13/12/2021

TRAORÉ Moussa

Rapport de projet

CONCEPTION ET DEVELOPPEMENT AVANCE D'APPLICATION

Table des matières

Partie interface utilisateur

Classe MainWindow

La classe *MainWindow* est la fenêtre d'interface principal avec l'utilisateur, elle s'occupe de faire le lien entre les données et l'utilisateur. L'interface se divise en deux partie, la partie gauche est destinée à la visualisation des données, *Contact*, *Interaction* et *Todos* par un *QTabWidget* et des classes héritant de *QWidget* spécialisées, *VisuContactWidget, VisuInteractionWidget et VisuTodoWidget*. C'est également dans cette partie que les contacts pourront être modifier ou supprimer. La partie droite est destinée à faire des recherche : on y trouve une barre de recherche pour chercher des motifs présent dans les objets, une fonction recherche avancée qui permet de spécifier s'il on veut voir les contact, Interactions ou todos, ainsi que la recherche par date ou par contact pour les interactions et todos, le tout sera afficher dans un *QTableView*. Une troisième partie de l'interface utilisateur est le menu qui permet d'ouvrir des boîtes de dialogues pour ajouter un contact ou une interaction.

La classe en elle-même dispose d'attributs pour lier le tout, notamment une instance de basededonnée et un pointeur sur une instance de gestion contact qui contient tous les contacts et les interactions ainsi que des listes de interactions et todos qui sont les résultats des recherches avancées.

Parmi les fonctions de cette classe, on peut noter les fonctions d'ajout de donnée qui selon si une recherche avancée nécessaire vont soit consulter la base de données et ajouter les objets qui correspondent, ou afficher tout le contenu de l'instance de *GestionContact*. Ensuite il y a de nombreux slots qui sont appeler de façon interne ou externe à la classe. Parmi les premiers, on trouve les résultats des actions du menu, qui créent la boîte de dialogue adéquate puis qui sauvegardent la base de données avec les nouvelles valeurs, le slot *modifModel* qui avec un entier qui représente l'identifiant du bouton sélectionné dans la recherche avancée (1 pour contact, 2 pour interaction et 3 pour les todos) modifie les valeurs de la *comboBox* de tri, les titres du tableau de visualisation et appel la fonction adéquate pour remplir le tableau, ainsi que le slot *affiche* qui est appelé lorsque l'on sélectionne un élément du tableau de visualisation : on récupère l'indice de l'élément sélectionné, dans la première colonne et on créer des instance des *Widgets* spécialisé que l'on place dans la partie gauche au sein d'un *QTabWidget*. On connecte si besoin les signaux des *widget* au slots activable de l'extérieur de la classe. Ces slots sont les suivants : *effaceContact* qui reçoit un contact à effacer *afficheTabTodo* qui créeune instance de *VisuTodoWidget* pour visualiser le *todo* à l'identifiant passé en paramètre et l'ajoute à la partie de gauche de l'application et sauvegarde pour sauvegarder l'instance de gestion contact dans la base de données

Classe FenetreAjoutContact

La classe hérite de l'objet *QDialog*, il s'agit d'une boîte de dialogue utilisée pour ajouter un contact dans l'application. La classe possède en attribut les chaine de caractère à ajouter dans le nouveau contact ainsi qu'en particulier une instance de *ImageWidget,* un widget pour afficher une image. La classe possède également une fonction *ParcourirImage* pour aller selection un chemin de fichier à l'aide d'une *QFileDialog*. Ainsi qu'une méthode sauvegarde pour vérifier que tous les champs sont remplis avec des valeurs correct, notamment la vérification du numéro de téléphone et de l'adresse mail.

