UBFC
UNIVERSITÉ
BOURGOGNE FRANCHE-COMTÉ

UFR
sciences & techniques

ANNÉE:2021-2022

# OPTION: CDAA .

**RÉALISE PAR :**

1.DIALLO ILIASSOU
2.BARRY THIERNO OUMAR

# Introduction

Dans le cadre de la mise en pratique des cours et des techniques appris et acquis en TP et TD au cours du module CDAA, il nous ai demandé de réaliser un projet en binôme enrichi par un apport personnel pour compléter le reste.

L'objectif de ce travail est d'approfondir les notions en développement d'application en utilisant un environnement avancé (QT) . Mais aussi de nous permettre d'apprendre à bien mener un projet depuis la phase de conception jusqu'à la réalisation finale. Cela implique une bonne gestion de temps, la gestion de stress mais surtout de bien savoir faire les liens entre les différentes classes et objets à manipuler, ce qui demande à l'étudiant un tel effort à programmer proprement pour pouvoir comprendre ses codes et aussi pouvoir l'expliquer aux autres.

L'objet de ce document est de décrire la conception et la réalisation de l'application demandée, en effectuant une description suivie d'une analyse sous forme de spécifications fonctionnelles détaillées,en utilisant du Doxygen afin de générer des commentaires adéquat et compréhensible et aussi on a proposé des fichiers export JSON pour exporter un contact,une interaction et aussi les taches .

Ainsi dans le lignes suivantes on apportera davantage d'explication sur les algorithmes principaux et sur les points techniques importants.

## I. Description du projet :

L'objet de ce projet est l'implémentation en C++ d'une application QT permettant de dialoguer avec une base de donnée SQL ,afin de gérer une base de donnée de contacts et des interactions  l'environnement QT Creator .

## II.Description de l'application :

Nous avons utilisé un interface user ( UI ) concernant la fenêtre principale et des boites de dialogues dans les sous menus pour pouvoir très bien structurés notre application .

## III. Description du programme

L'objectif de l'application est de proposer une structure de données fiable et performant pour permettre à un utilisateur d'insérer un contact dans la base de donnée , de modifier un contact ,de supprimer un contact et d'ajouter des interactions .
Dans un premier temps on a créé une base de donnée pour gérer tout ce qui concerne un  contact ,une base de donnée pour gérer l'interaction et une autre base de donnée pour gérer les taches et tags .
Dans la deuxième partie, nous avions utilisé une classe fenetreprincipale dans la quelle on a créé des boites de dialogue correspondantes aux fonctions de la base de donnée .
Et dans un dernier temps on a lié dans le main la base de donnée et la fenêtre principale.

