

PROJET CDAA

-

STIZ Romain
POUZIN Pierre-Emanuel

SOMMAIRE:

I)Présentation générale

II)Diagramme de classe

III)Base de donnée

I)Présentation Général:

Nous avons conçu une application permettant d'avoir une Gestion de Contacts. Cette application permet entre autres d'ajouter des contacts avec leurs informations, de leurs ajouter des interactions, des todos,...

Chaque Contact que l'on ajoute, modifie, supprime, est stocké dans une base de données et à chaque redémarrage de l'application celle-ci les récupère automatiquement.

Voici comment se déroule l'application:

The screenshot shows a software window titled 'MainWindow' with standard Windows window controls (minimize, maximize, close). The interface is divided into several sections:

- Form Section (Top Left):** Contains five input fields with labels: 'Nom:', 'Prénom:', 'Entreprise:', 'Mail:', and 'Numéro:'. Below these fields are three buttons: 'Modifier', 'Valider', and 'Supprimer'.
- Summary Section (Top Right):** Displays 'Nombre de Contacts:' followed by a large empty rectangular box.
- History Section (Middle Right):** Labeled 'Historique', it contains a list of contact entries:
 - Romain Stiz Google mail.ru 123456789
 - Jean Dupont Apple gmail 5362722882
 - Pierre Dupond Cora orange.fr 7654345667
 - Romain Stiz Google mail.ru 12345678
 - Pierre-emmanuel Pouzin Non mail 765456789876
- Buttons Section (Bottom):** Includes a 'Créer Interaction' button on the left, and 'Ajouter' and 'Supprimer' buttons on the right.
- Export Section (Bottom Left):** Features an 'Export JSON' button.

En haut à gauche, nous avons la partie où pour **Créer** un contact. Chaque contact est créé à l'aide d'un Nom, Prénom, Entreprise, Mail, Téléphone, ainsi qu'une photo lors du clique de la confirmation de la création.

Pour récupérer toutes ces informations, on crée des Line_Edit qui quand on appuie sur Valider appelle une méthode afin de la créer.

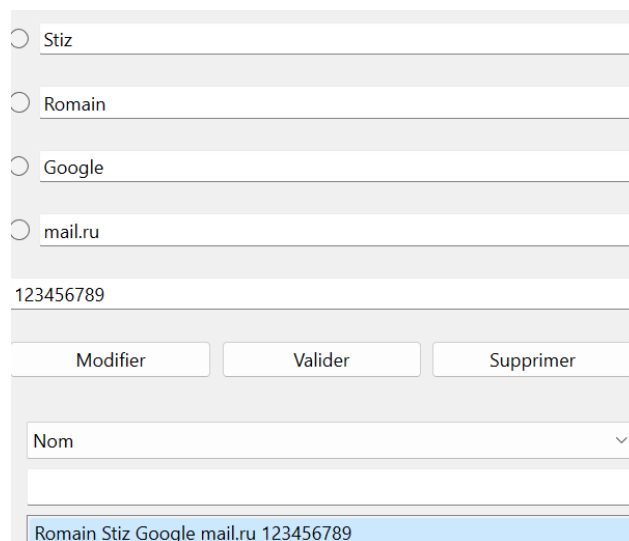
```
connect(ui->Bt_Valider_Contact, SIGNAL(clicked()), this, SLOT(onBt_ValiderContact()));  
-----  
std::string nom = ui->Line_Nom->text().toStdString();  
std::string prenom = ui->Line_Prenom->text().toStdString();  
std::string entreprise = ui->Line_Entreprise->text().toStdString();  
std::string mail = ui->Line_Mail->text().toStdString();  
std::string numero = ui->Line_Numero->text().toStdString();
```

Avec ces informations, on appelle le constructeur de Contact et on ajoute ce Contact à la Gestion de Contacts.

