

Rapport du projet :

**Conception et
développement avancé
d'application**

**Wassim Ennaji
Choukri Imane**

Sommaire

- La MainWindow
 - La recherche
 - Ajouter un Contact
 - Afficher un Contact
 - Modifier un Contact
 - Supprimer un Contact
- Ajouter une interaction et ses tâches
 - La base de données
 - Export json
 - Doxygen et Doxyfile
 - Diagramme de classes

La MainWindow

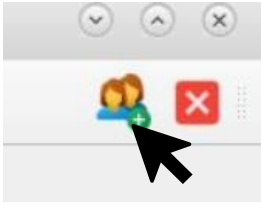
La classe MainWindow définit la fenêtre principale de l'application et gère diverses fonctionnalités, y compris l'interaction avec la base de données, l'affichage des contacts, et la gestion des recherches.



- Son constructeur initialise l'interface utilisateur (ui), crée une instance de la classe `basededonnee` (bd), remplit la liste des contacts (`ui->listWidget`) et affiche le nombre de contacts avant la recherche sur le nombre de contacts après la Recherche .
- Le destructeur libère la mémoire utilisée par l'interface utilisateur.
- La méthode **remplirListWidget** établit des connexions entre les signaux et les slots, elle récupère la liste des contacts à partir de la base de données en appelant la méthode `bd->getContacts()`. Ensuite, elle parcourt cette liste et ajoute chaque contact au widget de liste (`ui->listWidget`). De plus, elle émet le signal **cherchNBC()**, déclenchant ainsi le calcul du nombre total de contacts dans la base de données par le biais du slot **trouverNb()** de l'objet `bd`.
- La méthode **affichNb** de la classe **MainWindow** est un slot réagissant au signal émis par la classe **basededonnee** lors du calcul du nombre total de contacts. Elle actualise dynamiquement l'interface utilisateur en effaçant le contenu existant du widget de texte associé au nombre total de contacts (`label_NBC`) et en y affichant le nouveau total.
- La méthode **modifierListC** de la classe **MainWindow** orchestre la mise à jour en temps réel de l'interface utilisateur suite à des modifications dans la liste des contacts.

Ajouter un Contact

Au niveau de la MainWindow :



```
void MainWindow::on_actionAjouter_contact_triggered()
{
    aj=new AjoutContact();
    aj->show();
    connect(aj,SIGNAL(enregistrerContact(Contact *)),bd,SLOT(ajoutContact(Contact*)));
    connect(aj,SIGNAL(majList()),this,SLOT(modifierListC()));//update de la liste
}
```

Lorsque l'utilisateur appuie sur le bouton déclenchant l'action associée à la fonction **on_actionAjouter_contact_triggered()** dans la classe **MainWindow**:

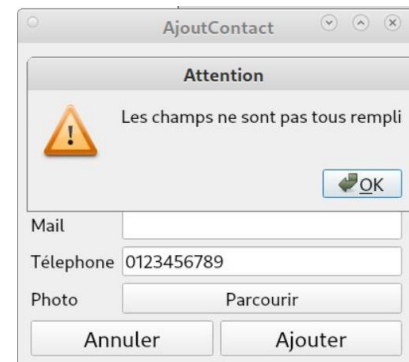
- un objet de la classe **AjoutContact** est créé et affiché, montrant ainsi une nouvelle fenêtre à l'utilisateur pour ajouter un contact.
- La première connexion est réalisée entre le signal **enregistrerContact** émis par l'objet **AjoutContact** et le slot **ajoutContact** de l'objet **bd**. Ainsi, lorsque l'utilisateur enregistre un contact dans la fenêtre **AjoutContact**, le signal **enregistrerContact** est émis et déclenche l'appel de la fonction **ajoutContact** de l'objet **bd**. Cette fonction **ajoutContact** de l'objet **bd** est responsable d'ajouter le contact à la base de données.
- La deuxième connexion est établie entre le signal **majList** émis par l'objet **AjoutContact** et le slot **modifierListC** de l'objet **MainWindow**. Lorsque la fenêtre **AjoutContact** émet ce signal, la fonction **modifierListC** de la classe **MainWindow** est invoquée. Cette fonction assure la mise à jour du widget de liste dans la fenêtre principale.

Au niveau de la classe AjoutContact:

Lorsque l'utilisateur clique sur le bouton "Enregistrer", la fonction **on_pushButton_clicked()** est appelée. Avant d'enregistrer le contact, cette fonction vérifie que tous les champs obligatoires sont remplis. Si c'est le cas, elle crée un nouvel objet **Contact**, remplit ses propriétés avec les données saisies dans l'interface utilisateur, émet des signaux (**enregistrerContact** et **majList**) puis ferme la fenêtre. En cas de champs manquants, un avertissement est affiché à l'utilisateur.

On vérifie que le numéro de téléphone saisi est composé uniquement de chiffres et a une longueur spécifique de 10 caractères.

En appuyant sur le bouton "Parcourir", l'utilisateur choisit une photo à lier au nouveau contact. On utilise donc une boîte de dialogue de fichier pour récupérer le chemin de la photo sélectionnée.



La recherche :

Suite à l'appui sur le bouton "Appliquer"

• Recherche par Nom :

Si la case à cocher "Nom" est activée (**clickedNom == true**) et que l'utilisateur a saisi un nom dans le champ correspondant, alors une recherche par nom est effectuée. La méthode **searchContactsByNom** de l'objet **bd** est appelée avec le nom saisi, et les résultats de la recherche sont stockés dans une liste appelée **contacts**. Ensuite, chaque contact de cette liste est ajouté à la **listWidget**, affichant ainsi les résultats de la recherche par nom à l'utilisateur et mettant à jour le nombre de contacts de la liste.