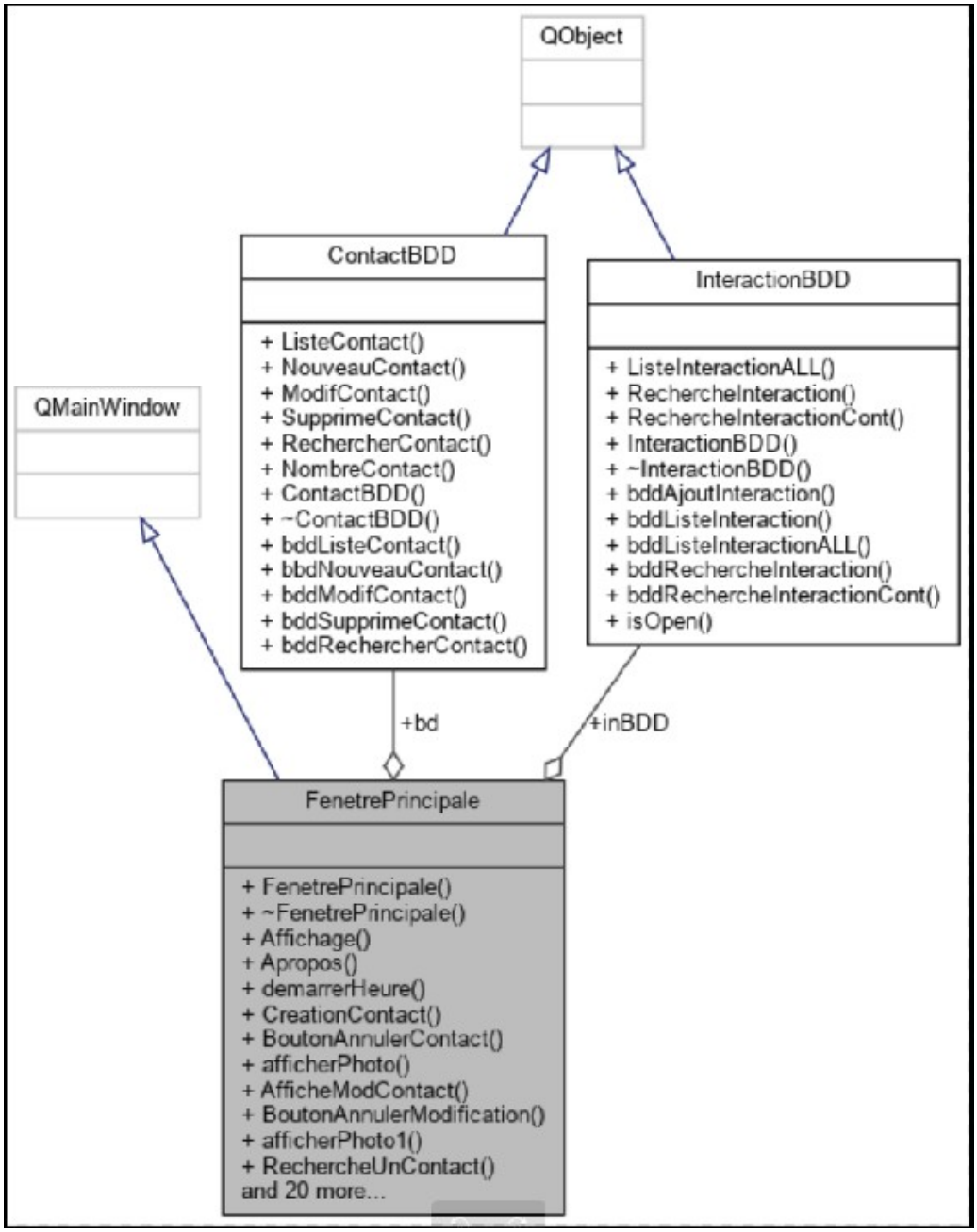
Remarque :

Dans le fichier build de notre projet on retrouve notre documents de la base de donnée .*SQLITE , et pour l'export .*JSON  .

## II. - Diagramme de conception UML:

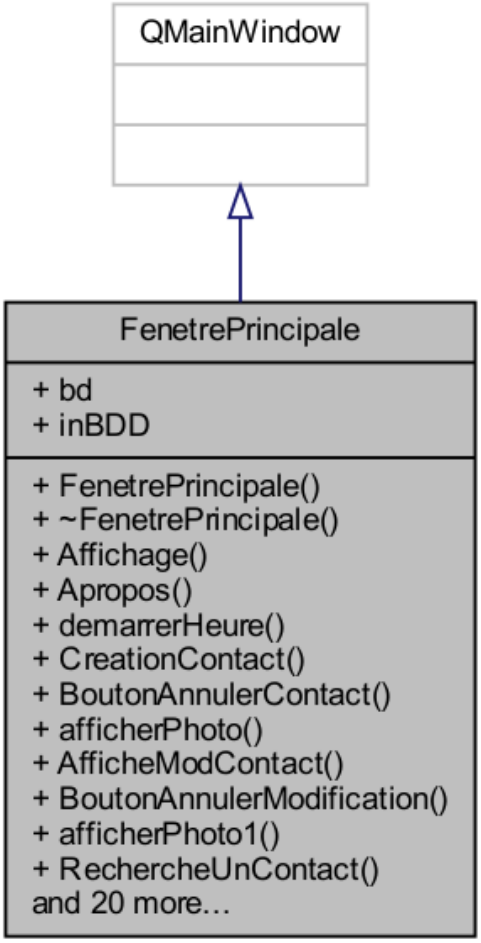
Précédemment dans notre pré-rapport du rendu jalon1 on avait fait le diagramme UML de toutes les classes c++ sans le QT et dans ce rapport on a généré le diagramme UML de la classe principale QT et les bases de données .
La classe FenetrePrincipale hérite de QmainWindow comme nous indique la flèche ci dessus et les classes ContactBDD et IntéractionBDD hérite à leur tour d'un QObjet qui sont utilisés par la classe fenêtre principale .

