RAPPORT PROJET

CONCEPTION ET DEVELOPPEMENT D'APPLICATION AVANCÉE 2023-2024

Membres du groupe n°C:

CORNEVIN CAMILLE

STRAINCHAMPS CLOTHILDE

I. INTRODUCTION

Ce projet de création d'une application de Contact que l'on peut tous trouver dans nos téléphones a pour objectif de nous familiariser à la conception et au développement applications en faisant usage des technologies C++ et QT.

Ce document fait l'objet d'un compte rendu sur l'ensemble du projet, la partie C++, l'interface graphique et la base de données.

II. MODELISATION DE L'APPLICATION

A. Modélisation de la partie en C++

Pour pouvoir définir les classes nécessaires pour répondre aux attentes de notre application nous avons d'abord identifier les objets principaux suivant :

- Un contact
- Une interaction
- Un todo

De plus, on suppose qu'une interaction est composée de plusieurs Todo et qu'un Contact est composé de plusieurs interactions et que l'application est elle-même composée de plusieurs contacts.

Pour pouvoir gérer le fait d'avoir plusieurs éléments nous choisissons la structure de données vector de c++, car elle est facile à mettre en œuvre et à utiliser. Maintenant pour pouvoir appliquer des traitements sur ces listes nous avons choisi de créer des classes pour gérer ces listes, par exemple pour ajouter, supprimer ou rechercher un élément.

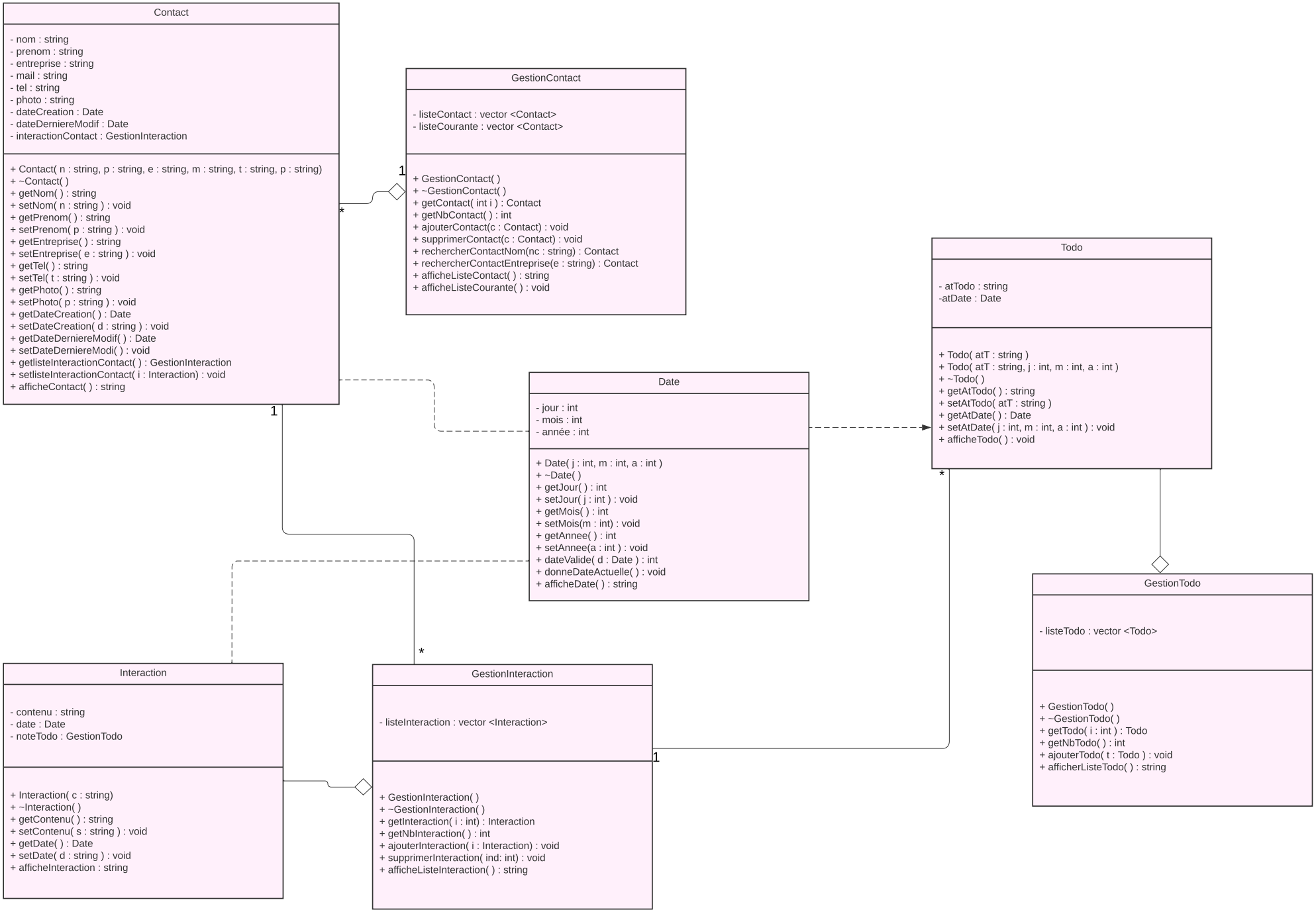
Nous avons donc les objets suivant en plus :