• Recherche par Entreprise:

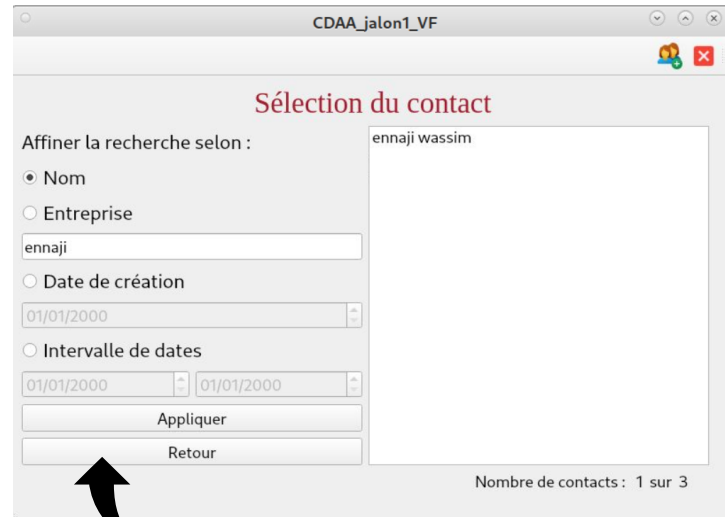
De la même manière si la case à cocher "Entreprise" est activée la méthode **searchContactsByEntreprise** de l'objet **bd** est appelée avec le nom de l'entreprise saisi, et les résultats de la recherche sont stockés dans une liste appelée **contacts**.

• Recherche par Date:

Si la case à cocher "Date de création" est activée (**clickedDate == true**) la date est obtenue à partir d'un widget de date (**ui->dateEdit->text()**), et la méthode **searchContactsByDate** de l'objet **bd** est utilisée pour récupérer les contacts correspondant à cette date. Les résultats sont ajoutés à la liste **contacts**, et chaque contact de la liste est ensuite ajouté au widget de liste, permettant à l'utilisateur de voir les contacts associés à la date spécifiée.

• Recherche par Intervalle:

Si la case "Intervalle de dates" est activée (**clicked2Date == true**), le système procède à une recherche basée sur un intervalle de dates. Les dates de début et de fin sont extraites de deux widgets de date distincts (**ui->dateEditdebut->text()** et **ui->dateEditfin->text()**). La méthode **searchContactsByDateRange** de l'objet **bd** est ensuite utilisée pour récupérer les **contacts** compris dans cette **plage de dates** spécifiée. Les résultats sont consignés dans une liste appelée **contacts**, et chaque contact de cette liste est intégré au widget de liste.



Le bouton "**Retour**" nous permet de réafficher la liste des contacts de notre base données initiale .



Si aucun contact ne s'affiche suite à la recherche menée par l'utilisateur on affiche une boîte de dialogue d'information contenant comme message principal : "**Aucun contact ne correspond à cette recherche**"

Afficher un Contact

Au niveau de la MainWindow :

choukri imane
ennaji wassim
test test
NOM PRENOM



La sélection :

Lors d'un double-click sur un des contacts de la listWidget, le texte de l'élément actuellement sélectionné dans la liste **listWidget** est extrait et analysé pour séparer le nom et le prénom du contact.

Ensuite, un signal **select** est émis avec le nom et le prénom du contact sélectionné en utilisant la fonction **emit**.

L'affichage :

```
//recupere le contact sélectionné
string l=ui->listWidget->currentItem()->text().toStdString();
string nomW="";
string prenomW="";
bool espaceRencontre = false;

for (char c : l) {
    if (c != ' ' && !espaceRencontre) {
        nomW+= c;
    } else if (c == ' ') {
        espaceRencontre = true;
    } else {
        prenomW += c;
    }
}

emit select(nomW,prenomW);
```

```
void MainWindow::on_listWidget_doubleClicked()
{
    fc=new FormContact();
    fc->show();

    //selection
    connect(this, SIGNAL(select(string,string)), bd, SLOT(reception(string,string)));
    //affichage
    connect(bd, SIGNAL(envoyerContact(Contact *,list<Interaction *>,list<list<tache*>>)),fc, SLOT(receiveContactf(Contact *,list<Interaction *>,list<list<tache*>>)));
}
```

- On commence par créer et afficher une nouvelle instance de la classe **FormContact**, qui est destinée à présenter les actions (affichage , gestion: modification , ajout interaction ou suppression) qu'on peut effectuer au contact sélectionné .
- Une connexion est établie entre la MainWindow et la classe **basededonnee (bd)** en émettant le signal **select** avec le nom et prénom du contact sélectionné.
- Une connexion est également établie entre la classe **basededonnee (bd)** et l'instance de **FormContact (fc)** en émettant le signal **envoyerContact** avec les données du contact (le contact , une liste de ses interactions et une liste de liste des taches de chaque interaction) .



Au niveau de la classe FormContact :

- Une fois la basededonnee émet le signal **envoyerContact** avec le contact sélectionné le slot **receiveContactf** a pour rôle de recevoir et de stocker les données relatives à ce contact, ses interactions et les tâches de chacune de ses interactions .
- Lorsque l'utilisateur appuie sur le bouton "**Afficher**" dans **FormContact** , un nouvel objet de la classe **AfficherContact** (**afc**) est créé.
- Une connexion est établie entre **FormContact** et **AfficherContact** (**afc**) en émettant le signal **AffichC** avec les données du contact reçu depuis la base de donnée suite à sa sélection .
- Le signal **AffichC** est émis pour transmettre les données du contact (**c**, **li**, **lt**) à l'objet **AfficherContact** (**afc**).

