

Projet CDAA

Pour plus de compréhension et montrer l'avancement du projet depuis l'échelon 1, le compte-rendu sera en 2 parties, le texte du compte rendu de l'échelon 1, et l'explication de tout ce qui a été rajouté à la suite de chaque classes et parties, pour voir les ajouts (le texte des ajouts sera en verts pour plus de compréhension et des liens seront ajouté à chaque parties pour passer l'ancien texte s'il n'est pas nécessaire)

1. Présentation de la base de données :

2. Présentation de l'interface :

3. Présentation des classes :

Présentation de la base de données :

La base de donnée est un ajout important du projet car elle permet de garder en sauvegarde tout ce qui a été ajouté et de pouvoir récupérer les différents contacts, interactions, etc ... lors d'un nouveau lancement du code, elle est constituée de 4 tables : Contacts qui va stocker les différentes informations d'un contact, Interactions qui contiendra l'information de l'interaction, sa date ainsi qu'une clé étrangère qui sera l'id du contact (pour faire le lien entre un contact et ses interactions), une table Todo qui comme interactions contiendra les informations du todo et une clé étrangère qui sera l'id de l'interaction et une table Historique qui ne sert pas vraiment comme d'une vraie table de donnée mais plus comme un fichier de stockage pour garder l'historique des changements fait lors des dernière utilisation du code. Les tables sont créées dans le main, le code vérifie d'abord qu'il existe bien un fichier BDD.sqlite dans le projet pour l'ouvrir, si ce n'est pas le cas il en crée un nouveau, puis vérifie s'il peut bien l'ouvrir

```
int main(int argc, char *argv[])
{
    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("BDD.sqlite");

    // Vérifier si le fichier de base de données existe
    if (!QFile::exists("BDD.sqlite")) {
        // S'il n'existe pas, créer le fichier
        QFile::copy(":/emptydb", "BDD.sqlite");
    }

    if (!db.open()) {
        qDebug() << "Erreur lors de l'ouverture de la base de données :" << db.lastError().text();
        return 1;
    }
}
```

Il va ensuite faire la création des différentes tables seulement si elle n'existe pas déjà pour ne pas les remplacer.

Création de la table contact :

```
QSqlQuery query;
// Requête SQL pour créer la table "contact" si elle n'existe pas déjà
QString createTableSQL = "CREATE TABLE IF NOT EXISTS contact ("
    "id INTEGER PRIMARY KEY AUTOINCREMENT,"
    "nom TEXT NOT NULL,"
    "prenom TEXT NOT NULL,"
    "entreprise TEXT,"
    "mail TEXT,"
    "telephone TEXT,"
    "urlphoto TEXT,"
    "date_jour INTEGER,"
    "date_mois INTEGER,"
    "date_annee INTEGER,"
    "date_heure INTEGER,"
    "date_min INTEGER"
    ")";

if (!query.exec(createTableSQL)) {
    qDebug() << "Erreur lors de la création de la table :" << query.lastError().text();
} else {
    qDebug() << "Table 'contact' créée avec succès.";
}
```

Création de la table interaction :

```
// Requête SQL pour créer la table "interaction" si elle n'existe pas déjà
QString createInteractionTableSQL = "CREATE TABLE IF NOT EXISTS interaction ("
    "id INTEGER PRIMARY KEY AUTOINCREMENT,"
    "contenu TEXT NOT NULL,"
    "date_jour INTEGER,"
    "date_mois INTEGER,"
    "date_annee INTEGER,"
    "contact_id INTEGER,"
    "FOREIGN KEY(contact_id) REFERENCES contact(id)"
    ");";

if (!query.exec(createInteractionTableSQL)) {
    qDebug() << "Error lors de la création de la table interaction : " << query.lastError().text();
    return 1;
} else {
    qDebug() << "Table 'interaction' créée avec succès.";
}
```

Création de la table todo :

```
// Requête SQL pour créer la table "todo" si elle n'existe pas déjà
QString createTodoTableSQL = "CREATE TABLE IF NOT EXISTS todo ("
    "id INTEGER PRIMARY KEY AUTOINCREMENT,"
    "contenu TEXT NOT NULL,"
    "date_jour INTEGER,"
    "date_mois INTEGER,"
    "date_annee INTEGER,"
    "interaction_id INTEGER,"
    "FOREIGN KEY(interaction_id) REFERENCES interaction(id)"
    ");";

if (!query.exec(createTodoTableSQL)) {
    qDebug() << "Error lors de la création de la table todo:" << query.lastError().text();
    return 1;
} else {
    qDebug() << "Table 'todo' créée avec succès.";
}
```