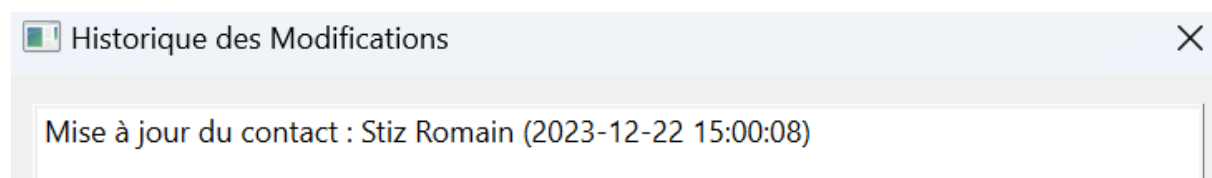
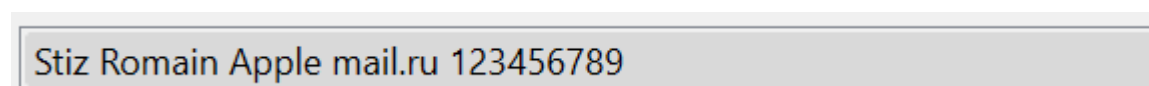
On ajoute ensuite dans la QListWidget le contact créé.

Dans la photo de l'application, on voit qu'à l'ouverture il y a déjà des Contacts créés.

Pour **Modifier** un contact, la méthode est assez semblable, pour cela il faut cliquer sur un Contact de la liste, ainsi les informations s'affichent dans les Line_Edit.



Ensuite on modifie les informations voulues et on appuie sur Modifier. On met ensuite ces résultats dans un historique que l'on retrouve sur le bouton Historique.



Pour **Supprimer** un contact, on procède toujours de la même manière sauf que nous le supprimons de la liste de Gestion Contact.

Les **Interactions** et **Todos** fonctionnent pareils.

En effet, une fois le contact/interaction sélectionné, on ouvre une QDialog afin de rentrer les informations sur le contenu et la date puis nous l'ajoutons à la liste des Interactions du contact.

Dialog

Entrez un Contenu :

Finir Rapport

Choisir une Date :

décembre 2023

	lun.	mar.	mer.	jeu.	ven.	sam.	dim.
48	27	28	29	30	1	2	3
49	4	5	6	7	8	9	10
50	11	12	13	14	15	16	17
51	18	19	20	21	22	23	24
52	25	26	27	28	29	30	31
1	1	2	3	4	5	6	7

Valider Annuler

Pour les Todos, on passe juste par une création d'Association avec l'interaction sélectionnée et le todo créé.

Historique

Finir projet @date 2023-12-22

Supprimer

@todo à rendre avant 22h @date 2023-12-22

Nous avons pensé à mettre en place des Regex dans la création du Contact afin de ne pas rentrer n'importe quoi.

```
QRegularExpression regex_lettre("^([A-Z][a-z\\-\\_])*$");
QRegularExpressionValidator *validator_lettre = new QRegularExpressionValidator(regex_lettre, this);
ui->Line_Nom->setValidator(validator_lettre);
ui->Line_Prenom->setValidator(validator_lettre);

QRegularExpression regex_num("[0-9]+");
QRegularExpressionValidator *validator_num = new QRegularExpressionValidator(regex_num, this);
ui->Line_Numero->setValidator(validator_num);
```

Nous avons d'autres options dans le programme, en effet nous pouvons **Tirer** les contacts de plusieurs façons.

Pour cela, nous avons mis en place des RadioBouttons que nous pouvons voir en haut à gauche.

Une fois cliquer sur l'un d'entre eux, il va effectuer un tri selon le bouton sélectionné.

```

void MainWindow::onRadioButtonClicked() {
    // Déterminez le critère de tri en fonction du bouton radio sélectionné
    std::function<bool(const Contact&, const Contact&> comparator;

    if (ui->radio_Prenom->isChecked()) {
        comparator = [](const Contact& c1, const Contact& c2) {
            return QString::compare(QString::fromStdString(c1.getPrenom()),
                                   QString::fromStdString(c2.getPrenom()),
                                   Qt::CaseInsensitive) < 0;
        };
    } else if (ui->radio_Nom->isChecked()) {
        comparator = [](const Contact& c1, const Contact& c2) {
            return QString::compare(QString::fromStdString(c1.getNom()),
                                   QString::fromStdString(c2.getNom()),
                                   Qt::CaseInsensitive) < 0;
        };
    }
}

```

Voici donc la liste du début triée par Nom.

```

Dupont Jean Apple gmail 5362722882
Dupond Pierre Cora orange.fr 7654345667
Pouzin Pierre-emmanuel Non mail 765456789876
Stiz Romain Google mail.ru 123456789
Stiz Romain Google mail.ru 12345678

```

Nous avons ensuite la possibilité de **Rechercher** dans notre liste de contacts, nous avons une combo box où nous choisissons entre le Nom, Prénom,...