Au niveau de la classe affichercontact:

- Lorsque le signal AffichC est émis, la fonction receiveContact est activée. Les informations du contact telles que le nom, le prénom, l'adresse e-mail, l'entreprise, etc., sont extraites et présentées dans les champs appropriés tels que QLineEdit, et l'image associée est affichée dans un QPushButton.
- En parcourant la liste d'interactions (**li**), chaque interaction est affichée dans un **QListWidget**. Chaque élément de la liste est formaté de manière à inclure le texte de l'interaction ainsi que sa date.
- En cliquant sur le bouton chercher , ceci nous affichera que les interaction entre les deux dates saisies .
- En appuyant sur "Retour " on revient à notre listes des interactions initiale .
- Lorsqu'un utilisateur double-clique sur un élément de la liste des interactions , la fonction **on_listWidget_itemDoubleClicked** est déclenchée. Les tâches liées à l'interaction sélectionnée sont extraites et présentées dans un **QTextEdit**. Si une tâche a une date correspondant à la date actuelle, cela est spécifié. Par "AJOURDHUI" .

The screenshot shows a window titled "AfficherContact" with a subtitle "Affichage du contact sélectionné". It contains several input fields for contact information: Nom (choukri), Prénom (imane), Entreprise (IKEA), Mail (im@gmail.com), Téléphone (0705589665), Date de création (8/12/2023), and Date de modification (18/12/2023). Below these is a list of interactions, with "INTERACTION2 @date 11/12/2023" selected. A magnifying glass highlights the "Nombre d'interactions: 2" label. At the bottom, there are date pickers for "Interactions entre" (01/01/2000 to 01/01/2000) and a "Chercher" button. A "Retour à toutes les interactions" button is also present. Below the interactions list, there is a list of tasks: "@todo TACHE1 @date 01/01/2024", "@todo TACHE2 @date 11/12/2023", and "@todo TACHE3 @date 11/12/2024". A magnifying glass highlights the "Nombre de tâches: 3" label. At the very bottom is an "Annuler" button.

- Si une interaction comporte des tâches associées, le nombre de tâches est également indiqué, de même que le nombre total d'interactions.
- Si l'interaction n'a aucune tâche , alors le texte "Cette interaction n'a aucune tâche" est affiché dans le **QTextEdit**.

Modifier un Contact

Au niveau de la Main Window :


On établit une connexion entre le signal `sendEbdd` émis par l'objet `fc` et le slot `reciveE` de l'objet `bd` qui met à jour les informations du contact sélectionné selon les modifications apportées.

Au niveau de la classe `FormContact` :

- Lorsque l'utilisateur appuie sur le bouton de modification (`on_pushButtonModifier_clicked`), un nouvel objet de la classe `FormModifContact` (`fmc`) est créé et affiché.
- Une connexion signal-slot est établie entre l'objet de la classe **`FormContact` (`this`)** et l'objet de la classe **`FormModifContact` (`fmc`)**. Ce signal est émis pour envoyer le contact actuel (`c`) à la classe **`FormModifContact`**, afin d'afficher les informations initiales du contact qu'on a sélectionné et qu'on veut modifier.
- Une connexion signal-slot est également établie entre l'objet de la classe **`FormModifContact` (`fmc`)** et l'objet de la classe **`FormContact` (`this`)**. Ce signal est émis lorsque des modifications sont confirmées dans la classe **`FormModifContact`**.
- Enfin, le signal **`sendCedit`** est émis pour envoyer le contact actuel sélectionné (`c`) à la classe **`FormModifContact`**.

Au niveau de la classe `FormModifContact` :

- Lorsque le signal `sendCedit` est reçu, la fonction `affichCm` est déclenchée. Cela permet d'afficher les informations du contact dans les champs de saisie de la classe `FormModifContact`.
- L'utilisateur peut alors modifier les informations du contact.
- On vérifie que le numéro de téléphone saisi est composé uniquement de chiffres et a une longueur spécifique de 10 caractères.
- En cliquant sur le bouton "Parcourir" (`on_pushButtonParcourir_clicked`), l'utilisateur peut sélectionner une nouvelle photo pour le contact.



The screenshot shows a window titled "ModifContact" with a close button. Inside, the text "Saisissez vos modifications" is displayed in red. Below this, there are several input fields for contact information: "Nom" (choukri), "Prénom" (imane), "Entreprise" (bmw), "Mail" (im@gmail.com), "Date de création" (8/12/2023), and "Téléphone" (0705589665). There is also a "Photo" field with a "Parcourir" button next to it. At the bottom, there are two buttons: "Annuler" and "Enregistrer".

- Lorsque l'utilisateur clique sur le bouton "Enregistrer" (on_pushButton_Enregistrer_clicked), un signal sendE est émis pour informer la classe FormContact des modifications apportées au contact.
- Selon le fait que l'utilisateur ait choisi une nouvelle photo (clicked est vrai) ou non, le signal sendE est émis avec les nouvelles données ou avec l'ancien chemin de la photo.
- Un message est affiché pour informer l'utilisateur que le contact a été modifié avec succès.
- L'utilisateur doit remplir tous les champs nécessaires avant de procéder à une modification.

Affichage du contact sélectionné

Nom	choukri
Prénom	imane
Entreprise	IKEA
Mail	im@gmail.com
Téléphone	0705589665
Date de création	8/12/2023
Date de modification	18/12/2023

Placeholder for a profile picture.

