Rapport de premier jalon

CDAA

MANZANO Lilian
PERE Brandon

Groupe T

Sommaire

1. Architecture des répertoires

Dans le dossier décompressé vous trouverez en plus de ce rapport, 3 répertoires :

- Le répertoire '*src*' qui contient les sources du projet, c'est-à-dire les '*.h*', '*.cpp*', et '*.pro*'.

- Le répertoire '*docs'* où nous mettons tous les documents d'informations sur le projet, vous pourrez ainsi y retrouver les diagrammes pour éventuellement mieux les voir que dans ce rapport, et y retrouver aussi la doc Doxygen générée, celle-ci dans le dossier '*generate_doc_doxygen*'.

- Le répertoire '*data*' qui sera l'emplacement de notre base de données mais aussi celle des fichiers d'initialisations de celle-ci, nous avons déjà fait le fichier '*init.sql*' qui contient les commandes '**CREATE TABLE …**', nous en parlons dans le 3ème point de ce rapport.

2. Le 'main'

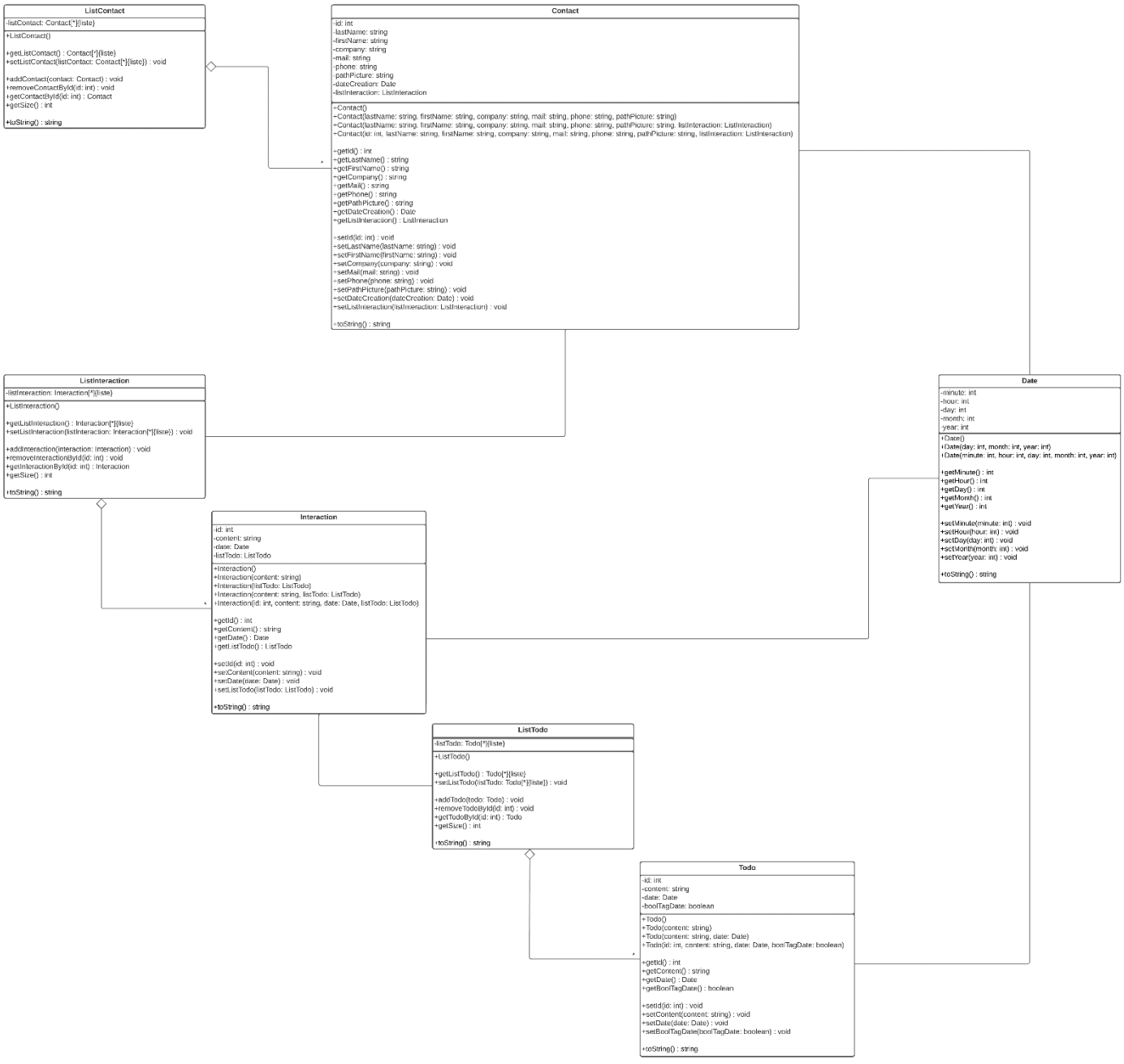
Concernant les tests fais dans le '*main*', nous avons testé chaque classe, ainsi nous avons fait des actions comme : création de dates, création de tags, ajout de tags à une interaction, etc…
Tout est affiché en console avec des explications sur ce que nous faisons.