Cela fonctionne comme suivant:

```

// Fonction de recherche par entreprise
void MainWindow::rechercherParEntreprise(const QString &text) {
    ui->Text_Contact->clear();

    for (const Contact &contact : gc.getContactsListe()) {
        if (QString::fromStdString(contact.getEntreprise()).contains(text, Qt::CaseInsensitive)) {
            QString contactString = QString::fromStdString(
                contact.getPrenom() + " " + contact.getNom() + " " + contact.getEntreprise() + " " +
                contact.getMail() + " " +
                contact.getTelephone()
            );
            ui->Text_Contact->addItem(contactString);
        }
    }
}

```

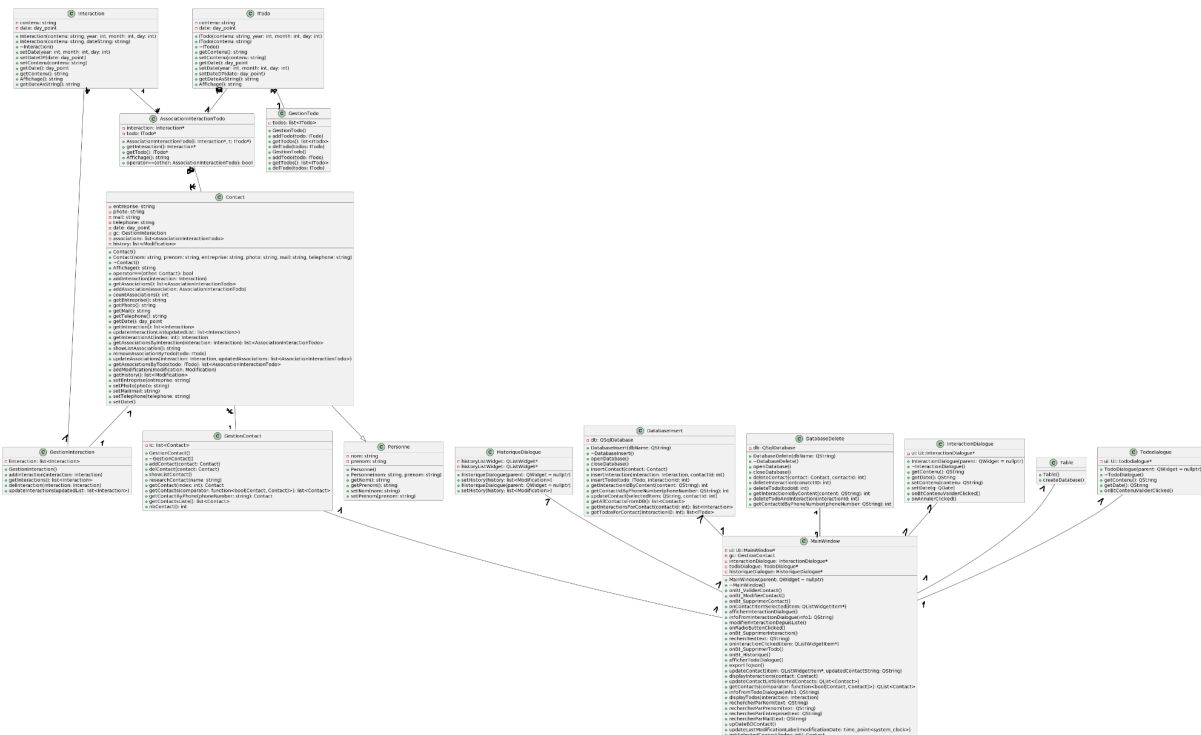
On parcourt la liste et on affiche suivant la valeur rentrée dans la barre.