Classe FenetreAjoutContact

La classe hérite de l'objet *QDialog*, il s'agit d'une boîte de dialogue utilisée pour ajouter une interaction dans un contact. La classe possède en attribut un vecteur de contact, à partir duquel on choisi un contact auquel ajouter une interaction. Lors de la sélection au sein de la *QComboBox*, on appelle le slot *selection* qui place le contact choisi comme attribut et permet la construction de l'interaction. Le bouton « Ajouter une tâche à faire » permet ensuite de créer un todo pour l'interaction en création. Enfin le slot *sauvegarde* crée l'instance de la classe Interaction à partir des données

communiqués et émet le signal fini, reçu par la *mainWindow* pour sauvegarder la base de données et par la même occasion attribuer un identifiant cohérent à l'interaction.

Classe ImageWidget

Cette classe hérite de l'objet *QWidget* et est spécialisé dans l'affichage d'images. L'image donnée en paramètre est affichée et si ce n'est pas possible, l'image anonyme.jpg est afficher à la place. Pour réaliser ce *Widget* nous avons réimplémenté l'évènement d'affichage du *Widget* en dessinant dessus l'image en paramètre.

Classes d'affichage d'objets

Dans cette partie nous abordons les classes *VisuContactWidget, VisuInteractionWidget* et *VisuTodoWidget*. Comme leur nom le laisse deviner, ces classes héritent de l'objet *QWidget.* Ces classes sont toutes composé de manière à afficher le contenu de l'objet qui les correspond. En particulier la classe *VisuContactWidget*, on a ajouté la fonctionnalité de modification par la méthode *sauvegarde* et de suppression avec le slot supprimer qui émet le signal efface en direction de la *MainWindow*.

Partie Gestion de la base de données

Schéma de la base de données Sqlite

Il y a 3 tables décrites comme suite :

CONTACT (nom, prenom,entreprise,mail,telephone,photo,dates,**id**) ;

INTERRACTION *(idCONTACT*, contenu,dates**,id**) ;

TODO (*idINTERAC*, contenu,**id,**dates) ;

Un contact à un nom (text), un prénom(text), une entreprise(text), un mail (text), un numéro de téléphone (text), une photo (text), une date (text) et un identifiant (integer) **id.**

Une interaction possède l'identifiant *idCONTAC (integer)* de son contact et possède un contenu(text) et un identifiant propre à lui **id (integer).**

Enfin un todo possède l'identifiant de son interaction *idINTERAC(integer),* un contenu (text) *et son* **id (integer)**

Classe Basededonnee

Cette classe sert d'interface entre l'IHM et la base de données facilitant ainsi les échanges de données.

Elle permet tout d'abord de récupérer l'ensemble des données de la base qui seront par la suite stockés dans sa variable de type *gestionContact*. Et c'est dans celle-ci qu'on fera toutes nos opérations notamment le retrait global des éléments de la base, les mise à jour fréquentes.

Elle permet aussi d'afficher des listes de contact, interactions ou de todos par nom et/ou par date pour les contact ou soit contact et/ou par date pour les interactions et todos.

Voici une explication détaillée des méthodes principale de la classe :

Explication de la methode getAll() :

Cette méthode permet de faire une copie de la base donnée dans la variable de type gestionContact de la classe. Pour cela on effectue une requêtes *QSlqQuery* sur la table contact retournant toutes ses lignes.

Ainsi pour chaque ligne on crée une instance de la classe contact avec éventuellement les colonnes de la ligne tout en pensant à modifier les indices de l'instance car indice ne fait pas partie des paramètres du constructeur.

Pour chaque instance de contact on effectue une nouvelle requête maintenant sur la table interactions qui comme pour la table contact retournera toutes les lignes et une instance d'interaction sera créer pour chacune de ses lignes. Ces instances seront ajoutées à la listes d'interactions de l'instance de contact correspondantes et permettront entre autres d'aller chercher les todos correspondantes en effectuera une dernière fois une requête sur la table todos. Pour chaque ligne on créer une instance de todo qui sera ajouté dans la liste des todos de l'instance d'interactions correspondante.

Enfin on ajoute le contact dans la variable gestioncontact.