Création de la table histo :

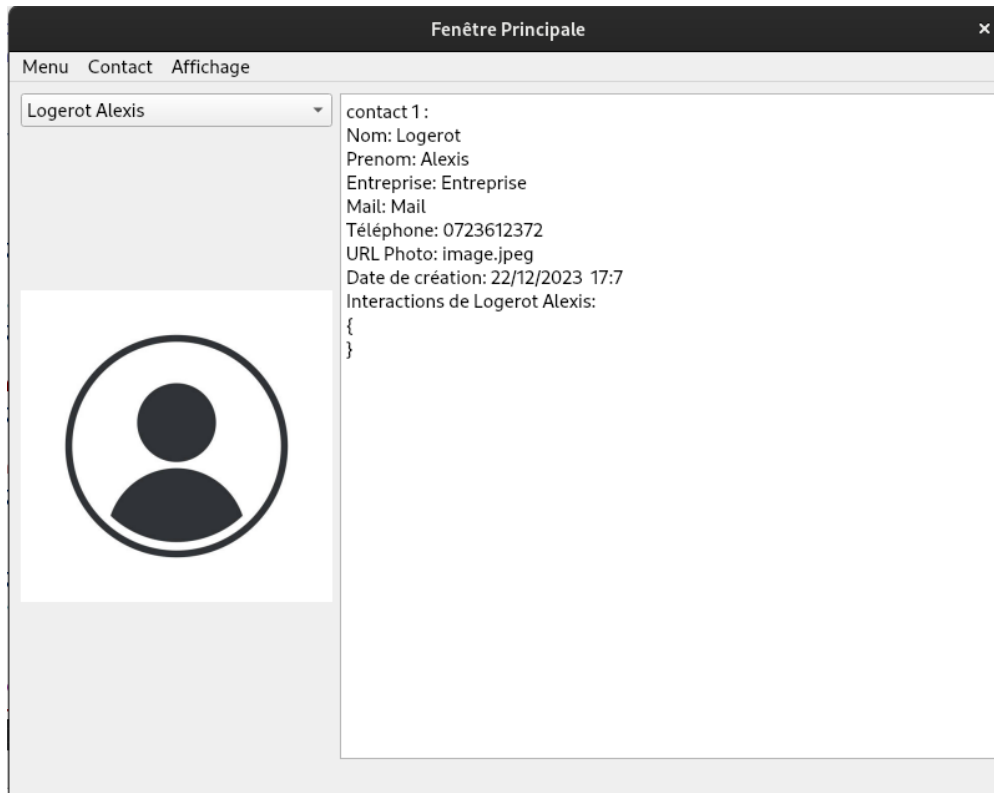
```
QString TableHistorique = "CREATE TABLE IF NOT EXISTS histo("
    "id INTEGER PRIMARY KEY AUTOINCREMENT,"
    "modif TEXT NOT NULL,"
    "date_jour INTEGER,"
    "date_mois INTEGER,"
    "date_annee INTEGER"
    ");";

if (!query.exec(TableHistorique)) {
    qDebug() << "Error lors de la création de la table histo:" << query.lastError().text();
    return 1;
} else {
    qDebug() << "Table 'histo' créée avec succès.";
}
```

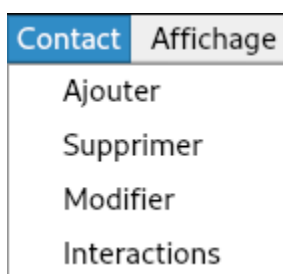
Une fois que les table sont créées, le main va ouvrir la MainWindow pour afficher l'interface mais aussi remplir la liste de contact et les autres informations avec la fonction **recupererContactsDepuisSQL()** du fichier sql.cpp et qui parcourt les différentes tables pour recréer les différents objets. Les différents requête se font ensuite lors de l'ajout, la suppression, etc ... lorsque la mainwindow reçoit le signal des différents widgets.