Ici on a modifié l'entreprise, on remarque que la date de modification change en fonction du jour pendant lequel la modification a été effectuée.



Supprimer un Contact

Au niveau de la MainWindow :

- On établit une connexion entre le signal deleteC émis par l'objet fc et le slot deleteCbd de l'objet bd qui est responsable de la suppression effective du contact, ses interactions et les tâches liées à ses interactions de la base de données.
- On établit une connexion entre le signal **deleteC** émis par l'objet **fc** et le slot **modifierListC** de l'objet **MainWindow** qui chargé de mettre à jour visuellement la liste des contacts affichée à l'utilisateur après la suppression d'un contact.

Au niveau de la classe FormContact :

- Lorsque l'utilisateur appuie sur le bouton "Supprimer" (on_pushButtonSupprimer_clicked), une boîte de dialogue (QMessageBox) est affichée pour confirmer la suppression.
- Si l'utilisateur confirme (QMessageBox::Yes), le signal supprimerC est émis.
- Lorsque le signal **supprimerC()** est émis, le slot **deleteeC()** sera appelé.
- La fonction deleteeC émet ensuite le signal deleteC(c) pour informer que le contact actuel doit être supprimé.
- La fenêtre de FormContact est fermée.



Ajouter une interaction et ses tâches

Au niveau de la MainWindow :

Deux connexions sont établies entre l'objet de la classe FormContact (fc) et l'objet de la classe basedonnee (bd):

- La première connexion "**connect(fc, SIGNAL(envoyerInteractionEbdd(Interaction *)), bd, SLOT(recInteractionE(Interaction *)))**;" envoie une interaction à la base de données.
- La deuxième connexion "**connect(fc, SIGNAL(envoyerTacheEbdd(list<tache*>)), bd, SLOT(recTacheE(list<tache*>)))**;" envoie une liste de tâches à la base de données.

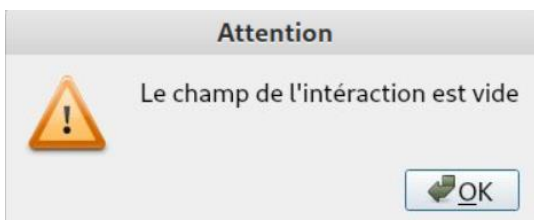
Au niveau de la classe FormContact :

- Lorsque l'utilisateur appuie sur le bouton "Ajouter Interaction" (on_pushButtonAjoutInter_clicked), une nouvelle fenêtre AjoutInteraction (ai) est créée et affichée.
- Des connexions sont établies entre la fenêtre AjoutInteraction et les slots de la classe FormContact pour recevoir les interactions et les tâches ajoutées dans la fenêtre AjoutInteraction.
- La fonction **recInteractionE** est appelée lorsque l'interaction est ajoutée dans la fenêtre AjoutInteraction. Cette fonction émet ensuite un signal (**envoyerInteractionEbdd**) pour informer la base de données (**bd**) de cette nouvelle interaction.
- La fonction **recTacheE** est appelée lorsque les tâches sont ajoutées dans la fenêtre AjoutInteraction. Cette fonction met à jour la liste de tâches dans la classe **FormContact** et émet un signal (**envoyerTacheEbdd**) pour informer la base de données (**bd**) de ces nouvelles tâches.

Au niveau de la classe AjoutInteraction :

Ajout de l'interaction :

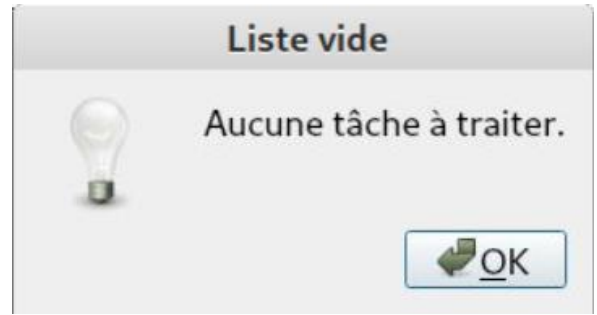
- Lorsque l'utilisateur appuie sur le premier bouton "Enregistrer", une nouvelle interaction (I) est créée.
- Si le champ de l'interaction n'est pas vide, l'interaction est configurée avec le texte fourni et la date du jour actuel puis le signal **envoyerInteractionE** est émis.
- La connexion entre AjoutInteraction et FormContact assure que l'interaction est transmise à la classe FormContact, qui va ensuite la transmettre à la mainwindow.
- Si le champ de l'interaction est vide, une boîte de dialogue d'avertissement est affichée.



Ajout des tâches au niveau de cette interaction :

Lorsque l'utilisateur appuie sur le bouton "enregistrer" dans la fenêtre AjoutInteraction, les étapes suivantes sont exécutées pour traiter le contenu du champ de texte :

- Le contenu du champ `textEditContenu` est récupéré sous forme de chaîne de caractères (`QString`) dans une variable **content**.
- Ensuite, on vérifie si ce contenu est vide. Si c'est le cas, une boîte de dialogue informe l'utilisateur qu'il n'y a pas de tâches à traiter, et la fonction se termine.
- Si le contenu n'est pas vide, on utilise des expressions régulières pour extraire les informations pertinentes des lignes commençant par "@todo" dans le texte. On utilise également "@date" pour valider le format des dates. Les informations extraites comprennent l'événement et la date associée à chaque tâche.
- En parcourant le texte ligne par ligne, on crée des objets de type **tache*** pour chaque tâche détectée, en utilisant les informations extraites. Ces objets sont ensuite ajoutés à une liste appelée **listeTaches**.
- On effectue également une validation de la date, détectant les dates invalides et les ajoutant à une liste appelée **invalidDates**.
- Après avoir traité toutes les lignes, on affiche une boîte de dialogue d'avertissement si des dates invalides ont été détectées, listant les événements associés.
- Ensuite, on émet un signal avec la liste des tâches créées, et le contenu du `QTextEdit` est effacé.



