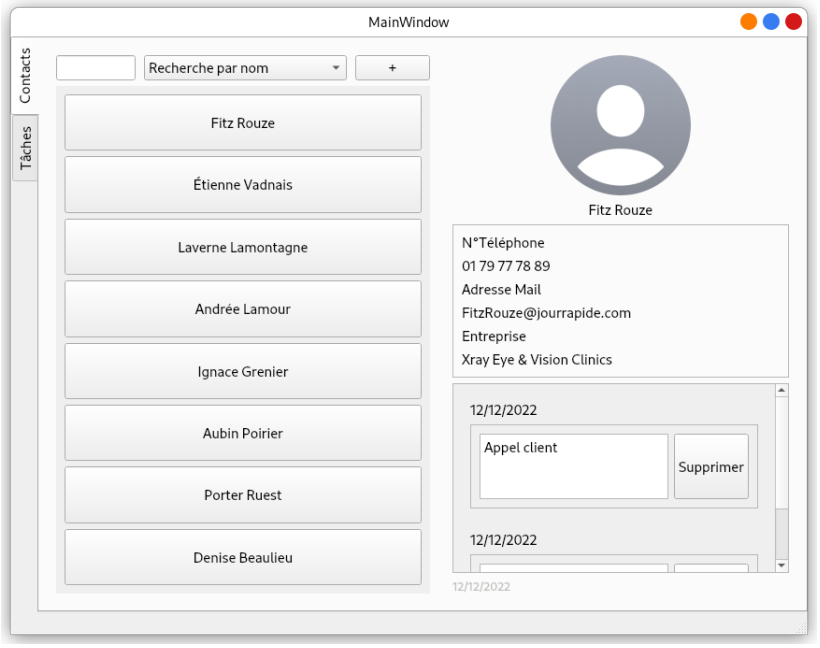
Lors du lancement de l'application l'utilisateur est accueilli par la fenêtre suivante. On peut y distinguer de multiples fonctionnalitées. En appuyant sur le bouton `+` en haut à droite de l'interface, on peut ajouter un contact à la liste de contacts.

1.4 Ajouter un contact



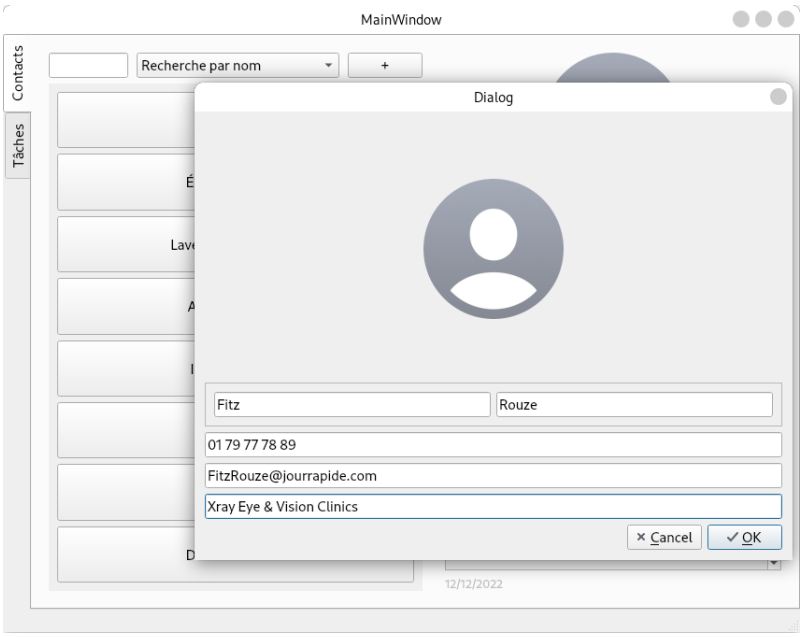
Un "Widget" devient alors visible à droite de la liste. Dans ce Widget on a alors un formulaire dont les champs servent à renseigner les informations du contact que l'on souhaite créer. On a également un bouton "Sélectionner Photo" qui, lorsque l'on clique dessus, nous permet de sélection une image à associer au contact. L'ajout n'est possible que si tous les champs sont remplis. Une image par défaut lui sera associé si aucune image est sélectionnée.