Présentation de l'interface :

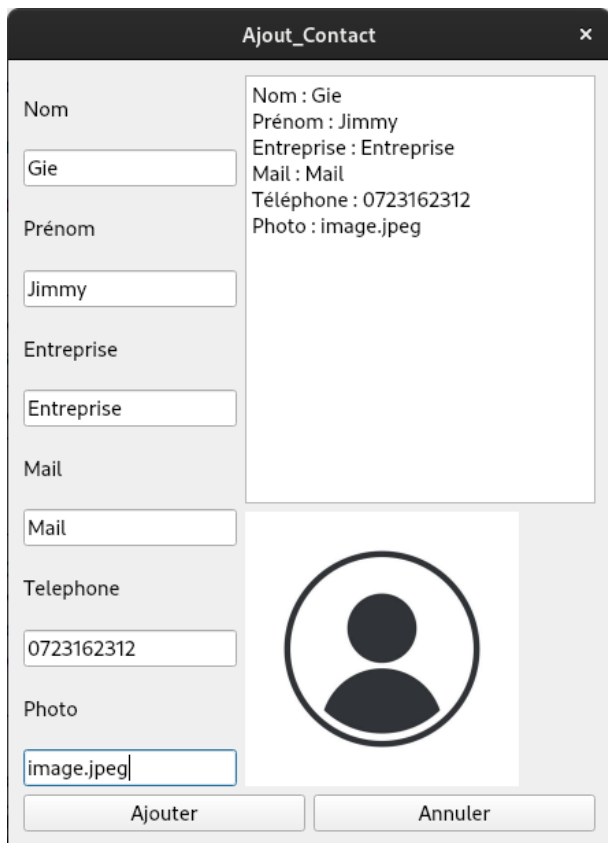
L'interface est composée de 5 fenêtres différentes, la fenêtre Principale MainWindow qui est le moteur principale du projet, elle permet l'affichage de la liste des contacts ou d'un contact précis, l'affichage de l'historique des changement le lancement des différents tri, et l'ouverture des autres Widgets permettant l'ajouts la modification et la suppressions des contacts.



Pour ajouter un contact il suffit d'aller dans le menu contact et d'ouvrir le Widget d'ajout :



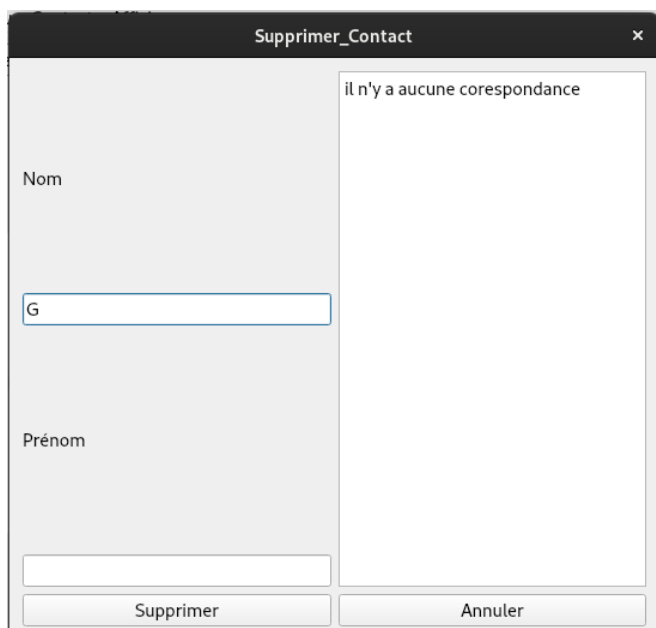
Lorsque la fenêtre s'ouvre il suffit de remplir chaque ligne par les informations du contact (le nom et prénom doivent commencer par une majuscules et ne peuvent pas avoir de chiffre, le numéro de téléphone ne peut pas dépasser les 10 chiffres) pour l'image il faut choisir l'image avec le nom parmi les images dans le dossier du projet



A screenshot of a Qt-style dialog box titled "Ajout_Contact". It features a sidebar on the left with labels for "Nom", "Prénom", "Entreprise", "Mail", "Telephone", and "Photo". Each label is next to a text input field. The "Nom" field contains "Gie", "Prénom" contains "Jimmy", "Entreprise" contains "Entreprise", "Mail" contains "Mail", "Telephone" contains "0723162312", and "Photo" contains "image.jpeg". To the right of these fields is a larger area displaying a summary of the entered data: "Nom : Gie", "Prénom : Jimmy", "Entreprise : Entreprise", "Mail : Mail", "Téléphone : 0723162312", and "Photo : image.jpeg". Below this summary is a circular icon representing a person. At the bottom of the dialog are two buttons: "Ajouter" and "Annuler".