- On ajoute une indication utile pour rappeler le format attendu des informations à saisir dans le champ `textEditContenu`.



- Si la variable date est vide. Si c'est le cas, cela signifie que la ligne "@todo" extraite n'inclut pas de date associée. On remplace cette absence par la date actuelle du système automatiquement.

La base de données

Table "Contact":

- "idC" : **Clé primaire** de type INTEGER, avec contrainte UNIQUE et AUTOINCREMENT.
- "nom" : Champ de texte (TEXT) pour le nom du contact.
- "prenom" : Champ de texte (TEXT) pour le prénom du contact.
- "email" : Champ de texte (TEXT) pour l'adresse e-mail du contact.
- "entreprise" : Champ de texte (TEXT) pour le nom de l'entreprise du contact.
- "dateC" : Champ de type DATE pour la date de création du contact.
- "dateM" : Champ de type DATE pour la date de dernière modification du contact.
- "photo" : Champ de texte (TEXT) pour le chemin de la photo du contact.
- "telephone" : Champ de texte (TEXT) pour le numéro de téléphone du contact.

Table "Interaction":

- "idl" : **Clé primaire** de type INTEGER, avec contrainte UNIQUE et AUTOINCREMENT.
- "idC" : *Clé étrangère* faisant référence à "idC" dans la table "Contact".
- "contenu" : Champ de texte (TEXT) pour le contenu de l'interaction.
- "dateinteraction" : Champ de type DATE pour la date de l'interaction.

Table "Todo":

- "idT" : **Clé primaire** de type INTEGER, avec contrainte UNIQUE et AUTOINCREMENT.
- "idl" : *Clé étrangère* faisant référence à "idl" dans la table "Interaction".
- "tache" : Champ de texte (TEXT) pour la description de la tâche.
- "dateTodo" : Champ de type DATE pour la date prévue de la tâche.

La classe basededonnee :

La classe basededonnee joue un rôle crucial en tant qu'interface centrale entre notre application et la base de données SQLite. Elle offre des méthodes permettant d'effectuer diverses opérations, notamment la récupération, la recherche, l'ajout et la suppression de données liées aux contacts. Elle gère les transactions et propose des fonctionnalités pour obtenir des informations statistiques sur la base de données. Ainsi, la classe basededonnee constitue un point d'accès centralisé et organisé pour toutes les interactions avec la base de données dans le contexte de notre application.

- Le constructeur de la classe initialise une connexion à la base de données SQLite. Il utilise **QSqlDatabase** et spécifie le chemin du fichier de base de données.
- Le destructeur ferme la connexion à la base de données lorsque l'objet est détruit.

Fonctions pour Récupérer des Contacts lors de la recherche :

- **getContacts()** : Exécute une requête SQL pour récupérer tous les contacts de la table "Contact" et les stocke dans une liste:

```
query.prepare("SELECT * from Contact");
```
- **searchContactsByNom()** et **searchContactsByEntreprise()** : Recherchent des contacts par nom ou entreprise et renvoient une liste de noms complets correspondants:

```
query.prepare("SELECT * FROM Contact WHERE nom = :nom");  
query.prepare("SELECT * FROM Contact WHERE entreprise = :entreprise");
```
- **searchContactsByDate()** et **searchContactsByDateRange()** : Recherchent des contacts par date de création, soit une date spécifique, soit dans une plage de dates:

```
query.prepare("SELECT * FROM Contact WHERE dateC = :date1");  
query.prepare("SELECT * FROM Contact WHERE dateC BETWEEN :dateDebut AND :dateFin");
```
- Même logique pour la recherche des interactions entre deux dates .

Ajouter un contact :

A chaque fois que le signal **enregistrerContact** est émis par l'objet aj de classe **AjoutContact** , la fonction **ajoutContact** de l'objet **bd** est appelée. Le rôle de ce slot est d'ajouter un contact à la base de données. Il prend un pointeur vers un objet **Contact** en tant que paramètre, extrait les informations du contact, puis exécute une requête SQL pour insérer ces informations dans la table **Contact** de la base de données.

```
query.prepare("INSERT INTO Contact  
(nom,prenom,email,entreprise,dateC,dateM,photo,telephone) VALUES  
( :n, :p, :m, :e, :dc, :dm, :ph, :t)");
```

Reception et émission du contact sélectionné afin de l'afficher :

Le slot reception dans votre classe basededonnee est connecté au signal select émis par la mainwindow à chaque fois qu'un contact est sélectionné . Ce slot effectue une recherche dans la base de données pour récupérer les informations sur ce contact spécifique, ainsi que les interactions et les tâches associées à ce contact.

Une première requête SQL est préparée pour sélectionner toutes les informations du contact dans la table Contact qui correspondent aux critères de nom et prénom fournis.

```
query.prepare("SELECT * FROM Contact WHERE nom = ' " + nom + "  
AND prenom = ' " + prenom + "';");
```

Ces informations sont utilisées pour créer un objet de type Contact (c).