- GestionContact
- GestionInteraction
- GestionTodo

Ensuite pour l'entité des dates nous avons trouvé plus optimisé de créer notre propre objet de type Date qui ne prend en compte que les jours, les mois et les années.

Les objets de type Contact, Interaction et Todo utiliseront un objet de type Date pour leurs attributs de date.

Pour représenter la situation décrite ci-dessus nous avons le diagramme de classes suivant :

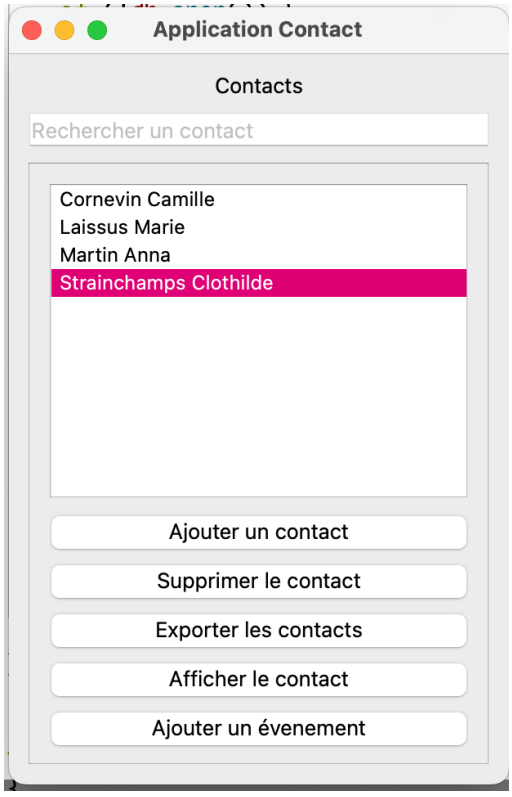

Contact
- nom : string
- prenom : string
- entreprise : string
- mail : string
- tel : string
- photo : string
- dateCreation : Date
- dateDerniereModif : Date
- interactionContact : GestionInteraction
+ Contact(n : string, p : string, e : string, m : string, t : string, p : string)
+ ~Contact( )
+ getNom( ) : string
+ setNom(n : string) : void
+ getPrenom( ) : string
+ setPrenom(p : string) : void
+ getEntreprise( ) : string
+ setEntreprise(e : string) : void
+ getTel( ) : string
+ setTel(t : string) : void
+ getPhoto( ) : string
+ setPhoto(p : string) : void
+ getDateCreation( ) : Date
+ setDateCreation(d : string) : void
+ getDateDerniereModif( ) : Date
+ setDateDerniereModi( ) : void
+ getlisteInteractionContact( ) : GestionInteraction
+ setlisteInteractionContact(i : Interaction) : void
+ afficheContact( ) : string
GestionContact
- listeContact : vector <Contact>
- listeCourante : vector <Contact>
+ GestionContact( )
+ ~GestionContact( )
+ getContact(int i) : Contact
+ getNbContact( ) : int
+ ajouterContact(c : Contact) : void
+ supprimerContact(c : Contact) : void
+ rechercherContactNom(nc : string) : Contact
+ rechercherContactEntreprise(e : string) : Contact
+ afficheListeContact( ) : string
+ afficheListeCourante( ) : void
Date
- jour : int
- mois : int
- année : int
+ Date(j : int, m : int, a : int)
+ ~Date( )
+ getJour( ) : int
+ setJour(j : int) : void
+ getMois( ) : int