Lorsque l'utilisateur ajoute le contact avec le bouton un signal est envoyé à la MainWindow pour bien ajouter dans la liste de contact et la base de données et la fenêtre d'ajout se ferme.

Pour la suppression, la fenêtre va supprimer un contact en fonction de son nom ou de son prénom (ou des deux s'il y a plusieurs du même nom ou prénom), il code est fait pour qu'à chaque changement dans les qlabel, une gestionContact temporaire est créée et le code y ajoute tous les contact avec le nom ou prénom donnée pour les afficher. Lorsque qu'il ne reste plus qu'un contact, l'utilisateur peut le supprimer avec le bouton (s'il y a 2 contact de même nom et donc sélectionné, le code va bloquer la suppression pour ne pas en supprimer plusieurs).



A screenshot of a Qt-style dialog box titled "Supprimer_Contact". It has a sidebar on the left with labels for "Nom" and "Prénom", each followed by a text input field. The "Nom" field contains the letter "G". To the right of these fields is a large text area that displays the message "il n'y a aucune corespondance". At the bottom of the dialog are two buttons: "Supprimer" and "Annuler".

Lorsqu'il n'y a aucun contact trouvé avec les infos donnés.

The dialog box 'Supprimer_Contact' has a dark title bar with a close button. It contains two input fields on the left: 'Nom' (with 'Gie' entered) and 'Prénom' (empty). On the right, a text area displays the following information:

```

contact 1 :
Nom: Gie
Prenom: Jimmy
Entreprise: Entreprise
Mail: Mail
Téléphone: 0723162312
URL Photo: image.jpeg
Date de création: 22/12/2023 17:29
Interactions de Gie Jimmy:
{
}

```

At the bottom are two buttons: 'Supprimer' and 'Annuler'.

Un contact trouvé à l'aide du nom seulement

The dialog box 'Supprimer_Contact' is shown with the same layout as the first screenshot. The 'Nom' field contains 'Gie'. The text area on the right displays two contact entries:

```

Entreprise: Entreprise
Mail: Mail
Téléphone: 0723162312
URL Photo: image.jpeg
Date de création: 22/12/2023 17:29
Interactions de Gie Jimmy:
{
}

contact 2 :
Nom: Gie
Prenom: Martin
Entreprise: Ent
Mail: Mail
Téléphone: 0723712837
URL Photo:
Date de création: 22/12/2023 17:34
Interactions de Gie Martin:
{
}

```

The 'Prénom' field is empty. The 'Supprimer' and 'Annuler' buttons are at the bottom.

2 contacts trouvés à l'aide du nom

Les fenêtres de modification et d'ajout d'interactions-Todo utilisent le même système de sélection de contact pour la modification et l'ajout d'interactions.

Pour la modification, lorsque le contact est sélectionné, ses informations à modifier s'ajoutent aux QLineEdit de modification pour pouvoir garder les anciennes valeurs si l'utilisateur ne veut en modifier qu'une seule

contact 1 :

Nom: Gie

Prenom: Jimmy

Entreprise: Entreprise

Mail: Mail

Téléphone: 0723162312

URL Photo: image.jpeg

Date de création: 22/12/2023 17:29

Interactions de Gie Jimmy:

{

}

Nom

Gie

Prénom

Jimmy

Entreprise

Entreprise

Mail

Mail

Telephone

0723162312

Photo

image.jpeg

Modifier

Annuler

Pour l'interface d'ajout d'interactions et de todo, après le choix du contact, il suffit de remplir l'annotation de l'interaction et voulu d'ajouter un ou plusieurs todo pour l'interaction (il est possible de faire une interaction sans Todo)

Ajout_Interaction_Todo

contact 1 :

Nom: Logerot

Prenom: Alexis

Entreprise: Entreprise

Mail: Mail

Téléphone: 0723612372

URL Photo: image.jpeg

Date de création: 22/12/2023 17:29

Interactions de Logerot Alexis:

{

}

Nom

Logerot

Prénom

Ajout Interactions

Interactions

Projet CDAA

Todo

Rendre le Projet

12/22/23 11:00 PM

AjoutTodo

Annuler

Lorsque le choix de la date et du texte du todo a été fait il suffit d'utiliser le bouton AjoutTodo pour l'ajouter a l'interaction, un petit texte apparaîtra pour valider l'ajout.