- Une deuxième requête est préparée pour récupérer l'ID (idC) du contact nouvellement créé. Cet ID sera utilisé pour rechercher les interactions liées à ce contact.

```
selectQuery.prepare("SELECT idC FROM Contact WHERE nom = '" + nom + "' AND prenom = '" + prenom + "';");
```
- Une troisième requête est préparée pour sélectionner toutes les interactions associées à l'ID du contact (idC).

```
query.prepare("SELECT * FROM Interaction WHERE idC = :idC");
```
- Les résultats de cette requête sont parcourus dans une boucle, et pour chaque interaction, un objet **Interaction** est créé et ajouté à une liste (**li**).
- Une requête est préparée pour chaque interaction afin de récupérer les tâches associées à cette interaction à partir de la table **Todo**.

```
tacheQuery.prepare("SELECT * FROM Todo WHERE idI = :idI");
```
- Les résultats de ces requêtes sont parcourus, et pour chaque tâche, un objet **tache** est créé et ajouté à une liste de tâches (**taches**).
- Cette liste de tâches est elle-même ajoutée à une liste de listes de tâches (**listTaches**).
- Après avoir récupéré les informations du contact, les interactions et les tâches associées, le slot émet un signal (**envoyerContact**) avec le contact **c**, la liste des interactions **li** et la liste des tâches **listTaches** en paramètres. Ce signal est connecté au slot qui affiche le contact, ses interactions et les tâches de chaque interaction.

Modifier un contact :

Lorsqu'un contact est modifié, un signal nommé **sendEbdd** est émis. Ce signal est connecté au slot **reciveE** de la classe **basededonnee** :

- Un nouvel objet de la classe Contact (**c1**) est créé.
- Les méthodes **getnom** et **getprenom** de notre contact local **c** sont appelées pour obtenir le nom et le prénom du contact sélectionné.
- Une requête SQL est préparée pour mettre à jour le contact dans la base de donnée. Cette requête utilise les paramètres fournis (**N,P,E,m, tele, photo**) pour mettre à jour les champs appropriés dans la base de données.

```
query.prepare("UPDATE Contact SET nom=:N,prenom=:P,email=:m,entreprise=:e,dateM=:dm,telephone=:tele,photo=:p WHERE nom = '" + n + "' AND prenom = '" + p + "';");
```
- On émet un signal **emit modifnomprenom()** qui est connecté dans la mainwindow à un slot **modifierListC** qui met à jour la liste affichée des contacts, cette fois-ci avec les noms et prénom modifiés si c'est le cas.
- En résumé, la fonction **reciveE** gère la mise à jour d'un contact dans la base de données en exécutant une requête SQL de mise à jour.

Ajout d'une interaction :

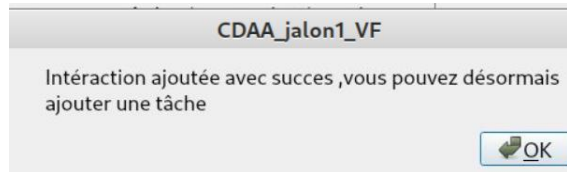
Lorsqu'une interaction est ajoutée à contact, un signal nommé **envoyerInteractionEbdd(Interaction *)** est émis. Ce signal est connecté au slot **recInteractionE(Interaction *)** de la classe **basededonnee**.

Voici une explication détaillée de chaque étape de ce slot :

- Une requête SQL préparée (**selectQuery**) est utilisée pour récupérer l'ID du contact dans la table **Contact** en fonction du nom et du prénom du contact sélectionné. Ces noms sont d'abord convertis en chaînes de type **QString** (**nomW1** et **prenomW1**). :

```
selectQuery.prepare("SELECT idC FROM Contact WHERE nom = :nomW1  
AND prenom = :prenomW1");
```
- Une requête d'insertion SQL préparée (**insertQuery**) est utilisée pour insérer une nouvelle interaction dans la table **Interaction**.
- Les valeurs à insérer comprennent l'ID du contact (**idC**), le contenu de l'interaction (**I->gettexte()**), et la date de l'interaction (**I->getdateI()**).

```
insertQuery.prepare("INSERT INTO Interaction (idC,  
contenu, dateInteraction) VALUES (:c, :t, :d)");
```
- Si l'exécution de la requête d'insertion est réussie (**insertQuery.exec()**), un message de débogage est affiché, et une boîte de message (**QMessageBox**) informe l'utilisateur que l'interaction a été ajoutée avec succès.
- Si une erreur se produit lors de l'exécution de la requête SQL pour récupérer l'ID du contact ou lors de l'insertion de l'interaction, des messages de débogage sont affichés. Le dernier texte de la requête SQL peut également être affiché pour aider à diagnostiquer le problème.



Ajout des tâches :

Lorsqu'une tâche est ajoutée à contact, un signal nommé **envoyerTacheEbdd(list<tache*>)** est émis. Ce signal est connecté au slot **recTacheE(list<tache*>)** de la classe **basededonnee**.

Voici une explication détaillée de chaque étape de ce slot :

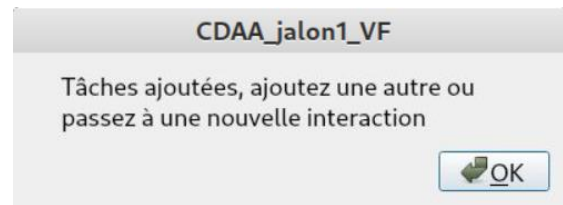
- Une requête SQL est exécutée pour sélectionner le dernier ID d'interaction de la table **Interaction**. La requête est structurée pour récupérer le dernier ID en utilisant **ORDER BY idI DESC LIMIT 1**:

```
if (selectQuery.exec("SELECT idI FROM Interaction ORDER BY idI  
DESC LIMIT 1") && selectQuery.next())
```