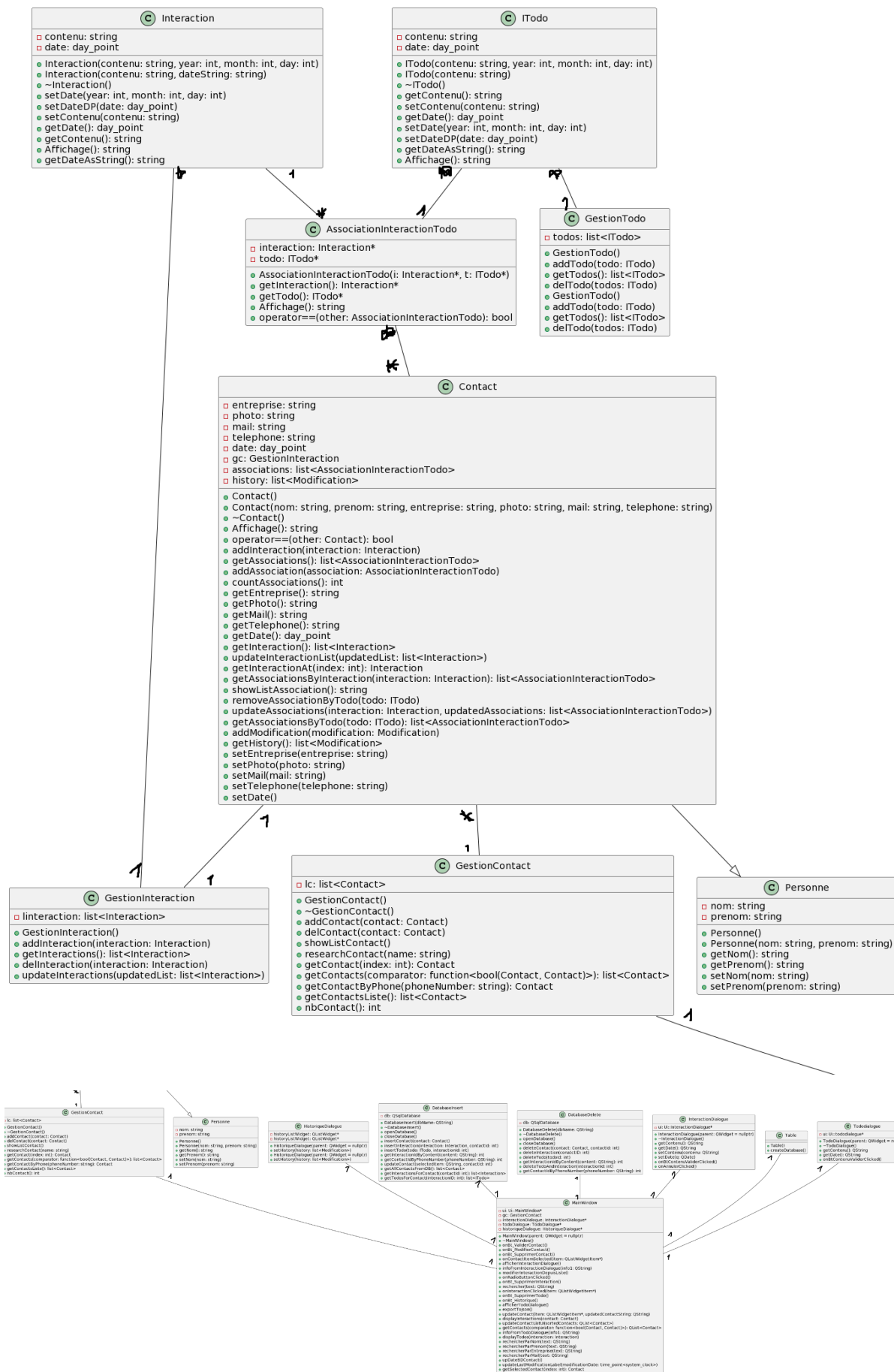
```
lt
  "contacts": [
    {
```

Voici le premier Contact dans le fichier json avec une interaction et 0 todo.

```
{
  "contacts": [
    {
      "entreprise": "Google",
      "interactions": [
        {
          "description": "Finir projet @date 2023-12-22",
          "todos": [
            ]
        }
      ],
      "mail": "mail.ru",
      "nom": "Romain",
      "numero": "123456789",
      "photo": "C:/Users/stizr/Desktop/OIP.jpg",
      "prenom": "Stiz"
    }
  ],
}
```

II) Diagramme de classe:





Le diagramme a été découpé afin de mieux le percevoir.