1.5 Voir les informations d'un contact



Une fois un contact ajouté, on peut alors cliquer sur le bouton au nom de ce contact dans la liste et un Widget devient visible. Il contient la photo, le nom, le prénom, l'adresse mail, le numéro de téléphone et l'entreprise du contact. En dessous de ces informations, on peut voir la liste des intéractions que l'on a eu avec le contact.

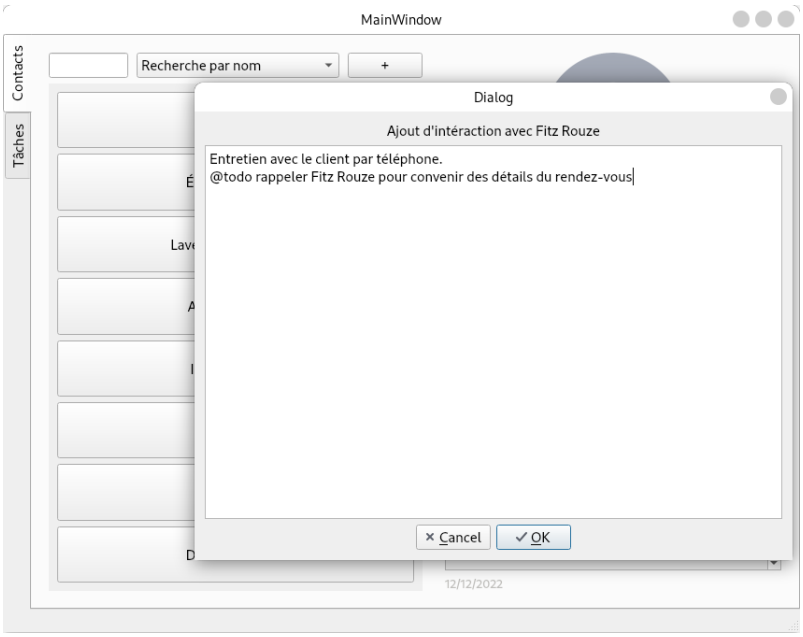
1.6 Modifier un contact



Pour modifier un contact, il faut faire un clic droit sur le bouton lui correspondant dans la liste. Un menu contextuel apparaît alors avec pour options :
- "Modifier" pour modifier le contact.
- "Supprimer" pour supprimer le contact.
Si l'utilisateur souhaite modifier le contact, une fenêtre de dialogue ci-dessus apparaît alors avec les informations du contact dans des champs de texte éditables. Une fois les modifications terminées, l'utilisateur peut appuyer sur "Ok" pour confirmer ou "Cancel" pour annuler.

1.7 Ajouter une intéraction à un contact

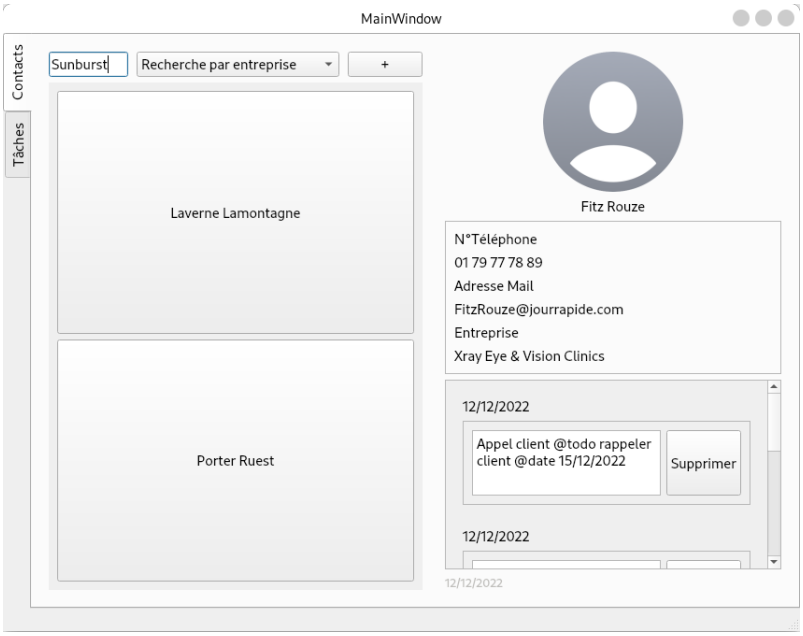
Une des fonctionnalitées spécifiques à cette application est l'ajout "d'intéractions" avec les contacts. L'utilisateur peut en effet ajouter des éléments relatifs à une discussion ou un échange avec un contact. Dans ces intéractions, le tag `@todo` permet de créer une tache. Le texte suivant le tag `@todo` sera alors le contenu de la tâche jusqu'à un retour à la ligne ou un tag `@date` suivi d'une date au format JJ/MM/AAAA si l'utilisateur souhaite spécifier une date à la tâche. Cette fonctionnalité n'était pas disponible lors de la présentation mais a depuis été ajoutée.