- Si la requête est exécutée avec succès (`selectQuery.exec()` et `selectQuery.next()`), l'ID de la dernière interaction est extrait et stocké dans la variable `dernierIdI`.
- Une boucle **for** itère à travers la liste de tâches (`lt`).
- Pour chaque tâche, une nouvelle requête d'insertion SQL préparée (`insertQuery`) est créée pour insérer la tâche dans la table **Todo**.
- Les valeurs nécessaires, telles que l'ID de l'interaction (`dernierIdI`), le texte de la tâche (`T->getevenement()`), et la date de la tâche (`T->getdateT()`), sont liées aux paramètres de la requête:

```
insertQuery.prepare("INSERT INTO Todo (idI, tache,
                        dateTodo) VALUES (:idI, :t, :d)");
```

- Si l'insertion réussit (`insertQuery.exec()`), un message de débogage est affiché indiquant que l'insertion a réussi.



- En cas d'échec, un message de débogage affiche l'erreur survenue lors de l'ajout de l'enregistrement à la table **Todo**.
- Une boîte de message (`QMessageBox`) est utilisée pour informer l'utilisateur que les tâches ont été ajoutées avec succès. Ce message est affiché après la boucle, une fois que toutes les tâches ont été insérées.

Déterminer le nombre d'entrées dans la table "Contact" de la base de données :

Lorsque le signal **cherchNBC()** de la mainwindow est émis, le slot **trouverNb()** de la base de donnée est effectué :

- Une variable entière **nb** est déclarée pour stocker le nombre d'entrées dans la table "Contact".
- Une requête SQL est préparée pour compter le nombre d'entrées dans la table "Contact". La requête utilise la fonction **COUNT(*)**, qui compte le nombre total de lignes dans la table.

```
query.prepare("SELECT COUNT (*) from Contact");
```

- Si la requête retourne des résultats, le nombre d'entrées est extrait à partir de la première colonne du résultat (`query.value(0).toInt()`) et est stocké dans la variable **nb**.
- Un signal nommé **envoyerNBC** est émis avec le nombre d'entrées (**nb**) comme argument. Ceci permet l'affichage du nombre de contacts de la base de données dans la mainwindow.

Suppression d'un contact :

La slot **deleteCbd** est e conçu pour supprimer un contact de la base de données, ainsi que toutes les tâches et interactions associées à ce contact. Voici une explication détaillée de ce qui se passe dans cette fonction :

- La méthode **QSqlDatabase::database().transaction()** est appelée pour commencer une transaction. C'est-à-dire que toutes les opérations sont exécutées avec succès ou aucune d'entre elles n'est exécutée.
- Une requête est exécutée pour sélectionner l'ID du contact en fonction du nom et du prénom du contact à supprimer.

```
selectQuery.prepare("SELECT idC FROM Contact WHERE nom = :nomW1 AND  
                    prenom = :prenomW1");
```

- Si un contact correspondant est trouvé, une requête est préparée pour supprimer ce contact de la table "Contact" en utilisant les valeurs du nom et du prénom.

```
query.prepare("DELETE FROM Contact WHERE nom = :nomW1 AND prenom  
              = :prenomW1");
```

- Une autre requête est préparée pour supprimer les tâches associées à l'IDI (ID d'interaction) du contact dans la table "Todo".

```
query.prepare("DELETE FROM Todo WHERE idI IN (SELECT idI FROM  
        Interaction WHERE idC = :idC)");
```

- Une troisième requête est préparée pour supprimer les interactions du contact dans la table "Interaction".

```
query.prepare("DELETE FROM Interaction WHERE idC = :idC");
```

- Si toutes les opérations de suppression sont effectuées avec succès, la transaction est validée (**commit**), ce qui signifie que toutes les modifications sont enregistrées de manière permanente dans la base de données.

Export json :

- On a utilisé la méthode **exporterJson** qui prend un chemin de fichier JSON en paramètre et utilise des objets **QJsonObject** et **QJsonArray** pour structurer les données de la base de données.
- Ensuite, nous utilisons la classe **QJsonDocument** pour créer un document JSON contenant ces données, puis écrivons ce document dans un fichier spécifié.
- Cette méthode est appelée dans le constructeur de la classe avec un chemin de fichier prédéfini. Cela signifie qu'à chaque création d'une instance de cette classe, les données de la base de données sont exportées au format **JSON** dans le fichier spécifié.

Mise à jour Json :