3. Architecture des classes

Concernant les classes, nous en avons pour le moment 7 :

- Date
- Contact
- Interaction
- Todo
- ListContact
- ListInteraction
- ListTodo

Cela en correspondant à ce diagramme de classe :



Reprenons chaque classe :

- La classe ***Date*** permet donc de gérer des dates, nous stockons jour, mois, année, heure, et minutes, cependant suivant les cas tout ne sera pas utilisé, ainsi par exemple pour stocker la date d'un tag @date nous aurons besoin seulement du jour, du mois, et de l'année, en revanche pour l'horodatage des modifications sur des contacts, nous voulons la date mais aussi l'horaire, c'est ici que les heures et minutes nous seront utiles.

  Cette classe possède des accesseurs sur tous les attributs, et pour le moment uniquement une fonction '**toString()**' qui renvoie un string de la forme 'jj/mm/aaaa', nous pourrons éventuellement en ajouter d'autres par la suite pour obtenir d'autres formats, notamment avec les heures et minutes.

- La classe ***Todo*** permet de gérer un tag @todo, il a comme attribut un identifiant, celui-ci n'a pas grand intérêt pour le moment sans base de données, cependant par la suite à chaque fois qu'un nouveau tag sera créé par l'utilisateur il sera directement inséré dans la base de données qui lui associera un identifiant auto-incrémenté et unique, ainsi les identifiants nous servirons à différencier les tags même avec un contenu similaire, et surtout à faire des requêtes pour la base de données.

  En autre attribut, la classe possède évidemment le contenu qui est un string, un pointeur sur un objet ***Date*** correspondant au tag @date qui peut être ajouté au tag @todo, ce pointeur peut être donc éventuellement nul, pour être sûr de ne pas chercher un élément pointé inexistant, le dernier attribut de la classe est un booléen informant sur la présence d'un tag @date.

  Cette classe possède des accesseurs sur tous les attributs, et pour le moment uniquement une fonction '**toString()**' qui renvoie un string de la forme '@todo …' ou '@todo … @date jj/mm/aaaa', suivant la valeur du booléen.

- La classe ***ListTodo*** permet de gérer une liste de pointeurs vers des objet Todo, elle possède ainsi uniquement un attribut qui est de type '**std::list<Todo*>**'.

  Nous avons décidé de créer une classe pour gérer les listes dans l'objectif d'ajouter des méthodes nous facilitant la gestion de celles-ci et la recherche de tags. Cette classe possède pour le moment en plus des accesseurs, des méthodes pour ajouter un tag à la liste, en supprimer un en fonction de son identifiant, en rechercher un en fonction de son identifiant, et une méthode pour connaitre la taille de la liste.

  Pour le moment, nous avons qu'une méthode de recherches par identifiant, mais nous pourrons éventuellement en ajouter d'autres par la suite suivant nos besoins, par exemple rechercher par date, cela ne sera pas compliqué puisque c'est juste un copier-coller de la méthode de recherche déjà existante mais en remplaçant l'attribut que l'on vérifie.

  Enfin la classe possède une méthode '**toString()**' qui renvoie un string avec tous les tags, cela en utilisant la méthode '**toString()**' de la classe ***Todo*** sur chaque élément de la liste.

- La classe ***Interaction*** est assez similaire à la classe ***Todo***, elle permet de gérer une interaction. Elle possède elle aussi un attribut identifiant qui est présent pour les mêmes raisons que celui de la classe ***Todo***, ce sont donc les mêmes remarques que précédemment.

  En autre attribut, la classe possède évidemment le contenu qui est un string, elle possède aussi un pointeur vers un objet ***Date***, et un pointeur vers un objet ***ListTodo***. Effectivement l'interaction peut posséder plusieurs tags, l'attribut doit donc être une liste.

  La classe possède des accesseurs sur tous les attributs, et une seule méthode '**toString()**' qui renvoie un string avec le contenu de l'interaction puis ses tags @todo, cela en utilisant la méthode '**toString()**' de la classe ***ListTodo***.

- La classe ***ListInteraction*** permet de gérer une liste de pointeurs vers des objets ***Interaction***, nous avons choisi de la faire pour les mêmes raisons que la classe ***ListTodo***, et ces deux classes sont exactement pareilles excepté qu'au lieu de gérer des pointeurs vers des objets ***Todo***, ce sont des pointeurs vers des objets ***Interaction***.

  Alors nous pouvons nous demander pourquoi avoir voulu faire plusieurs classes pour gérer des listes, au lieu de faire une seule classe gérant des listes de pointeurs vers des pointeurs qui aurait donc pu gérer n'importe quels objets. Ceci car malgré que pour le moment ces classes soient identiques elles pourront évoluer par la suite et donc posséder des méthodes différentes, comme des méthodes de recherches adaptées à chaque objet.