Todo 1 Ajouté

12/22/23 11:00 PM

AjoutTodo

Revenons à la MainWindow, nous pouvons donc afficher la liste de contact avec le menu affichage mais aussi afficher l'historique ou trier par nom ou par date les contacts.

Affichage

- AfficheContact
- Historique
- Tri Alphabetique ▶
- Tri par Date ▶

L'historique permet de voir les changement fait depuis l'interface avec leurs date d'exécution

Historique des changements de gestionContact :

22/12/2023 17:29 : ajout du contact Logerot Alexis
22/12/2023 17:29 : ajout du contact Gie Jimmy
22/12/2023 17:34 : ajout du contact Gie Martin

Et pour finir le menu déroulant permet de voir tous les contact, et de voir ses informations détaillé du contact choisi

Logerot Alexis

Gie Jimmy

Gie Martin

Présentation des classes :

Classe Contact :

La classe Contact représente les informations et les interactions avec un contact. Elle stocke les attributs : le nom, le prénom, l'entreprise, l'adresse e-mail, le numéro de téléphone, l'URL de la photo, la date de création, et une liste d'interactions.

1. Constructeurs :

- Le constructeur par défaut crée un objet Contact avec des valeurs si aucune information n'est fournie.
- Le constructeur avec paramètres permet de créer un contact en fournissant des valeurs spécifiques pour ses propriétés, comme le nom, le prénom, l'entreprise, l'adresse e-mail, etc.

2. Destructeur :

- Le destructeur de la classe est appelé lorsqu'un objet Contact est détruit libérant ainsi la mémoire

3. Ajout d'Interactions :

- Nous permet d'ajouter des interactions à un contact. Les interactions peuvent être des événements ou des actions liés à ce contact, comme des appels téléphoniques, des réunions, des courriels, etc(que l'on pourrait qualifier d'agenda).

4. Surcharge de l'opérateur de sortie (operator<<) :

- Cette surcharge nous permet d'afficher les détails du contact, pour l'afficher dans le terminal. Elle affiche le nom, le prénom, l'entreprise, l'adresse e-mail, le numéro de téléphone, la date de création et les interactions associées au contact.

5. Surcharge de l'opérateur d'égalité (operator==) :

- Cette surcharge permet de comparer deux contacts pour vérifier s'ils sont égaux. Deux contacts sont considérés égaux si leurs noms, prénoms, entreprises et adresses e-mail sont identiques.

De manière générale notre classe Contact est utile pour stocker et gérer les informations relatives à un contact.

Classe GestionContact :

1. Constructeur de GestionContact :

- Le constructeur par défaut GestionContact::GestionContact() initialise un objet de la classe GestionContact.
- Le destructeur GestionContact::~~GestionContact() nettoie la liste de contacts en la vidant.

2. Méthode addContact :

- Nous permet d'ajouter un contact à la liste de contacts en utilisant la fonction push_back.

3. Méthode getListe :

- Nous renvoie une copie de la liste de contacts actuelle.

4. Méthode ajouteHisto :

- Ajoute un élément à l'historique (Histo). Elle prend un message en paramètre, crée un objet Historique avec ce message, et l'ajoute à la liste d'historique.

5. Méthode modifContact :

- Modifie un contact existant en remplaçant l'ancien contact par un nouveau. Elle recherche le contact par son nom, puis ajoute un enregistrement à l'historique pour indiquer la modification.

6. Méthode removeFirstByName :

- Supprime le premier contact ayant un nom spécifique de la liste de contacts. Elle ajoute également un enregistrement à l'historique pour indiquer la suppression.

7. Méthode removeByName :

- Supprime tous les contacts ayant un nom spécifique de la liste de contacts. Elle ajoute des enregistrements d'historique pour chaque contact supprimé.

8. Surcharge des opérateurs :

- Les opérateurs + et - sont surchargés pour permettre d'ajouter et de supprimer des contacts de la liste. Ces opérations sont également enregistrées dans l'historique.
- L'opérateur = est surchargé pour copier les contacts d'un autre objet GestionContact.