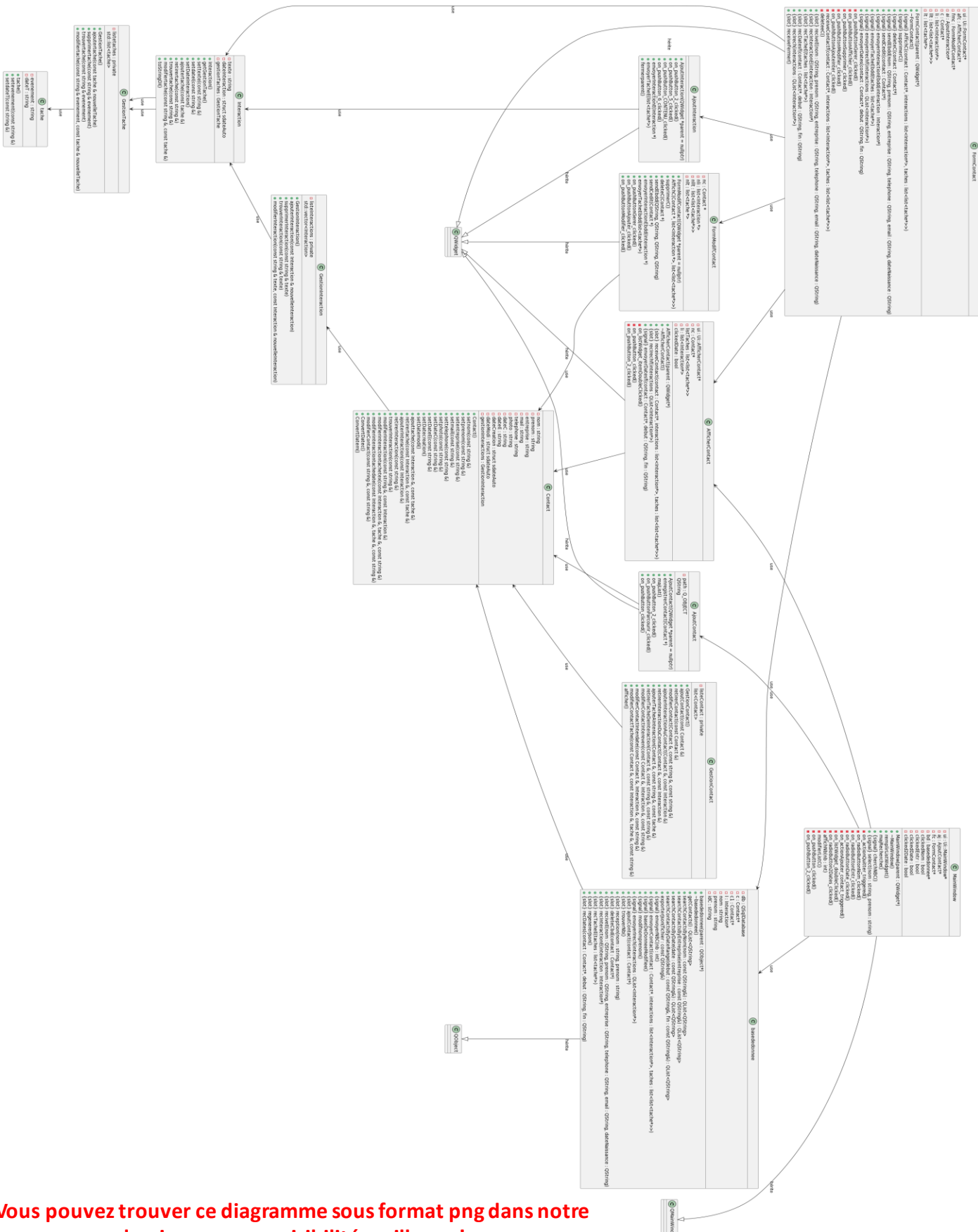
- On établit une connexion entre le signal **baseDeDonneeModifiee** et le slot **regenererJson()**. Donc , à chaque fois que le signal **baseDeDonneeModifiee** est émis, la fonction **regenererJson()** sera appelée.
- La fonction **regenererJson()** elle-même appelle la méthode **exporterJson** avec un chemin de fichier prédéfini .
- Ainsi, la méthode **regenererJson** est généralement appelée lorsque le signal est émis à partir de fonctions de modifications suivantes : **ajoutContact, reciveE, recInteractionE, recTacheE, deleteCbd** pour mettre à jour le fichier JSON exporté chaque fois que des modifications sont apportées à la base de données.
- Cela garantit que le fichier exporté reflète toujours les dernières données de la base de données sans avoir à réouvrir à chaque l'application .

```
{
  "Contact": [
    {
      "dateC": "8/12/2023",
      "dateM": "18/12/2023",
      "email": "img@gmail.com",
      "entreprise": "IKEA",
      "idC": 19,
      "nom": "choukri",
      "photo": "/home1/depositphotos_671048078-stock-illustration-bussines-icon-people-black-color.webp",
      "prenom": "imane",
      "telephone": "0705589665"
    },
  ],
  "Interaction": [
    {
      "contenu": "INTERA1",
      "dateInteraction": "8/12/2023",
      "idC": 19,
      "idI": 10
    },
    {
      "contenu": "INTERACTION2",
      "dateInteraction": "11/12/2023",
      "idC": 19,
      "idI": 42
    },
  ],
  "Todo": [
    {
      "dateTodo": "01/01/2024",
      "idI": 42,
      "idT": 36,
      "tache": "TACHE1"
    },
    {
      "dateTodo": "11/12/2023",
      "idI": 42,
      "idT": 37,
      "tache": "TACHE2"
    },
    {
      "dateTodo": "11/12/2024",
      "idI": 42,
      "idT": 38,
      "tache": "TACHE3"
    },
  ],
}
```

Doxygen et Doxyfile

Nous avons créé un fichier Doxyfile pour générer une documentation à partir de notre code source. En utilisant Doxygen, nous avons personnalisé ce fichier en spécifiant les fichiers source, les répertoires, et d'autres paramètres pertinents. Dans notre code, nous avons ajouté des commentaires spéciaux Doxygen pour décrire nos classes, fonctions, et variables. En exécutant Doxygen avec le Doxyfile, nous avons généré une page web HTML contenant une documentation détaillée de notre projet, organisée par sections telles que les classes et les fonctions. Cette documentation facilite la compréhension du code et peut être mise à jour automatiquement chaque fois que des modifications sont apportées au code source. En résumé, l'utilisation de Doxygen avec notre Doxyfile simplifie le processus de documentation, favorise la collaboration au sein de l'équipe, et garantit une documentation à jour de notre projet.

Diagramme de classes



Vous pouvez trouver ce diagramme sous format png dans notre dossier pour une visibilité meilleure !