- La classe ***Contact*** permet de gérer un contact, elle possède en attribut un identifiant présent pour la même raison que dans les classes ***Todo*** et ***Interaction***, donc mêmes remarques que précédemment.

  Elle possède aussi en attributs de type string tous les éléments caractérisant un contact, c'est-à-dire nom, prénom, entreprise, mail, numéro de téléphone. Pour gérer l'image de profil du contact, nous n'avons pas un attribut de type image d'une quelconque bibliothèque mais un simple string du chemin vers l'image, celui-ci sera plus simple à stocker dans la base de données. En autre attribut, le contact possède un pointeur vers un objet ***Date*** pour la date de création, et enfin un pointeur vers un objet ***ListInteraction***, ainsi depuis le contact nous pouvons accéder à toutes ses interactions, mais aussi à tous ses tags qui sont compris dans les objets de type ***Interaction***.

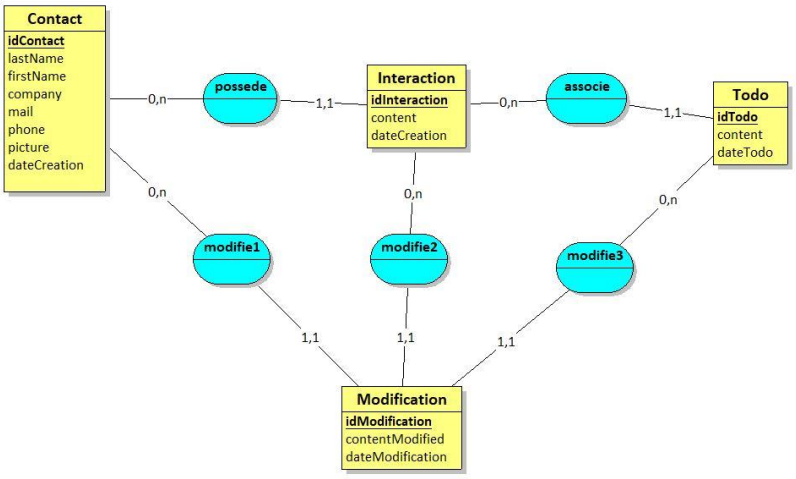
  La classe possède pour le moment des accesseurs pour tous les attributs et une seul méthode '**toString()**' qui renvoie un string avec toutes les informations du contacts, mais aussi toutes ses interactions et donc aussi tous ses tags. Nous pourrons éventuellement en ajouter d'autres par la suite suivant nos besoins, pour par exemple avoir moins d'informations.

- La classe ***ListContact*** permet de gérer une liste de pointeurs vers des objets ***Contact***, nous avons choisi de la faire pour les mêmes raisons que les classes ***ListTodo*** et ***ListInteraction***, et cette classe est exactement pareille que les précédentes excepté le type de pointeurs.

  Comme précédemment nous pouvons nous demander l'utilité de cette classe quand on peut imaginer une classe de gestion de liste de n'importe quel objet, et comme précédemment cela est un choix dû à des méthodes qui pourront être ajoutées par la suite suivant nos besoins, nous pouvons notamment imaginer des méthodes de recherches par mail ou encore par entreprise, chose inutile pour des interactions et des tags.

4. Architecture de la base de données

Nous avons déjà commencé à concevoir la base de données, en voici son diagramme :



Nous pouvons voir sur ce schéma 4 tables, nous retrouvons nos objets ***Contact***, ***Interaction***, et ***Todo***, mais la 4ème table nous n'en avons encore jamais parlé. C'est une table qui stockera les modifications notamment l'horodatage de celle-ci, et nous pourrons en plus de la date avoir un contenu, par exemple '*modification de l'entreprise du contact Jean Dupont : UB -> UFR Sciences et techniques*'. Nous ferons aussi probablement par la suite une classe pour gérer ceci.

Nous avons en plus du diagramme fait un fichier '*init.sql*' que vous retrouverez dans le dossier '*data*', ce fichier contient les commandes SQL pour créer ces tables. Ces tables sont ainsi liées :

- un contact à un identifiant en clé primaire ;
- une interaction à un identifiant en clé primaire mais aussi un identifiant de contact en clé étrangère, ainsi nous pourrons savoir à quel contact est lié cette interaction ;
- un todo à un identifiant en clé primaire mais aussi un identifiant d'interaction en clé étrangère, ainsi nous pourrons savoir à quelle interaction est lié ce tag ;
- la modification à un identifiant en clé primaire, mais aussi 3 identifiant en clé étrangère, un identifiant contact, un identifiant interaction, et un identifiant todo, ainsi suivant quelle clé étrangère n'est pas nulle nous pourrons savoir quel type d'élément concerne la modification et en plus savoir lequel exactement.