9. Méthode afficheHisto :

- Nous affiche l'historique des changements de GestionContact, en itérant sur la liste d'objets Historique et en utilisant la méthode Affichage de la classe Historique pour afficher chaque enregistrement d'historique.

10. Méthode TrierParNomAZ() et TrierParNomZA() :

Permet le tri de la liste de contact à l'aide du nom de chaque contact de A à Z et inversement pour l'affichage.

11. Méthode TrierParDateRecent() ; et TrierParDateAncien() :

Permet le tri de la liste de contact par la création de chaque contact du plus récent au plus ancien ou du plus ancien au plus récent.

12. Méthode ajoutHistoSQL :

Ajoute à la table SQL histo l'historique de chaque création, modification faites au contact ou autres pour les garder au relancement du code.

De manière générale la classe GestionContact est conçue pour gérer une liste de contacts et suivre l'historique des changements effectués sur ces contacts. Elle permet l'ajout, la modification et la suppression de contacts, ainsi que la possibilité d'afficher l'historique des actions effectuées sur ces contacts.

Classe Historique :

1. Constructeur de la classe Historique :

- Le constructeur Historique::Historique(std::string m) est utilisé pour créer un objet Historique. Il prend en paramètre une description de la modification m.
- Dans le constructeur, la date de l'historique est initialisée à la date actuelle en appelant le constructeur par défaut de la classe Date.

2. Méthode Affichage :

- Est utilisée pour afficher les détails de l'historique, y compris la date et la description de la modification.
- Elle utilise std::cout pour afficher la date au format "jour/mois/année heure:minute", suivi de la description de la modification.

De manière générale notre classe Historique est conçue pour stocker et afficher les enregistrements des modifications et suppressions de nos contacts par l'utilisateur, ainsi qu'une court description de celle-ci pour savoir qui a été modifié et en quoi et qui a été supprimé.

Classe Interaction :

1. Constructeur par défaut et constructeur surchargé :

- Le constructeur par défaut `Interaction::Interaction()` initialise un objet `Interaction` avec la date actuelle en utilisant la fonction `getCourante()` de notre structure `Date`.
- Le constructeur surchargé `Interaction::Interaction(const std::string& c, const struct Date d)` permet de créer un objet `Interaction` en spécifiant le contenu (`c`) et la date (`d`) de l'interaction.

2. Accesseurs :

- La classe fournit des méthodes d'accès pour obtenir la date de l'interaction, le contenu de l'interaction et la liste de "Todos" associés à l'interaction.

3. Mutateurs

- Les méthodes `setDate` et `setContenu` permettent de définir la date et le contenu de l'interaction, respectivement.

4. Méthode `ajouteTodo` :

- Nous permet d'ajouter un objet `Todo` à la liste de "Todos" associés à l'interaction.

5. Surcharge de l'opérateur de sortie (`operator<<`) :

- La surcharge de l'opérateur `<<` permet d'afficher les détails de l'interaction, y compris la date, le contenu et la liste de "Todos" associés.
- Pour chaque "Todo", elle affiche également les détails de ce "Todo".

De manière générale notre classe `Interaction` permet de représenter des interactions avec une date, un contenu et une liste de tâches à accomplir (les "Todos"). Ces interactions peuvent être utilisées pour enregistrer des actions ou des événements avec des informations associées.

Classe Todo :

1. Constructeurs :

- Le constructeur `Todo::Todo(std::string c, struct Date d)` permet de créer un objet `Todo` en spécifiant le contenu (`c`) et la date (`d`) de la tâche.
- Le constructeur par défaut `Todo::Todo()` initialise un objet `Todo` avec un contenu par défaut, en l'occurrence, "aucun contenu".

2. Accesseurs :

- La classe fournit des méthodes d'accès pour obtenir le contenu et la date du `Todo`.

3. Mutateurs :

- Les méthodes `setContenu` et `setDate` permettent de définir le contenu et la date du `Todo`, respectivement.

4. Surcharge de l'opérateur de sortie (`operator<<`) :

- La surcharge de l'opérateur `<<` permet d'afficher les détails du `Todo`, y compris la date et le contenu.

La classe `Todo` nous permet de représenter des tâches (todos) avec des informations telles que le contenu et la date entrée manuellement.

Diagrammes des classes :