L'ajout d'intéraction se fait en faisant un clic droit dans la zone sous les informations du contact. Un menu contextuel avec l'option "Ajouter Interaction" apparaît alors. Lorsque l'option est sélectionnée, la fenêtre de

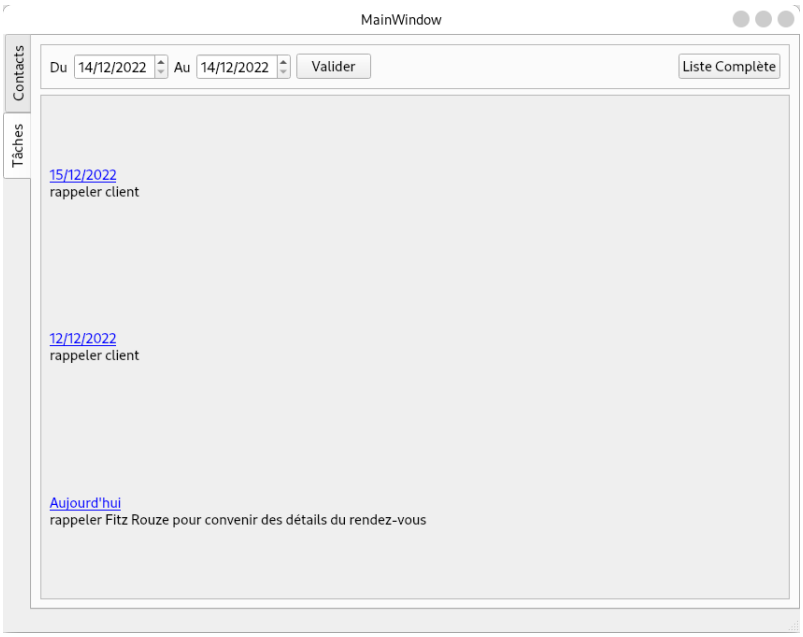
dialogue ci-dessus apparaît alors permettant de saisir librement le contenu de l'intéraction. L'utilisateur peut alors appuyer sur "Ok" pour confirmer l'intéraction ou "Cancel" pour annuler.

1.8 Rechercher dans la liste des contacts



En haut de la liste de contact, l'interface dispose d'un champ de texte permettant de rechercher un contact. Le choix du mode de recherche se fait via le menu déroulant à droite de ce champ de texte. Les différents mode de recherche sont :
- Recherche par nom qui permet de rechercher un contact via son nom ou son prénom.
- Recherche par entreprise affichera les contacts ayant pour entreprise l'entreprise saisie.
- Recherche par date de création : Le champ de texte est remplacé par 2 champs permettant de sélectionner une date. Les contacts affichés dans la liste seront uniquement ceux créés entre ces 2 dates.