+ setMois(m : int) : void
+ getAnnee( ) : int
+ setAnnee(a : int) : void
+ dateValide(d : Date) : int
+ donneDateActuelle( ) : void
+ afficheDate( ) : string
Todo
- atTodo : string
-atDate : Date
+ Todo(atT : string)
+ Todo(atT : string, j : int, m : int, a : int)
+ ~Todo( )
+ getAtTodo( ) : string
+ setAtTodo(atT : string)
+ getAtDate( ) : Date
+ setAtDate(j : int, m : int, a : int) : void
+ afficheTodo( ) : void
GestionTodo
- listeTodo : vector <Todo>
+ GestionTodo( )
+ ~GestionTodo( )
+ getTodo(i : int) : Todo
+ getNbTodo( ) : int
+ ajouterTodo(t : Todo) : void
+ afficherListeTodo( ) : string
Interaction
- contenu : string
- date : Date
- noteTodo : GestionTodo
+ Interaction(c : string)
+ ~Interaction( )
+ getContenu( ) : string
+ setContenu(s : string) : void
+ getDate( ) : Date
+ setDate(d : string) : void
+ afficheInteraction : string
GestionInteraction
- listeInteraction : vector <Interaction>
+ GestionInteraction( )
+ ~GestionInteraction( )
+ getInteraction(i : int) : Interaction
+ getNbInteraction( ) : int
+ ajouterInteraction(i : Interaction) : void
+ supprimerInteraction(ind: int) : void
+ afficheListeInteraction( ) : string
1
*
1
*
1
*
*
1

B. Modélisation de l'interface graphique

Pour pouvoir implémenter l'interface graphique nous avons utilisé la librairie Qt.
Notre application est composée de quatre fenêtres.
La fenêtre principale de l'application, qui reste en cours d'exécution pendant toute la durée de vie de notre application. Une fenêtre de visualisation d'un contact sélectionné dans la fenêtre principale. Nous avons pour que l'utilisateur puisse saisir les données d'un nouveau contact, une fenêtre pour ajouter un contact, et de même pour saisir un nouvel évenement à ajouter à un contact.

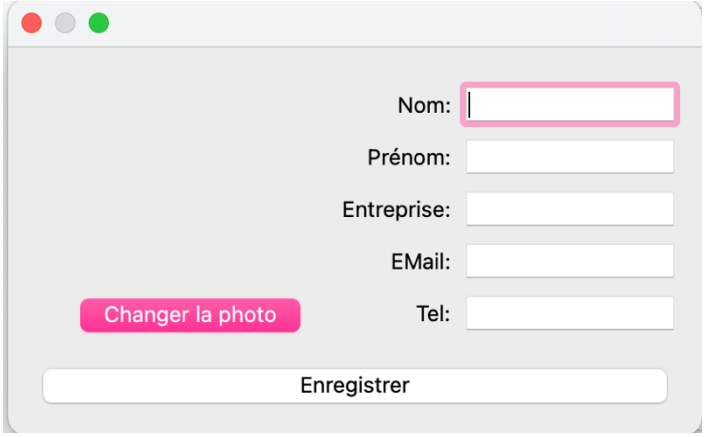
Pour la partie visuelle de notre application voici des images des différentes fenêtres :