Explication de ce diagramme:

La classe Contact, au cœur de notre projet, encapsule tous les attributs de la classe Personne, et y ajoute des attributs spécifiques tels que l'entreprise, l'adresse e-mail, le numéro de téléphone, etc. Elle constitue une représentation complète d'une personne au sein de notre application.

Pour enrichir la fonctionnalité de la classe Contact, nous introduisons la classe GestionInteraction. Cette classe permet à un contact de gérer une liste d'interactions. Chaque interaction peut être associée à un ou plusieurs Todos, ce qui permet de créer des liens significatifs entre les actions entreprises et les tâches à accomplir.

La classe GestionAssociation vient compléter le modèle en établissant des liens entre les Interactions et les Todos. Ainsi, un contact peut être associé à diverses actions et tâches.

En résumé, un contact peut maintenant être caractérisé par une liste d'Interactions et de Todos.

Par la suite, la classe GestionContact agit en tant qu'intermédiaire entre la MainWindow et la classe Contact. Tous les contacts sont stockés dans une liste gérée par GestionContact, facilitant leur intégration dans l'application. La MainWindow utilise une instance de GestionContact pour importer toutes les données nécessaires, ce qui inclut les contacts, les dialogues graphiques (qDialog), les opérations sur la base de données, etc.

Ainsi, la MainWindow devient le centre de l'application, orchestrant les interactions entre l'utilisateur et les données. Elle utilise la GestionContact comme interface pour manipuler et afficher les contacts, assurant ainsi le bon fonctionnement de l'application dans son ensemble.

III)Base de donnée:

Voici l'implémentation de notre base de donnée avec les tables suivantes:
Les clés primaires sont en gras et les clés étrangères sont soulignées.

1. Table **contacts :**

- Colonnes :

- **`id`** (INTEGER) : Identifiant unique pour chaque contact.
- **`nom`** (TEXT) : Nom du contact.

- `prenom` (TEXT) : Prénom du contact.
- `entreprise` (TEXT) : Nom de l'entreprise associée au contact.
- `photo` (TEXT) : Chemin ou URL de la photo du contact.
- `mail` (TEXT) : Adresse e-mail du contact.
- `telephone` (TEXT) : Numéro de téléphone du contact.

2. *Table interactions* :

- **Colonnes** :

- `id` (INTEGER) : Identifiant unique pour chaque interaction.
- `contact_id` (INTEGER) : Clé étrangère faisant référence à l'identifiant du contact associé à cette interaction.
- `contenu` (TEXT) : Contenu de l'interaction.
- `date` (DATE) : Date de l'interaction.

3. *Table todos* :

- **Colonnes** :

- `id` (INTEGER) : Identifiant unique pour chaque todo.
- `contenu` (TEXT) : Contenu du todo.
- `date` (DATE) : Date du todo.

4. *Table interaction_todo* :

- **Colonnes** :

- `interaction_id` (INTEGER) : Clé étrangère faisant référence à l'identifiant de l'interaction.
- `todo_id` (INTEGER) : Clé étrangère faisant référence à l'identifiant du todo.

Les relations entre les tables sont établies à l'aide de clés étrangères, la table "interactions" est liée à la table "contacts" via la clé étrangère `contact_id`. De même, la table "interaction_todo" gère la relation entre les interactions et les todos à l'aide des clés étrangères `interaction_id` et `todo_id`.

En outre, cette structure de base de données permet de stocker des informations sur les contacts, leurs interactions, les todos associés, et les relations entre les interactions et les todos.

La structure en tables facilite l'interrogation de la base de données à l'aide de requêtes SQL. Les relations entre les tables permettent d'extraire des informations spécifiques et d'effectuer des opérations complexes.

En adoptant cette structure, l'ajout de nouvelles fonctionnalités à l'application devient plus facile.

Les modifications ou les mises à jour de la structure peuvent être effectuées de manière plus cohérente et sans causer d'effets indésirables sur d'autres parties de la base de données.

