RAPPORT DE PROJET

Structure de Données

GESTION D'UN

CARNET

DE RENDEZ-VOUS

BARRÈRE Manuel
JANON Alexandre
POMMIER Logan

TABLES DES MATIÈRES

TABLES DES MATIÈRES

1. INTRODUCTION

Ce projet concerne la réalisation d'un carnet de rendez-vous, ainsi que de leurs différents participants. A travers ce projet, nous recherchons à créer une application permettant à un utilisateur quelconque de gérer un carnet de rendez-vous. Il s'agit de permettre à ce dernier d'ajouter des rendez-vous à des dates et des heures précises, également d'en enlever, ainsi que d'ajouter des personnes ayant un nom, un prénom, un numéro de téléphone ainsi qu'une adresse de messagerie électronique, le tout à travers une base de données. Il s'agit également de créer une interface graphique répondant au mieux aux conditions réelles d'utilisation d'un tel carnet de rendez-vous.

Dans ce document, nous allons détailler les différents objectifs à atteindre, les décisions prises pour la réalisation de l'application, la structure et l'organisation de notre programme, l'utilisation de l'interface graphique et enfin les différentes contributions à ce projet et remerciements.

2. OBJECTIFS

Les objectifs de ce carnet de rendez-vous sont les suivants : permettre à l'utilisateur de créer un rendez-vous avec une date, une heure de début et une heure de fin. Il pourra également y ajouter, respectivement y enlever, des participants. Il aura non seulement la possibilité de modifier la date et les heures d'un rendez-vous, mais également les données personnelles de chaque personne présente dans la base de données, telles que le numéro de téléphone ainsi que l'adresse de messagerie électronique. Une interface graphique est à réaliser pour permettre une utilisation simple et intuitive du carnet de rendez-vous par l'utilisateur.

2.1. La gestion du carnet

Pour réaliser ces objectifs, nous avons besoin de créer plusieurs éléments essentiels à la gestion du carnet.

Premièrement, il nous faut créer une classe *Date*. Cette classe nous permettra de stocker et gérer efficacement une date comprenant un jour, un mois et une année.

Également, une classe *Hour* est à prévoir. Celle-ci nous permettra de stocker et gérer aisément une heure qui comprend des heures, des minutes et des secondes.

Ensuite, la classe *Personne* qui modélise une personne de notre base de données. Cette personne contiendra un nom, un prénom ainsi qu'un numéro de téléphone et une adresse de messagerie électronique. Chaque personne contient également une liste de tous les rendez-vous auxquels elle doit participer. Toute personne n'est pas obligatoirement reliée à un rendez-vous, et ne peut être présente à deux rendez-vous qui ont lieu au même moment.

Par conséquent, il faut prévoir une classe *RDV* qui représente un rendez-vous. Chaque rendez-vous est symbolisé par un nom, et se déroule à une date avec un horaire de début et de fin. Une liste de tous les participants du rendez-vous est jointe.

Nous prévoyons une classe *LDCP* qui n'est autre qu'une Liste Doublement Chaînée de Personne. Cette classe servira donc à stocker et gérer toutes les personnes qui seront dans notre base de données.

Par conséquent, une classe *LDCR*, acronyme de Liste Doublement Chaînée de Rendez-vous, servira au stockage et à la gestion de tous les rendez-vous présents dans les bases de données.

Enfin, la classe *Manager* connectera l'ensemble de toutes les classes énumérées précédemment et assurera la cohésion de leurs différentes méthodes qui forment la gestion du carnet de rendez-vous.

2.2. L'interface graphique

L'interface graphique représente le lien entre l'utilisateur et le programme. C'est la seule façon pour l'utilisateur d'interagir avec le programme, il est donc primordial d'implémenter une interface simple et efficace pour ce dernier.

L'application sera donc présentée sous la forme suivante :

- Une barre de menus permettant à l'utilisateur d'effectuer diverses actions, explicitement indiquées, concernant la base de données des personnes et des rendez-vous
- Des boutons modifiant l'interface et permettant à l'utilisateur d'interagir avec le carnet de rendez-vous et donc de répondre aux besoins d'une telle application
- Un affichage en temps réel des deux bases de données

Lors du chargement des fichiers externes (menu Fichier, Ouvrir un Fichier), il est recommandé de charger d'abord le fichier contenant la base de données des personnes puis celle des rendez-vous.

3. STRUCTURE DU PROGRAMME

3.1. Définition des classes

Le projet est constitué de neuf classes comprenant chacune un fichier header et un fichier source, permettant respectivement de définir et d'implémenter, les différentes méthodes et fonctions utiles à chaque classe. La fonction d'exécution principale se trouve dans le fichier main.cpp.