A. La fenêtre principale.



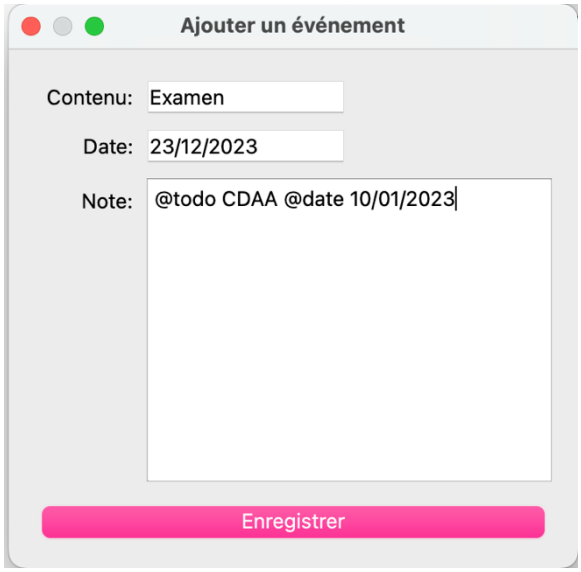
Pour ajouter un contact nous n'avons pas besoin de sélectionner un contact, alors que pour les autres actions cela est nécessaire.

B. La fenêtre d'ajout d'un contact.



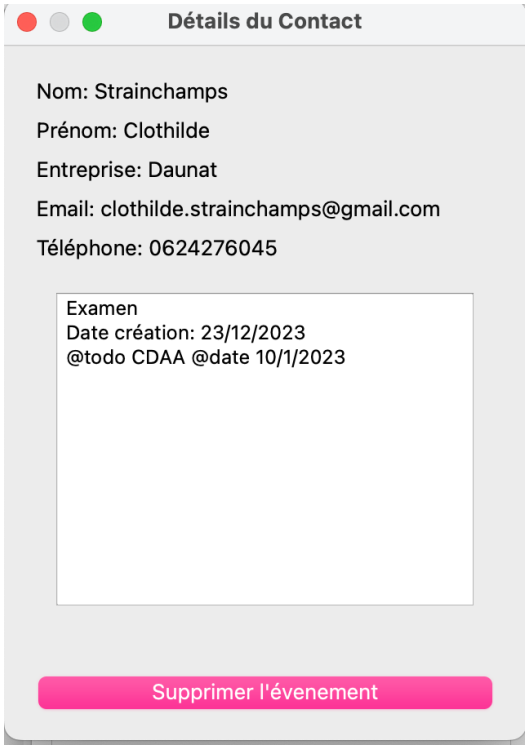
On précise que seule les lettres sont acceptées pour le nom, prenom, entreprise, pour l'email un email classique, et le numéro de téléphone ne peut être que composé de chiffres.