Ce processus s'effectuera pour chaque ligne de la table contact et ainsi à la fin on a une copie complète de la base de données qu'on pourra facilement utiliser pour nos utilisations.

Explication de la méthode Ajoutcontact :

Dans un premier temps on insère le contact dans la table contact et si sa liste d'interactions est non vide alors on insère toutes ses interactions. Pour chacune de ces lignes s'il existe une liste de todos ils seront tous insérer à leur tour.

Pour respecter les contraintes de la base de données nous avons pensé à récupérer pour chaque connexion à la base de données l'indice courant de chaque table. Qui s'incrémenteront au fur à mesure des ajouts dans les tables respectives.

Par exemple :

Pour chaque ajout de contact l'idContact sera incrémenté et ces pareils pour les interactions et les todos et la colonne idCONTACT aura la valeur idContact-1 ;

Explication de la méthode sauvegarde :

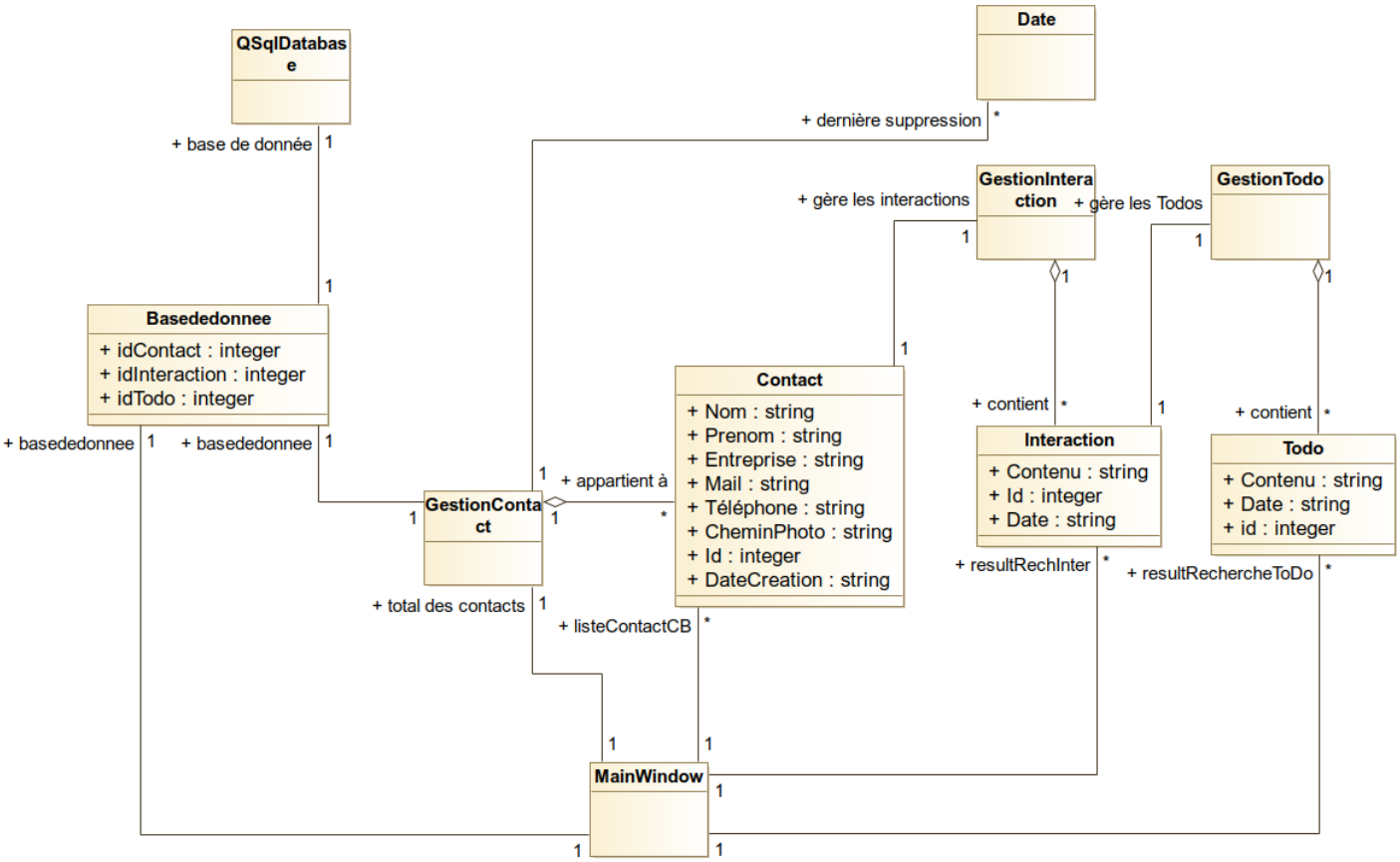
On récupère dans une nouvelle variable le contenu de notre variable gestion contact celle dans laquelle on avait stocker la copie de notre base de données qui n'est sans doute plus la même.