3.1.1. Classe Date

D'abord, la classe *Date* sert à représenter une date donnée. Dans cette dernière, il y a premièrement les constructeurs par défaut, de recopie et avec paramètres. Il y a, ensuite, les principales surcharges d'opérateurs servant à afficher, calculer, modifier et comparer une date. De plus, les méthodes dites *getters* et *setters* servent à obtenir ou à modifier un paramètre en particulier de la date courante. Ensuite, on définit des méthodes particulières telles que les méthodes *add()* et *remove()* qui ajoute ou enlève un nombre de jours à une date courante. Ces dernières tiennent compte du nombre de jours dans un mois, grâce à la méthode *lengthMonth()* et du fait qu'une année soit bissextile ou non, grâce à la méthode *isLeap()*. La méthode *afficher()*, comme son nom l'indique, affiche les éléments de la date courante sur la sortie standard. Quant à la méthode *compareTo()*, elle permet la comparaison de deux dates. Les méthodes *toQString()* et *toString()* convertissent la date courante respectivement en chaîne de caractères spécifiques à Qt et en chaîne de caractères standard, tandis que la fonction globale *stoDate()* convertit une chaîne de caractères en Date. La classe *Date* contient également une fonction globale, nommée *today()*, définie en ami afin de permettre à cette classe de récupérer la date du système lorsque la fonction est appelée. La surcharge de l'opérateur *operator<<()* de la classe *ostream* permet à une date d'être envoyée sur un flux sortant.

3.1.2. Classe Hour

Puis, on définit la classe *Hour* qui sert à représenter une heure donnée. Premièrement, on définit les constructeurs par défaut, par recopie et avec des paramètres tels que l'heure, la minute et la seconde. On suppose, pour plus d'aisance, que les secondes sont, par défaut, initialisées à zéro. On suppose que les paramètres sont toujours valides. Ensuite, il y a les méthodes définissant les principales surcharges d'opérateurs permettant de calculer, modifier, comparer ou encore afficher une heure. A noter que les surcharges d'opérateurs *QString* et *string* permettent la conversion d'une heure en chaîne de caractères comme le font les méthodes *toQString()* et *toString()*, à l'exception qu'elles permettent un raccourci d'écriture. Les méthodes *add()* et *remove()* permettent respectivement l'ajout et la

suppression de l'heure en paramètre. La méthode *afficher()* définit l'affichage en format heure, minute, seconde. Puis, la méthode *compareTo()* permet la comparaison entre deux heures. Il y a, de plus, les méthodes *getters* et *setters* permettant la récupération ou la modification d'un des paramètres de l'heure courante. La fonction globale, nommée *now()*, est définie en ami de la classe *Hour* afin de permettre à cette dernière d'obtenir l'heure du système à chaque appel de la fonction globale. Puis, la fonction globale *stoHour()* convertit une chaîne de caractères en une heure. La surcharge de l'opérateur *operator<<()* de la classe *ostream* permet à une heure d'être envoyée sur un flux sortant.

3.1.3. Classe Personne

Après cela, on définit la classe *Personne* qui gère les caractéristiques propres d'une personne. On commence par définir la classe *RDV*, décrite plus bas, ainsi que le constructeur ayant comme paramètres le nom, prénom, numéro de téléphone et email de la personne. Elle aura également un tableau dynamique de pointeurs de rendez-vous afin de visualiser les rendez-vous de la personne. On définit également les méthodes principales de surcharges d'opérateurs permettant de comparer une personne en paramètre à celle courante. En outre, les méthodes *addRDV()* et *removeRDV()* permettent l'ajout ou la suppression d'un rendez-vous pour la personne courante. La méthode *afficher()* s'occupe essentiellement de l'affichage à l'écran. Puis, *compareTo()* est une méthode servant à comparer une personne à une autre afin d'éviter notamment une réplique identique. Les méthodes *rdvtoQString()* et *rdvtoString()* convertissent un rendez-vous en chaîne de caractères propre à Qt ou non. Il y a également les deux méthodes appelées respectivement *toQString()* et *toString()* qui convertissent une personne en chaîne de caractères, c'est-à-dire, en une chaîne contenant le nom, prénom, numéro de téléphone et email de la personne. En outre, les méthodes dites *getters* et *setters* correspondent à chacun des paramètres d'une personne à savoir le prénom, nom, numéro de téléphone, email et liste de rendez-vous.