C. La fenêtre d'ajout d'un évenement.

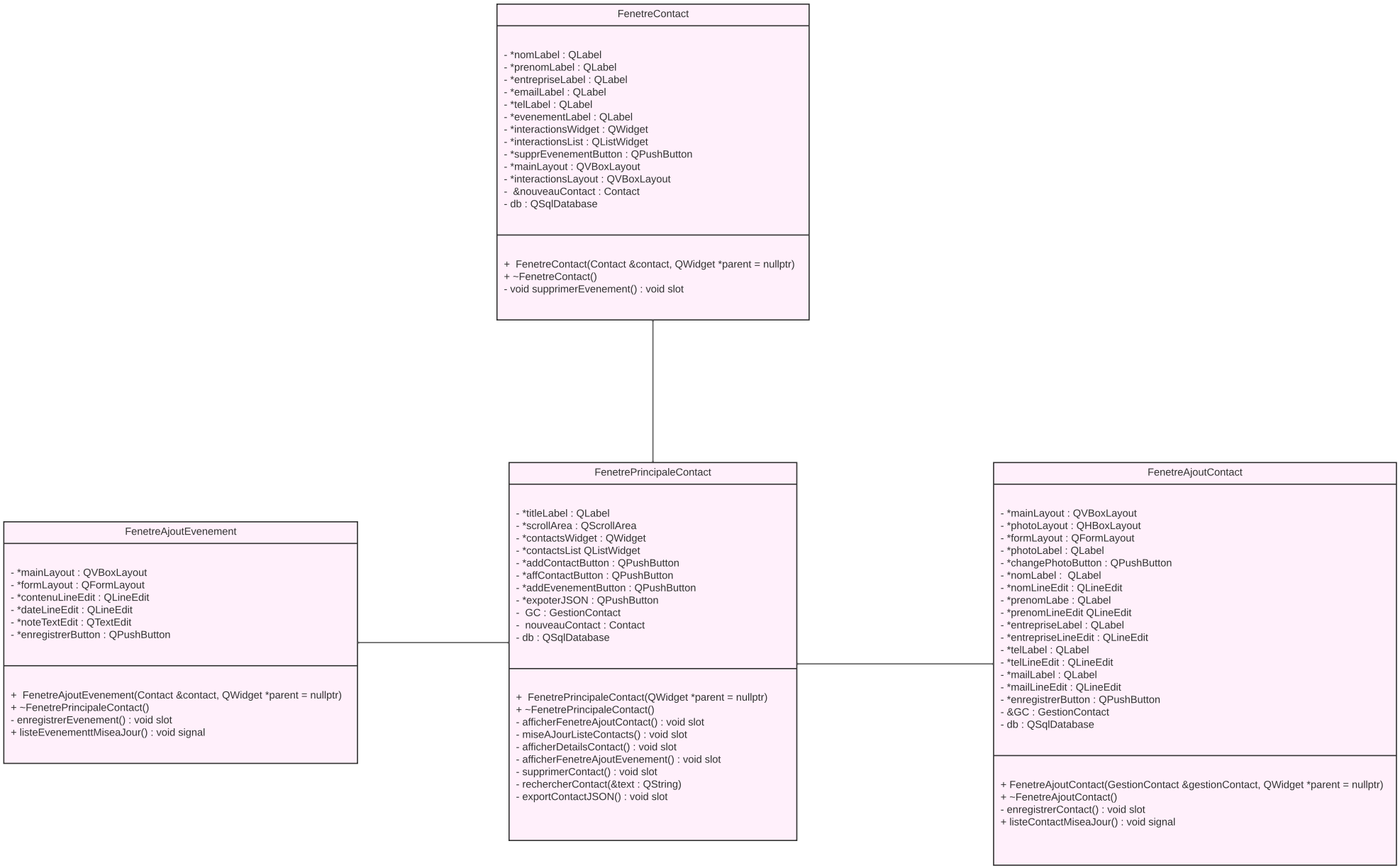


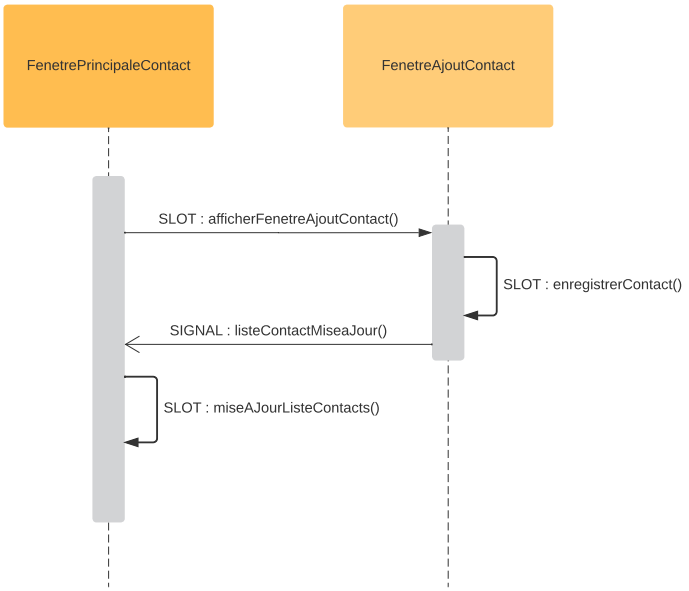
On précise que les dates et note doivent être écrit comme dans l'exemple ci-dessus.