Ensuite on fait appel à la méthode getAll, on aura ainsi deux versions des données une qui n'a pas été toucher depuis la dernière mise à jour et l'autre qui a été modifier et qu'on voudrait ajouter dans la base mais pour des raison d'optimisation on parcourra les deux versions et on ajoutera et supprimera que ce qui doivent l'être.

Partie programmation orientée objet

Pour la partie programmation orientée objet, nous avons créé sept classes, trois classes de gestion de liste et quatre classes pour gérer les données.

Diagramme UML des données



Les classes de gestion de données

On regroupe dans cette partie les classes *Contact*, *Interaction*, *toDo* et *Date*. Ces classes sont fortement dépendante et donc reliées, en effet une interaction dépend d'un contact et un toDo dépend d'une interaction, c'est pour cela qu'elles disposent en attribut d'un pointeur vers leur instance « propriétaire », c'est-à-dire celle qui les a créés. Les classes Contact et Interaction disposent aussi en attribut une instance de classe de gestion de liste que nous aborderont en seconde partie. Toutes les données sont horodatées, nous devons donc générer et stocker ces dates. Pour cela les classes ont besoin d'appeler une instance de date pour obtenir une date qui sera ensuite stocker dans une chaîne de caractères. Nous avons fait ce choix car les dates ne sont pas destinées à changer.

Classe Contact

La classe doit contenir les informations relatives à un contact (son nom, prénom, entreprise, téléphone, adresse mail ainsi que le lien vers une photo et la date de création) qui sont enregistrés sous forme d'une chaîne de caractère, ainsi qu'un identifiant unique parmi les contacts. De plus, la classe possède un attribut de type GestionInteraction qui sert à contenir les interactions relatives au Contact, dont nous discuterons dans une seconde partie. Pour les fonctions de la classe, nous avons des accesseurs en lecture et en écriture classique mis à part une subtilité pour les setters, chaque appel de ces fonctions génère une nouvelle interaction témoignant des modifications apportés au contact. La classe possède également deux méthode pour ajouter/enlever des interactions dans la liste, ces fonctions sont juste une manière de simplifier la création des Interaction qui sont ensuite ajoutés au sein de l'instance de la classe de gestion de liste. Pour terminer, la classe possède une surcharge de l'opérateur de comparaison d'égalité qui compare les attributs de deux instances de Contact et de l'opérateur de flux pour l'affichage en console.

Classe Interaction

La classe désigne une interaction avec un contact, elle contient donc un contenu enregistré sous forme de chaine de caractère et une date générée de manière automatique comme pour les Contacts ainsi qu'un identifiant unique parmi les interactions. La classe est très semblable à la précédente mais l'on peut noter la particularité de la méthode *setContenu* qu'il convient d'expliquer : à la création de l'interaction, la classe reçoit sous forme de texte plusieurs instructions sous forme d'un texte multiligne duquel on distingue deux cas, les instructions dites classiques par oppositions aux instructions balisés, on les distingue par la présence en début de ligne de « @todo ». Les premières seront mises sans modifications dans la partie contenu de la classe tandis que les secondes seront intégrées sous forme d'instances de classe toDo. La fonction sert à faire la différence entre ces deux types de contenu.