1.9 Voir les tâches à faire

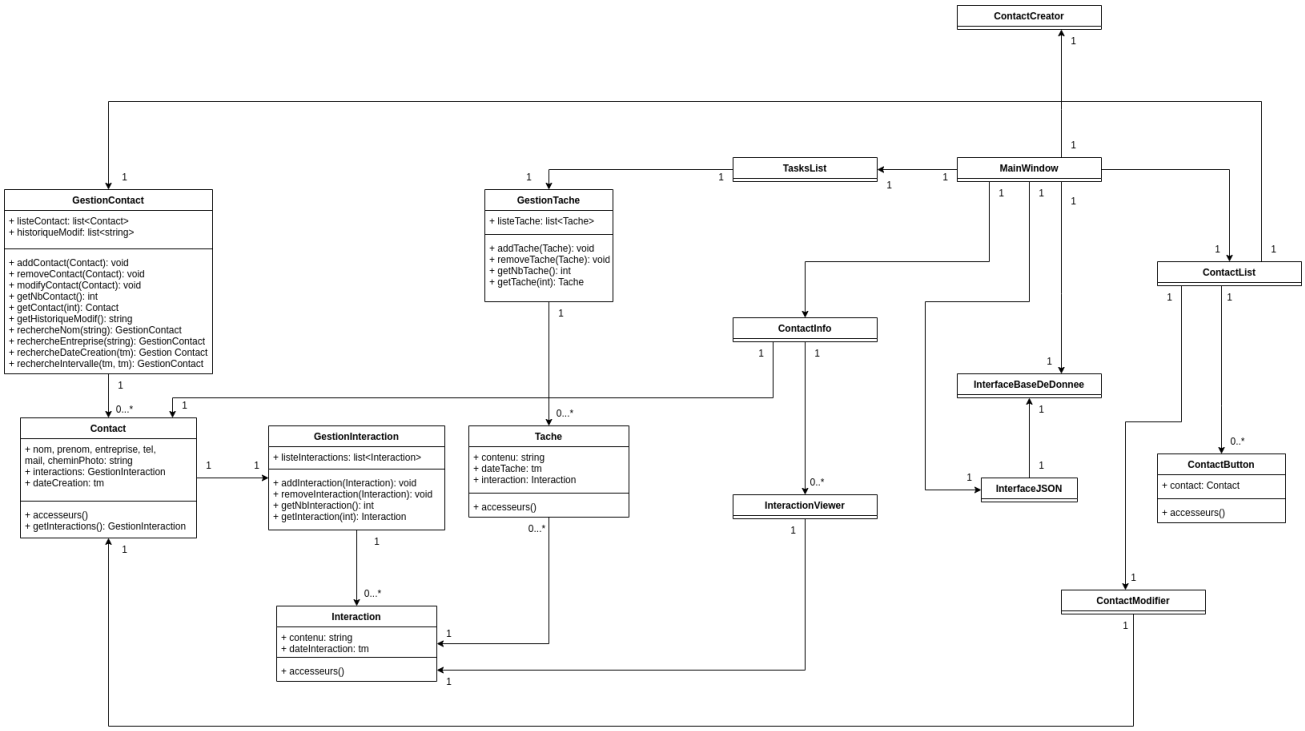


Si des intéractions avec des contacts contiennent des tags `@todo` alors des tâches sont créées. Les tâches créées sont alors affichées dans une liste visible dans l'onglet "Tâches" sur la gauche de l'interface. Cette liste

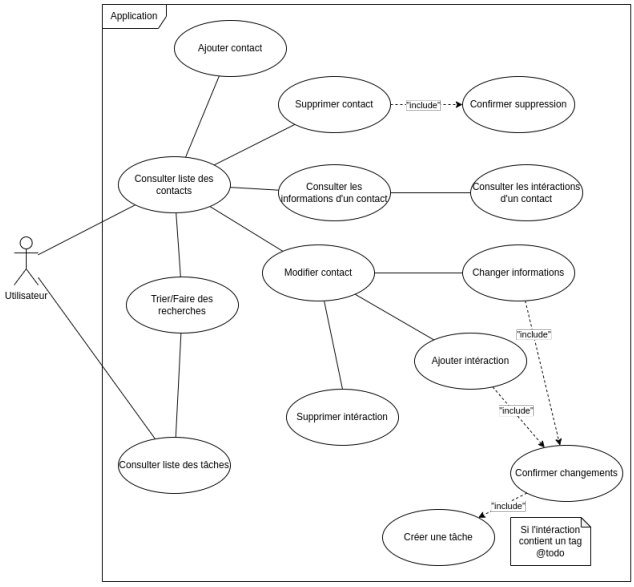
peut-être réduite à un intervalle de date sélectionnable en haut à gauche de l'onglet. On peut toujours afficher la liste complète avec le bouton "Liste complète" en haut à droite de l'onglet.