## III. Diagramme et Explications des classes :



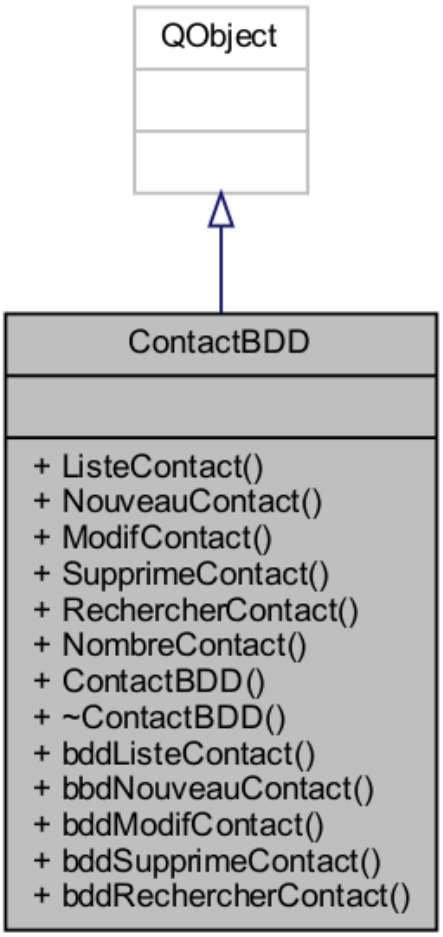
## a. La classe FenetrePrincipale :

Elle hérite d'une QmainWindow qui elle aussi hérite d'un Qwidget, qui est une structure de QT,la Qmainwindow est conçu pour gérer la fenêtre principale d'une application très complexe comme la notre.
Notre classe FenetrePrincipale nous a permis de créer des méthodes pour tout ce qui est création des boites de dialogues,fonctions,slots afin de les lier aux slots correspondants pour l'ajout,la modification,la suppression d'un contacts ou interactions .



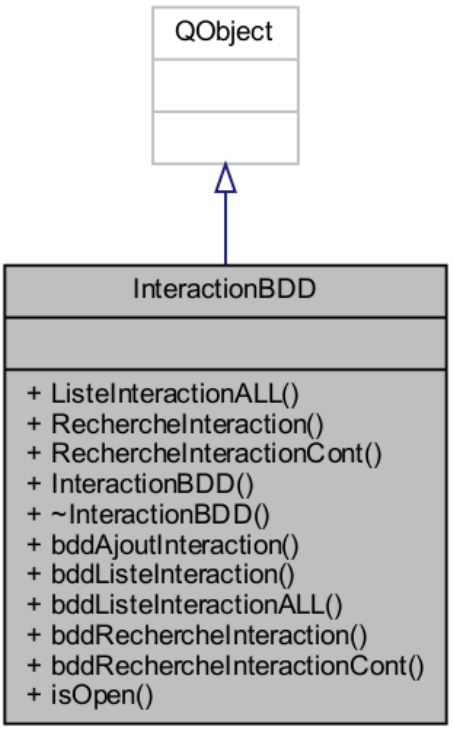
## b. La classe ContactBDD :

La classe ContactBDD hérite d'un QObject qui est le cœur du modelobject en QT .Elle est celle qui gère tout ce qui concerne un contact dans notre base de donnée,elle contient les méthodes permettant d'ajouter,modifier,supprimer,rechercher un contact dans la base et nous ramener les informations nécessaire dans le main grâce aux slots qu'on liera avec les signaux provenant de la fenêtre principale .




## c. La classe InteractionBDD :

La classe InteractionBDD hérite d'un QObject qui est le cœur du model object en QT .Elle est celle qui gère tout ce qui concerne une interaction dans notre base de donnée,elle contient les méthodes permettant d'ajouter,de rechercher des intéractions d'un contact,de rechercher des intéractions entre deux dates,de rentres des taches pour une intéraction, dans la base et nous ramener les informations nécessaire dans le main grâce aux slots qu'on liera avec les signaux provenant de la fenêtre principale.