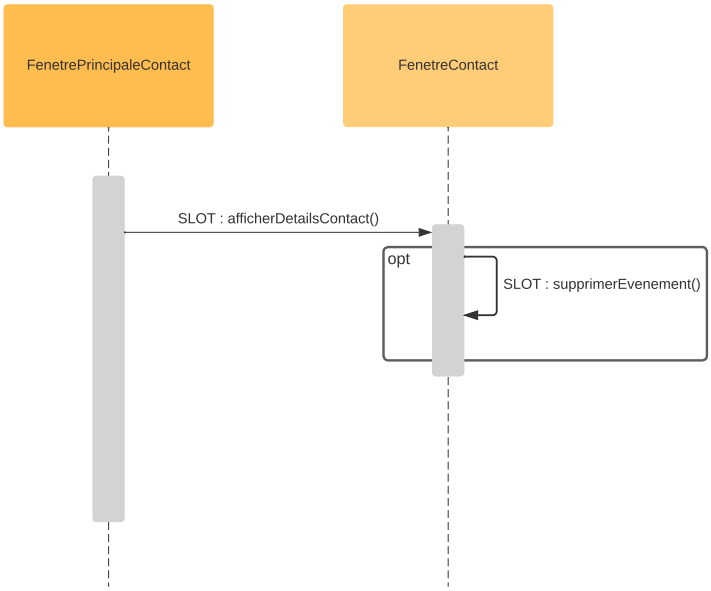
D. La fenêtre de visualisation d'un contact.

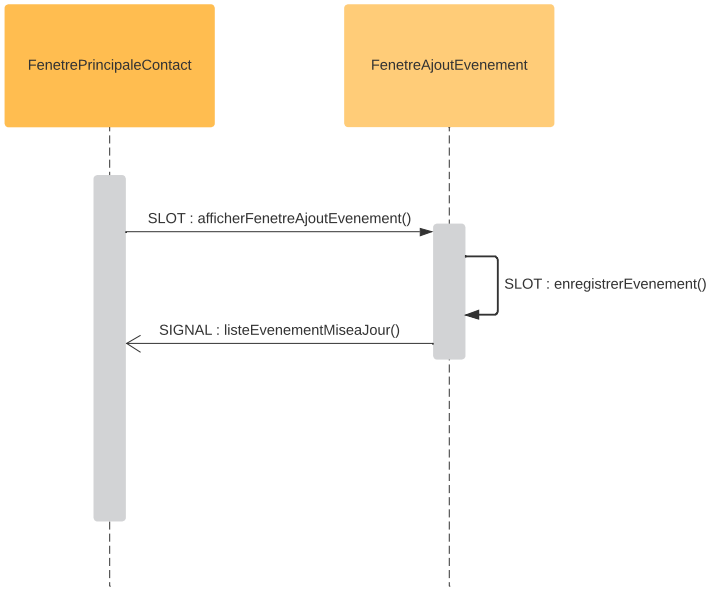


Pour représenter la situation décrite ci-dessus nous avons le diagramme de classes et de séquence suivant :


FenetreContact
- *nomLabel : QLabel
- *prenomLabel : QLabel
- *entrepriseLabel : QLabel
- *emailLabel : QLabel
- *telLabel : QLabel
- *evenementLabel : QLabel
- *interactionsWidget : QWidget
- *interactionsList : QListWidget
- *supprEvenementButton : QPushButton
- *mainLayout : QVBoxLayout
- *interactionsLayout : QVBoxLayout
- &nouveauContact : Contact
- db : QSqlDatabase
+ FenetreContact(Contact &contact, QWidget *parent = nullptr)
+ ~FenetreContact()
- void supprimerEvenement() : void slot
FenetrePrincipaleContact
- *titleLabel : QLabel
- *scrollArea : QScrollArea
- *contactsWidget : QWidget
- *contactsList QListWidget
- *addContactButton : QPushButton
- *affContactButton : QPushButton
- *addEvenementButton : QPushButton
- *expoterJSON : QPushButton
- GC : GestionContact
- nouveauContact : Contact
- db : QSqlDatabase
+ FenetrePrincipaleContact(QWidget *parent = nullptr)
+ ~FenetrePrincipaleContact()
- afficherFenetreAjoutContact() : void slot
- miseAJourListeContacts() : void slot
- afficherDetailsContact() : void slot
- afficherFenetreAjoutEvenement() : void slot
- supprimerContact() : void slot
- rechercherContact(&text : QString)
- exportContactJSON() : void slot
FenetreAjoutEvenement
- *mainLayout : QVBoxLayout
- *formLayout : QFormLayout
- *contenuLineEdit : QLineEdit
- *dateLineEdit : QLineEdit
- *noteTextEdit : QTextEdit
- *enregistrerButton : QPushButton
+ FenetreAjoutEvenement(Contact &contact, QWidget *parent = nullptr)
+ ~FenetrePrincipaleContact()
- enregistrerEvenement() : void slot
+ listeEvenementtMiseaJour() : void signal
FenetreAjoutContact
- *mainLayout : QVBoxLayout
- *photoLayout : QHBoxLayout
- *formLayout : QFormLayout
- *photoLabel : QLabel
- *changePhotoButton : QPushButton
- *nomLabel : QLabel
- *nomLineEdit : QLineEdit
- *prenomLabe : QLabel
- *prenomLineEdit QLineEdit
- *entrepriseLabel : QLabel
- *entrepriseLineEdit : QLineEdit
- *telLabel : QLabel
- *telLineEdit : QLineEdit
- *mailLabel : QLabel
- *mailLineEdit : QLineEdit
- *enregistrerButton : QPushButton
- &GC : GestionContact
- db : QSqlDatabase
+ FenetreAjoutContact(GestionContact &gestionContact, QWidget *parent = nullptr)
+ ~FenetreAjoutContact()
- enregistrerContact() : void slot
+ listeContactMiseaJour() : void signal