3.1.4. Classe LDCP

Subséquemment, on définit la classe *LDCP* qui gère les listes doublement chaînées de personnes. On y déclare et définit deux constructeurs : l'un par défaut et l'autre par recopie. En outre, on définit un destructeur car on a besoin, pour détruire une personne, de supprimer la liste de rendez-vous auxquels elle devait participer. Après, on précise des méthodes de surcharges d'opérateurs de comparaison telles que l'affectation, l'égalité et la différence et, afin de permettre un raccourci d'écritures non négligeable, on définit la

surcharge de l'opérateur *operator[]()* qui sert à accéder aux éléments de la liste doublement chaînée de pointeurs de personnes. On définit aussi la méthode *afficher()* qui, comme son nom l'indique, affiche une liste de personnes sur la sortie standard. La méthode *rechD()* permet de rechercher un élément de la liste chaînée en utilisant la recherche dichotomique pour permettre un parcours optimal de cette liste. De surcroît, *insérer()* permet l'insertion d'un élément dans la liste de personnes en tenant compte des différents cas possibles. La méthode *size()* renvoie la taille de la liste doublement chaînée. Puis, la fonction globale de surcharge de l'opérateur *operator<<()* de classe *ostream* permet un affichage plus aisé de la liste doublement chaînée sur un flux de sortie.

3.1.5. Classe RDV

Ensuite, on définit la classe *RDV* qui représente un rendez-vous avec toutes ses caractéristiques. On commence par définir la classe *Personne* en ami et à définir le constructeur ayant en paramètres le nom, la date, l'heure à laquelle ce dernier débute et l'heure à laquelle ce dernier prend fin. De plus, on définit une liste doublement chaînée de pointeurs de personnes comme paramètres du constructeur, qui représente les membres du rendez-vous. Puis, on définit les méthodes correspondant aux surcharges d'opérateurs de comparaison et d'affectation. Les méthodes *addMember()* et *removeMember()* permettent respectivement l'ajout et la suppression d'un membre du rendez-vous courant. En outre, la méthode *compareTo()* compare deux rendez-vous et *afficher()* affiche le rendez-vous courant sur la sortie standard par défaut. Les deux méthodes nommées *participantsToQString()* et *participantsToString()* convertissent un membre du rendez-vous en chaîne de caractères avec ses éléments correspondants tandis que les deux méthodes *toQString()* et *toString()* convertissent le rendez-vous courant en chaîne de caractères. Après cela, on définit les méthodes dites *getters* et *setters* permettant d'accéder librement au nom, à la date, aux heures de début et de fin afin de les afficher ou de les modifier. La fonction globale de surcharge de l'opérateur *operator<<()* de la classe *ostream* permet d'afficher aisément un rendez-vous sur un flux de sortie.

3.1.6. Classe LDCR

On définit la classe *LDCR* qui gère les listes doublement chaînées de rendez-vous. On commence par définir en ami la classe *RDV*. Puis, on indique les constructeurs par défaut et de recopie. Afin de supprimer aisément une liste de rendez-vous, on précise un destructeur qui détruit cette liste en la parcourant entièrement. On définit ensuite des méthodes sur les surcharges d'opérateurs d'égalité et de différence afin de pouvoir éviter des doublons parmi

les rendez-vous de la liste. En outre, on définit la méthode d'affectation pour pouvoir modifier le rendez-vous courant à partir d'un rendez-vous en paramètres. De surcroît, on définit également la surcharge de l'opérateur *operator[]()* pour pouvoir aisément parcourir la liste des rendez-vous. Puis, grâce à la méthode *afficher()* on affiche une liste de rendez-vous. Ensuite, avec la méthode *rechD()*, on peut parcourir de façon optimale grâce à la recherche dichotomique la liste de rendez-vous puis avec l'aide de la méthode *insérer()* on peut également insérer un rendez-vous dans la liste de rendez-vous. On définit une méthode *supprimer()* qui, quant à elle, supprime un rendez-vous de la liste doublement chaînée de pointeurs de rendez-vous ainsi qu'une méthode *size()* qui renvoie la taille de la liste courante. Comme fonction globale, on définit la surcharge de l'opérateur *operator<<()* de la classe *ostream* qui offre la possibilité d'envoyer la liste doublement chaînée courante vers un flux de sortie.