## IV. Description de la base de donnée :

Le fichier .sqlite de la base de donnée se trouve dans le fichier build : *build-ProjetFinalCDAA-Desktop-Debug .*

On a un seul fichier.sqlite qu'on a mis dans le dossier build dans le quel on à créé toutes les tables de notre projet à savoir :

### a.Explication de la création de la table CONTACT :

si la table n'existe pas on la crée on l'a appelé **CONTACT** dans le fichier.sqlite et son attribut id_contact est la clé primaire et il doit s'incrémenter à chaque ajout d'un contact .

```
"CREATE TABLE
        IF NOT EXISTS CONTACT"
        "(id_contact integer primary key AUTOINCREMENT, "
        "nom TEXT,"
        "prenom TEXT,"
        "entreprise TEXT,"
        "mail TEXT,"
        "telephone TEXT,"
        "photo TEXT ,"
        "datecreation TEXT");
```

### b.Explication de la création de la table INTERACTION :

si la table n'existe pas on la crée on l'a appelé **INTERACTION** dans le fichier.sqlite et son attribut id_interaction est la clé primaire et il doit s'incrémenter à chaque ajout d'une interaction ,id_contact est la clé étrangère .
Et dans ce cas on a des contraintes à savoir l'identifiant du contact qui est ici est une clé étrangère dans interaction et clé primaire dans la table CONTACT ensuite on a utilisé ON DELETE CASCADE qui nous permet:
Quand on supprime le père ,tous ses enfants vont être supprimés ,donc c'est pourquoi quand on supprime un contact tout est supprimé .

```
CREATE TABLE IF NOT EXISTS INTERACTION"
                "(id_interaction integer primary key AUTOINCREMENT, "
                "id_contact integer NOT NULL,"
                "dateinteraction TEXT,"
                "contenu TEXT,"
              "FOREIGN KEY (id_contact) REFERENCES CONTACT(id_contact) ON DELETE
CASCADE )") ;
```

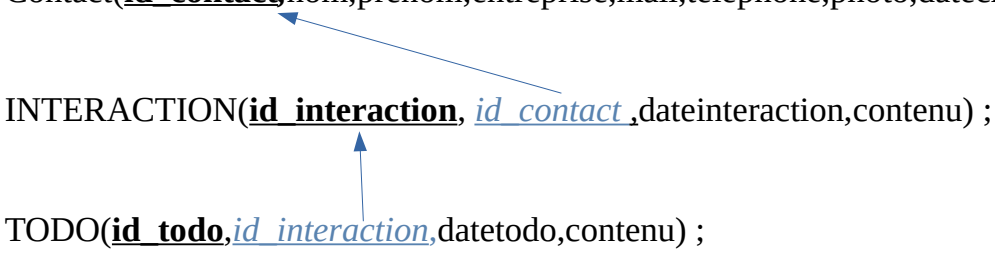
### *c.Explication de la création de la table TODO :*

si la table n'existe pas on la crée on l'a appelé ***TODO*** dans le fichier.sqlite et son attribut id_todo est la clé primaire et il doit s'incrémenter à chaque ajout d'une interaction ,id_interation est la clé étrangère .
Et dans ce cas on a des contraintes à savoir l'identifiant de l'interaction qui est ici est une clé étrangère dans TODO et clé primaire dans la table INTERACTION donc on a utilisé ON DELETE CASCADE qui nous permet:
Quand on supprime le père ,tous ses enfants vont être supprimés ,donc c'est pourquoi quand on supprime un contact tout est supprimé .

```
 CREATE TABLE
                    IF NOT EXISTS TODO"
                      "(id_todo integer primary key AUTOINCREMENT, "
                      "id_interaction integer NOT NULL,"
                      "datetodo TEXT,"
                      "contenu TEXT,"
 "FOREIGN KEY (id_interaction) REFERENCES INTERACTION (id_interaction) ON
DELETE CASCADE ");
```

## d .Modèle Relationnel des tableaux :

On a lié par les flèches les clé étrangères .



## V. Justification des choix de connexion :

Les fonctions slots qu'on a implémenté dans la base de donnée pour pouvoir créer,supprimer,modifier un contact ou ajouter des interactions et taches sont récupérés dans le main.cpp de notre programme on lui crée la connexion en lui liant avec le signal émis dans la fenêtré principale dans le slot correspondant à la tache qu'on veut faire .