FenetrePrincipaleContact
FenetreAjoutContact
SLOT : afficherFenetreAjoutContact()
SLOT : enregistrerContact()
SIGNAL : listeContactMiseaJour()
SLOT : miseAJourListeContacts()


FenetrePrincipaleContact
FenetreContact
SLOT : afficherDetailsContact()
opt
SLOT : supprimerEvenement()


FenetrePrincipaleContact
FenetreAjoutEvenement
SLOT : afficherFenetreAjoutEvenement()
SLOT : enregistrerEvenement()
SIGNAL : listeEvenementMiseaJour()

C. Modélisation de la base de données.

Pour la base de données, nous avons trois types principaux de données qui sont : les contacts, les interactions et les todos, de plus les interactions et todos sont des attributs multivalués. Nous allons donc créer trois tables : Contact, Interaction et Todo et utiliserons des clés étrangères pour par exemple savoir quels todos sont associés à une interaction et quelles interactions sont associées à un contact.

Pour définir notre base de données nous avons dons les trois relations suivantes :

CONTACT(idcontact, nom, prenom, entreprise, mail, tel, uriphoto)

INTERACTION(idinteraction, idcontact, contenu, date)

TODO(idtodo, idinteraction, idcontact, attodo, atdate)

III. CONCEPTION DE L'APPLICATION

Pour voir quels attributs ont été défini pour chaque classe, on peut se référer au code commenté, mais surtout à la documentation Doxygène qui lui est liée.

A. Conception de la partie en C++

Pour la partie C++ uniquement, les principales structures de données utilisées dans ce projet sont les chaînes de caractères, les listes de type vector présentes dans c++ et les entiers. Nous avons choisi des structures de données simple, car dans une prochaine étape, les données seront stockées dans une base de données et cela facilitera leur insertion et traitements.

B. Conception de la partie QT

Pour la partie QT, la fenêtre principale hérite de QmainWindow car cela défini une fenêtre principale, les autres fenêtres héritent de Qdialog car ce sont des fenêtres en cours d'exécution pendant que notre fenêtre principale est toujours en cours d'exécution.

C. Conception de la base de données

Pour la base de données, la stratégie est simple.

Pour recharger notre liste de contact on sélectionne toutes nos lignes de la table Contact, on extrait chacune des donnés et on reconstruit un objet contact que l'on ajoute à notre objet GestionContact. Ceci est réalisé lors du lancement de notre application avec la fenêtre principale.

Pour recharger la liste d'interactions et de todos d'un contact cela se fait quand on affiche la fenêtre d'affichage des informations d'un contact.

A chaque enregistrement il y a un insert dans la base de données. De même qu'à chaque suppression il y a un delete.

IV. Conclusion

Pour conclure, notre application est fonctionnelle dans l'ensemble, mais certaines fonctionnalités peuvent être améliorées.