2 Diagrammes de conception

2.1 Diagramme de classes



2.2 Diagramme de cas d'utilisation



2.3 Description de la base de données

La base de donnée s'établit sur les trois relations suivantes :

contacts(**id**, nom, prenom, mail, entreprise, cheminPhoto, tel, dateCreation)
interactions(**id, idContact**, contenu, dateInteraction)
taches(**idTache**, idInteraction, idContact, contenu, dateTache)

Les intéractions étant liées à un contact, on a besoin de stocker l'id du contact auquel elle est liée. La clé du contact fait aussi partie, en tant que clé étrangère de l'identifiant de l'intéraction. Une tache étant liée à une intéraction, le schéma se répète : la clé de l'intéraction est une clé étrangère dans la tache.

3 Classes

3.1 Structures de données C++

3.1.1 Contact

La classe Contact représente un contact et ses différents attributs (nom, prénom, n°téléphone, adresse mail, entreprise) et peut contenir un chemin (relatif ou absolu) vers une photo du contact. La création du contact est horodatée et stockée au sein de la classe. Les différentes intéractions avec un contact sont enregistrées dans un attribut GestionInteraction. La classe est programmée de façon monolithique afin de simplifier sa destruction pour simplifier les ajouts/suppressions de contact sans risquer de fuite mémoire. Cette architecture peut-être changée plus tard dans le développement si nécessaire.

3.1.2 GestionContact

Cette classe est la classe regroupant l'intégralité des contacts dans une liste. Elle permet de supprimer, modifier et ajouter des contacts à la liste qu'elle contient. Chaque ajout, suppression ou modification effectuée est horodatée et enregistrée dans une liste de logs. Elle permet également de faire des recherches par nom, nom d'entreprise ou par date de création.

3.1.3 Interaction

Cette classe sert à enregistrer les différentes intéractions que l'on peut avoir avec un contact. Elle contient le contenu de l'intéraction ainsi que la date à laquelle elle a eu lieu. Le contenu des intéractions peut être composé de "tags" tels que "@todo" ou "@date" servant à créer des taches et à spécifier les dates de ces taches.

3.1.4 GestionInteraction

Cette classe est la classe regroupant l'intégralité des intéractions avec un contact dans une liste. Elle permet de supprimer et d'ajouter des intéractions à la liste qu'elle contient.

3.1.5 Tache

La classe Tache représente une tache à faire à une date donnée et est créée grâce aux tags figurant dans les intéractions. L'identification des tags et la création d'une tâche correspondant à ce que l'intéraction décrit se fera plus tard dans le développement à l'aide d'un "parser".

3.1.6 GestionTache

Cette classe est la classe regroupant l'intégralité des taches à faire dans une liste. Elle permet de supprimer et d'ajouter des taches à la liste qu'elle contient. Elle est indépendant des autres structures de données afin de pouvoir naviguer facilement dans l'ensemble des tâches à faire indépendamment de la liste des contacts.

3.2 Classes Qt5

3.2.1 MainWindow

Cette classe est en quelque sorte "la clé de voute" du programme. C'est au travers de cette classe que les différents signaux sont reliés aux slots auxquels ils ne peuvent être reliés directement. C'est aussi au travers de cette classe que la majorité de l'interface graphique s'articule.

3.2.2 InterfaceBaseDeDonnee

La classe InterfaceBaseDeDonnee permet d'effectuer des requêtes sur la base de données depuis n'importe quelle autre classe. Les différentes requêtes executables vont de:
- Lecture/Ecriture sur la liste des contacts
- Lecture/Ecriture sur la liste des tâches
- Lecture/Ecriture sur la liste des intéractions

Le fait de passer par une classe dédiée permet de détacher les requêtes sur la base de données du reste du programme.