***Exemple de connexion avec la création d'un contact : dans le main :***

```
int main(int argc, char *argv[])
{
 QApplication a(argc, argv); /*!< Initialisation de a avec les parametres argc et argv */
 FenetrePrincipale w; /*!< déclaration de w de type fenetre principale qui est une classe */
 ContactBDD c;//on crée une variable c de type contactBDD ;
```

On met &w pour dire que le signal viendra de la fenêtre principale en mettant **sigInsert()** avec pour paramètre un **contact** .

Ensuite on met la variable de la base de donnée en lui liant au slot qui se trouve dans la BD qui sert à créer un contact dans la base qui lui aussi à un paramètre **contact** .

Exemple :

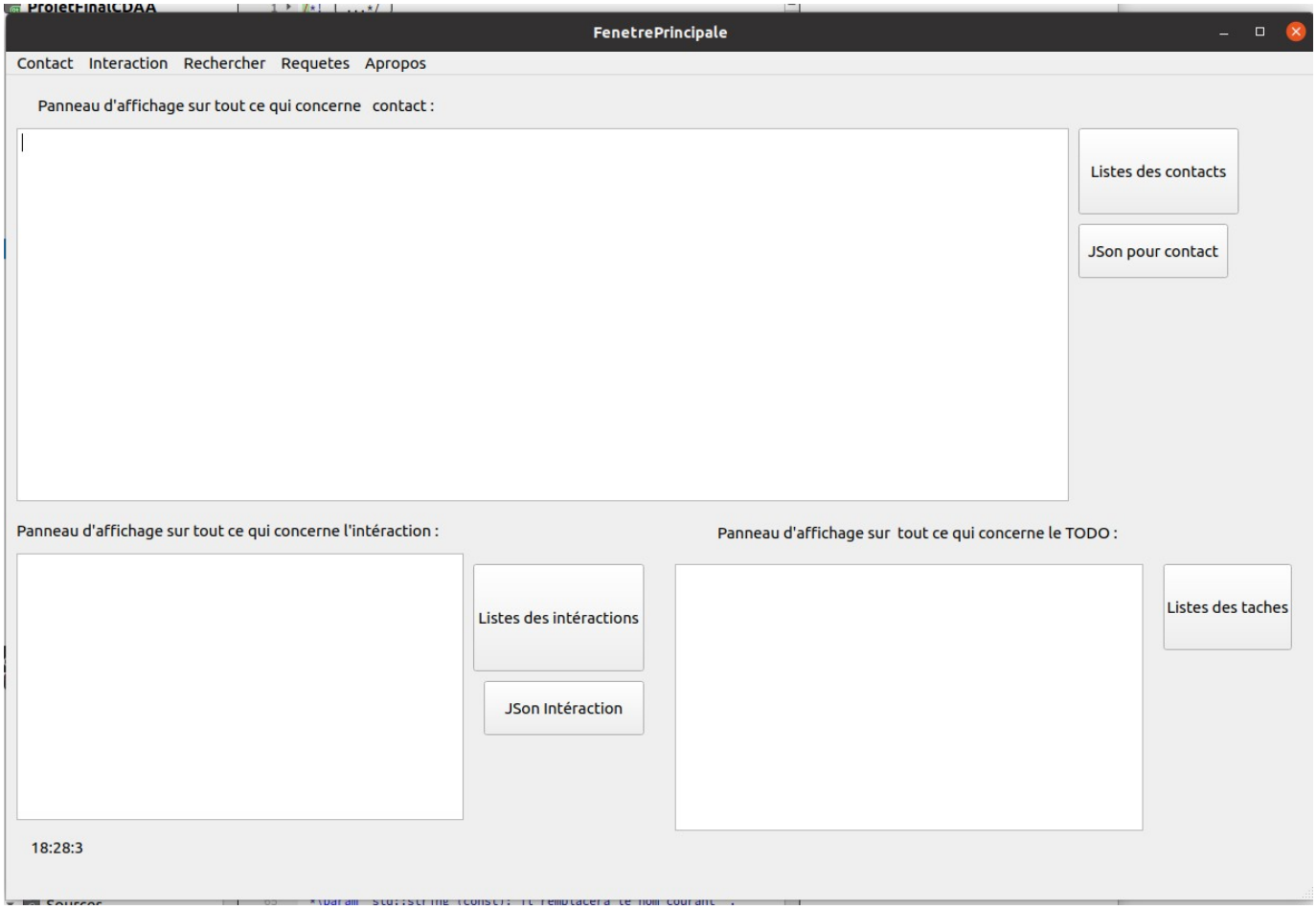
```
connect(&w,SIGNAL(sigInsert( Contact &)),&c,SLOT( bbdNouveauContact(Contact &)));

}
```

**Attention!!**

Nous avions remarqué en faisant une connexion il faut forcement que la fonction qui représente le **SIGNAL** et la fonction qui représente le **SLOT** doivent avoir les même type et nombre de paramètre .