3.1.7. Classe Manager

Dans cette partie, il s'agit de définir la classe *Manager*, qui gère les liens que peuvent avoir les personnes et les rendez-vous. On crée un constructeur par défaut et on initialise quatre constantes qui sont des noms de fichiers. De plus, la fonction annexe *isStringEmpty()* sert à déterminer si une chaîne de caractères ne contient pas que des caractères de contrôle. Dans cette classe, on définit également des méthodes spécifiques. Les méthodes *addPersonne()* et *removePersonne()* permettent respectivement l'ajout et la suppression d'une personne au sein de la base de données. Subséquemment, les méthodes *addRDV()* et *removeRDV()* permettent la même chose à la seule différence qu'elles concernent la base de données contenant les rendez-vous. De surcroît, les deux méthodes *addPersonneToRDV()* et *removePersonneFromRDV()* permettent à l'utilisateur de saisir un nom de rendez-vous suivi des nom et prénom d'une personne pour permettre l'ajout ou la suppression de ce dernier dans ce rendez-vous. Les quatre méthodes *loadPersonne()*, *loadRDV()*, *savePersonne()* et *saveRDV()* permettent le chargement ou la sauvegarde des bases de données correspondantes à partir d'un fichier nommé à partir des constantes qui valent *Personne.carnetRDV* ainsi que *RDV.carnetRDV*. On définit aussi la méthode *changePhone()* qui, comme son nom l'indique, modifie le numéro de téléphone d'une personne donnée en paramètres. Quant à la méthode *changeMail()*, elle modifie de la même manière l'adresse de messagerie électronique de la personne en paramètres. Afin de faciliter l'appel de ces deux méthodes, il est possible grâce à la méthode *changePhoneAndMail()* de modifier consécutivement le numéro de téléphone et l'email de la personne. La méthode *changeDateAndHour()* permet de modifier la date et le créneau horaire d'un rendez-vous

donné en paramètres. De plus, on initialise les méthodes dites *getters* afin de permettre l'accès aux deux bases de données.

3.1.8. Classe LoadingDialog

La classe *LoadgindDialog* permet d'afficher l'état ainsi que l'avancement lors du chargement d'un fichier de Personne ou de Rendez-vous vers la base de données. Le constructeur permet de récupérer la fenêtre principale mais également le chemin pour chercher le fichier à charger.

3.1.9. Classe ManagingDialog

Gérer la base de données, c'est l'objectif de cette classe. En effet, elle permet d'ajouter ou de retirer des Personnes ou des Rendez-vous.

3.1.10. Classe MainWindow

Cette classe est la la classe de la fenêtre principale de l'interface graphique de notre application. C'est cette classe qui gère les différents affichages lorsque l'utilisateur interagit avec cette dernière. On y définit premièrement des fonctions annexes, notamment *hideOrShow()* qui permet l'affichage ou non de certains éléments de l'interface graphique. Ensuite, on définit des constantes et des variables privées facilitant la compréhension et l'implémentation de certaines fonctions. Puis, pour initialiser certains attributs de la classe, on utilise des méthodes privées précédées du mot clé *setup* qui rend explicite la fonctionnalité réelle de ces méthodes. D'autres ont le mot clé *show* qui gèrent certains affichages et d'autres encore qui gèrent la mise à jour de l'affichage, précédées du mot clé *update*. De plus on déclare les méthodes *loadFile()* et *saveFile()* qui permettent respectivement de charger un fichier de Personne ou de Rendez-vous vers la base de données, ou de sauvegarder les bases de données vers des fichiers. Les méthodes dites *getters* et *setters* facilitent l'accessibilité de certains attributs privés de cette classe par d'autres classes. Enfin, les slots privés sont des méthodes réceptrices recevant des signaux émis par les différents objets de Qt.