3.2.3 InterfaceJSON

La classe InterfaceJSON propose des fonctions permettant d'exporter la liste des contacts dans un fichier JSON ou d'importer des contacts dans la liste à partir d'un fichier JSON. Elle est particulièrement utile pour charger rapidement des contacts d'une instance de l'application à une autre. Elle n'était pas présente lors de la démonstration mais a été ajoutée entre temps.

3.2.4 ContactInfo

La classe ContactInfo permet d'afficher sur l'interface les informations du contact qu'elle a en attribut. Elle permet aussi d'ajouter des intéractions au contact depuis une fenêtre de dialogue. Lors de la confirmation de l'ajout de l'intéraction, le programme se charge d'extraire les tags `@todo` et `@date` de l'intéraction afin d'en créer des tâches et de les ajouter à la liste des tâches.

Signaux :
`void interactionDeleted(Interaction)` : Signal émit lors de la suppression d'une intéraction associé au Contact.
`void interactionAdded(Contact)` : Signal émit lors de l'ajout d'une intéraction au contact en paramètre du signal.
`void refreshContactInfo()` : Signal émit lors du rafraîchissement des informations du contact affiché.

3.2.5 InteractionViewer

Cette classe permet de visualiser sur l'interface graphique le contenu d'une des intéractions d'un contact.

Signal :
`void interactionDeleted(Interaction)` : Signal émit lors de la suppression de l'intéraction associé.

3.2.6 ContactList

Cette classe affiche l'entièreté des contacts contenu dans l'objet GestionContact qu'elle a en attribut dans l'interface. Elle permet aussi, lors d'une entrée de la part de l'utilisateur dans l'interface graphique, d'effectuer des recherches de contact et d'afficher un objet GestionContact actualisé avec uniquement le résultat de la recherche.

Signaux :
`void refreshContactList(GestionContact *)` : Signal émit pour rafraîchir l'affichage de la liste des tâches.
`void showContactInfo(Contact)` : Signal émit lorsqu'un qu'un des contacts de la liste est à afficher.
`void createButtonClicked()` : Signal émit lorsque le bouton d'ajout de contact est déclenché.
`void contactModified(Contact, Contact)` : Signal émit lorsqu'un contact est modifié. Le signal contient l'ancienne version et la nouvelle version du contact.
`void contactDeleted(Contact)` : Signal émit lorsqu'un contact est supprimé.

3.2.7 ContactButton

Classe héritant de la classe QPushButton. Elle dispose d'un contact en attribut qui est passé dans un signal `clicked()` qui est émit lorsque l'on clique sur le bouton.

Signaux :
`void clicked(Contact)` : Signal émit lorsque l'utilisateur clique sur le bouton. Il émet le contact associé au bouton.
`void modifyContact(Contact)` : Signal émit lorsque l'utilisateur choisi l'option "Modifier" dans le menu contextuel du bouton.
`void deleteContact(Contact)` : Signal émit lorsque l'utilisateur choisi l'option "Supprimer" dans le menu contextuel du bouton.

3.2.8 ContactModifier

Classe héritant de la classe QDialog. Elle affiche une fenêtre de dialogue lorsqu'elle est éxecutée. Cette classe possède un contact en attribut et permet de modifier ses informations. Si les modifications sont confirmées alors les informations du contact en attribut sont remplacée par celles saisies par l'utilisateur.

3.2.9 TasksList

Classe affichant le contenu de la classe GestionTâche et permettant d'effectuer une recherche sur les tâches à faire en fonction d'un intervalle de dates.

Signal :
`void refreshTaskList(GestionTache *)` : Signal émit pour rafraîchir l'affichage de la liste des tâches.

3.2.10 ContactCreator

Fenêtre de dialogue permettant de renseigner les informations saisie par l'utilisateur dans une nouvelle instance de contact et l'ajouter à la liste des contacts si l'utilisateur confirme l'ajout.

Signal :
`void validateContact(Contact)` : Signal émit lorsque la création du contact est validée.