# VI. Fonctionnement de notre application:

## Aperçu général:




**Premiere vue :**

A l'exécution de notre programme voilà l'interface principale qui s'affiche elle est structuré comme suit pour afficher tout ce qui concerne un contact on a le textEdit en haut avec les boutons à droite,pour afficher les interactions on a le textEdit du bas à gauche et pour les taches l'interface en bas à droite .

On a les boutons Json contacts et interaction pour pouvoir exporter la liste des contacts et la liste des interactions .

**les Sous Menus :**

On a fait de tel sorte que si on veut ajouter un contact ou modifier ou supprimer un contact il faut cliquer avec la souris sur le menu «Contact » et on aura une liste de sous menus et on choisira ce qu'on veut exécuter .

Lors de la **création** d’un **contact :** on remplis toutes les cases sinon on reçoit un message d’ Erreur de bien remplir toutes les cases et aussi on a le bouton parcourir pour ajouter une photo.

Pour la **Modification** d’un **contact :**

on aura une comboBox avec des identifiants lors du clic sur l’identifiant de la comboBox tous les champs seront pré-remplis pour la modification du contact choisis .

Pour la **suppression** d’un **contact** :

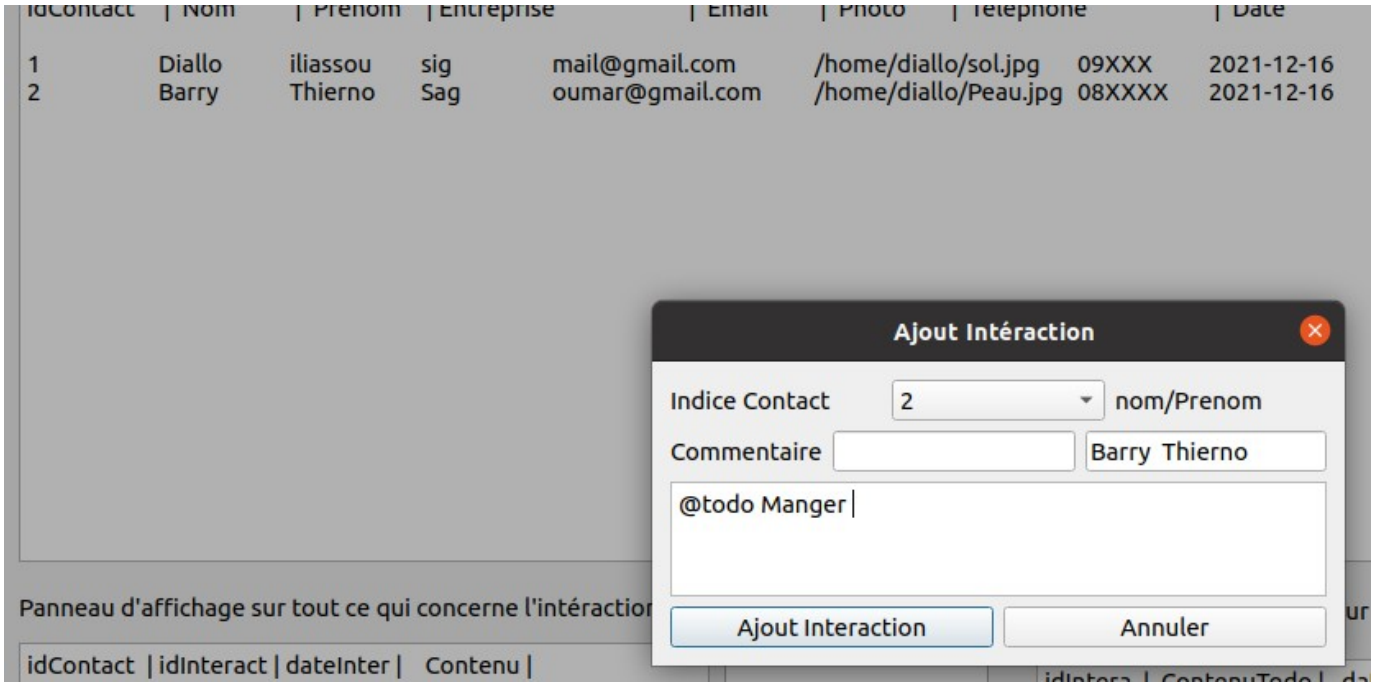
on a une comboBox lors du clic sur la comboBox on voit le nom,prenom et l’identifiant du contacts qu’on veut supprimer .