Algorithmiquement, tant que l'on n'a pas traité tout le contenu de la chaine placé en paramètre, on recherche la présence dans la chaîne du mot clef « @todo », si il existe on envoie le texte précédent comme instruction classique dans l'attribut *contenu* et le texte suivant, jusqu'à la fin de la ligne ou du texte s'il s'agit de l'ultime ligne, sert à la création d'une instance de toDo ajouté ensuite à la liste de l'instance de la classe de gestion de liste.

Classe toDo

La classe désigne une instruction spécialement balisé, ce type d'instruction contient sémantiquement une instruction à faire avant une date donné, cela se représente par un contenu, un date ainsi qu'un identifiant unique parmi les todos. La classe ressemble encore à la précédente au niveau de ses accesseurs et de ses attributs mis à part la classe de gestion de liste qui est inutile dans ce cas. La spécificité de notre classe se trouve dans son constructeur qui décompose la ligne d'instruction. La partie précédent la balise « @date » si présente est enregistrée dans l'attribut contenu de la classe tandis que la partie suivante sert à créer une instance de date particulière. Si cette dernière partie n'est pas présente, la date est initialisée à la date du jour.

Classe Date

La classe Date sert à symboliser une date pour la transmettre sous forme de chaîne de caractère. Pour ce faire nous utilisons un pointeur sur la structure *tm* de la librairie *ctime*. Comme la classe a pour fonction de créer une date, ses fonctions articule dans ce sens. Premièrement, deux constructeurs permettent de créer soit la date du jour, le constructeur sans paramètre, soit une date à partir d'une chaine au format jj-mm-aaaa. Dans ce dernier cas, on utilise la méthode *setDate* avec comme paramètre une chaine de caractère : les différents champs sont isolés en découpant la chaîne au niveau des « - » on vérifie la cohérence des données et on paramètre une instance de *tm* pour avoir une structure depuis laquelle on peut obtenir la date sous forme de chaîne de caractère. Cette dernière fonctionnalité est obtenu par la fonction *getDateToString()* qui récupère les paramètre dans l'instance de *tm* pointée en attribut pour créer une chaine de caractère représentant la date, cette méthode est importante d'une part car nos dates sont stockées sous forme de chaîne de caractère dans les autres classes et d'autre part car bien que l'un de nos constructeur travaille avec une chaîne de caractère en paramètre, ce n'est pas le cas du constructeur par défaut qui lui crée tout de même une date, la date du jour.

Nous avons donc maintenant des classes pour gérer les données et qui sont reliées dans un sens, toDo connait son Interaction et Interaction connait son Contact, il nous reste à les relier dans l'autre sens et comme l'on doit gérer plusieurs instances avec un même propriétaire, nous avons opter pour la création de classes dites de gestion de liste.

Les classes de gestion de liste

Les classes de gestion de liste fonctionnent de manière similaire, mis à part quelques particularités que nous évoquerons dans un point suivant. De manière générale, ces classes comportent un attribut *list* contenant des pointeur sur instances de type éponyme, nous avons fait ce choix pour faciliter l'accès en modification des donnée et non sur des copie des donnée. On dénote

également des fonction pour ajouter des membres à la liste ou en retirer. Pour les méthodes de suppression, on utilise un itérateur sur la liste et on supprime l'instance voulue. Enfin on a aussi des méthode pour accéder à un membre par son identifiant.

Classe GestionContact

La classe GestionContact est un peu différent, elle sert de souche aux autres classes. Elle possède en plus une chaine de caractère qui contient la date de dernière suppression de contact, modifié par la fonction *removeContact*.

Installation

Pour avoir une connexion à la base de donnée correct, il faut placer le fichier « base1 » dans le répertoire */tmp* puis compiler et exécuter le programme.