En régardant bien l’image ci dessous on verra bien que lorsqu’on clique sur la comboBox on recupere le nom et prenom du contact qu’on veut supprimer et on le supprime et la base de donnée se met à jour automatiquement .

**Pour l’interaction :**

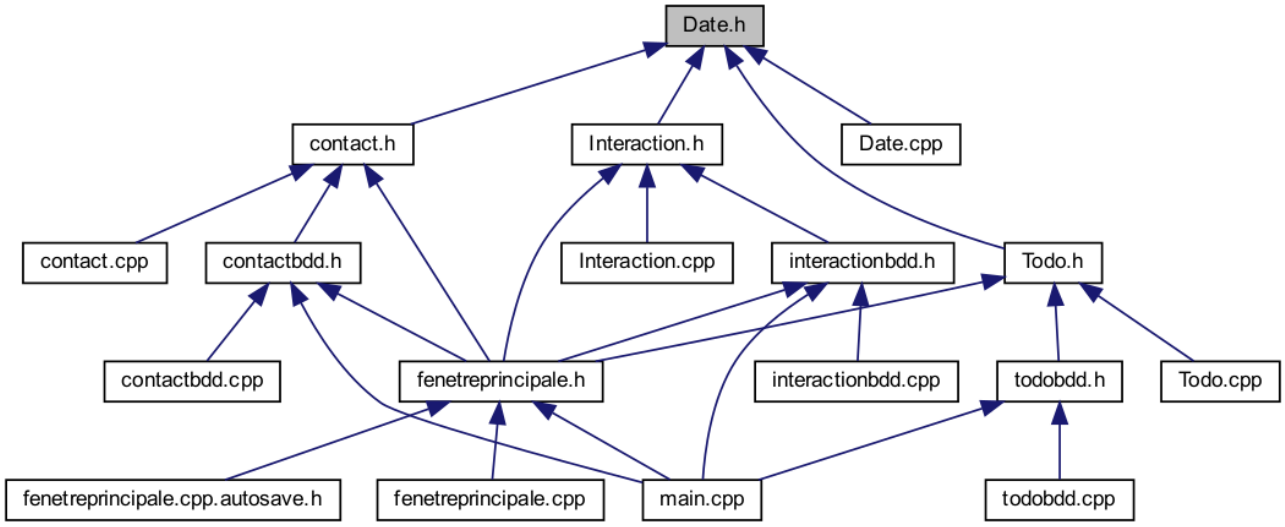
On a une comboBox pour récupère l’identifiant du contact  en cliquant sur la combo on aura son nom et prénom pour le quel on veut ajouter une tache ou une interaction,toutes les interactions sont horodaté automatique lors de sa création ,si on l’ajoute le tag dans le textEdit ,si on met pas  la date dans la tache on aura comme tache urgente (à faire maintenu) .

**Voir Image ci dessous pour l’exemple illustratif de l’ajout d’une tache et interaction :**




**C’est de la même manière comme nous avons fait ci dessus  notre application fonctionne .**

## VII. Diagramme générale de classe :




## VIII. Conclusion :

Ce projet QT nous a permis d'approfondir nos connaissances en langage C++ ,de découvrir de plein l'application QT .
Nous avons acquis beaucoup de connaissances sur les bases de données : comment créer une Base de donnée,comment connecter une classe c++ avec une base de donnée Sql ,comment les faire interagir ,comment utiliser le Doxygen,comment faire des exports sur des fichiers avec JSON .
Ce projet a été une grande chance pour nous car on a découvert le bonheur de la programmation .