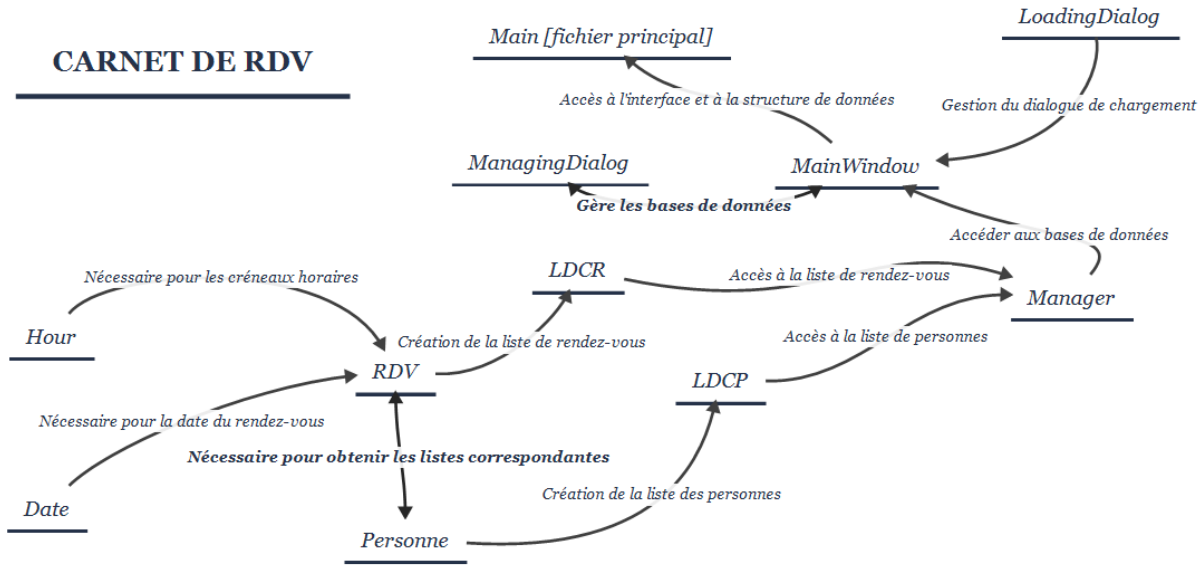
3.2. Le Main

La fonction principale, *main()*, située dans le fichier main.cpp, est la fonction principale exécutée lors du lancement du programme. Elle permet de créer l'application, à rediriger les flux de sortie standard et d'erreurs vers un fichier .log nommé de la date et l'heure au lancement du programme, à traduire l'application dans la langue choisie par l'utilisateur

(uniquement dans la langue courante du système pour le moment et seulement pour les objets pré-faits par Qt, une traduction complète sera à réaliser) et enfin à exécuter l'application.

3.3. Structure globale

Le programme est structuré de façon très simpliste. Au lancement de l'application, le programme exécute tout d'abord la fonction principale *main()*. Ensuite, il crée l'application et la fenêtre principale puis le manager qui se chargera de gérer la base de données et les rendez-vous. Le manager contient donc les différentes bases de données, à savoir LDCP pour les Personnes et LDCR pour les Rendez-vous. Ces deux bases de données contiennent donc respectivement les différentes Personnes et les différents Rendez-vous. Puis chaque rendez-vous est doté d'une date, d'un horaire de début et un de fin. Enfin, les classes LoadingDialog et ManagingDialog permettent la gestion de la base de données depuis l'interface graphique. Référez-vous au diagramme suivant pour plus de clarté.



4. INTERFACE GRAPHIQUE

4.1. Barre des menus

La barre des menus de notre application, située en haut de l'interface initiale, est constituée de trois menus, respectivement nommés “Fichier”, "Éditer" et “Affichage” qui offre une modification et une lecture interactive du carnet de rendez-vous. A l'intérieur de chaque menu, on trouve des boutons. Ces derniers possèdent respectivement un raccourci clavier afin de faciliter encore plus l'interaction avec le carnet à l'utilisateur.

4.1.1. Fichier

Le premier d'entre eux, nommé “Fichier”, est composé de quatre boutons. L'un d'entre eux assure l'ouverture de fichiers externes à l'application pour notamment récupérer, lire et modifier un carnet de rendez-vous déjà existant. C'est un bouton de chargement nommé “Ouvrir un fichier”. Les deux boutons d'en-dessous assurent l'enregistrement des modifications effectuées au cours du lancement de l'application. Le carnet est, soit enregistré si ce dernier existait déjà auparavant, soit l'explorateur de fichiers s'ouvre et, après avoir nommé le fichier courant, il est sauvegardé. Un bouton permet juste l'enregistrement alors que l'autre enregistre et quitte l'application successivement. Quant au quatrième bouton, il permet de tout simplement quitter l'application. En revanche, si l'utilisateur clique sur ce dernier alors qu'il n'a pas enregistré ces modifications, un message informatif s'ouvre en lui proposant d'enregistrer avant de quitter.

4.1.2. Éditer

Le deuxième menu, nommé “Editer”, contient également quatre boutons. D'abord, on trouve deux boutons dit d'ajout. En effet, le premier d'entre eux offre la possibilité d'ajouter autant de personnes que l'on veut dans la base de données courante. Une boîte de dialogue s'ouvre donc afin d'entrer les caractéristiques (nom, prénom, téléphone et mail) d'une personne et, finalement, de l'ajouter. Cette boîte de dialogue permet d'ajouter successivement autant de personnes que l'utilisateur le souhaite ou d'annuler certains ajouts précédemment réalisés au sein du lancement de cette boîte de dialogue. Puis, une fois les ajouts réalisés, l'utilisateur peut soit appuyer sur “Terminer”, ce qui enregistre les nouvelles personnes saisies, soit sur “Annuler”, ce qui, au contraire, n'enregistre aucune modification. Le bouton “Ajouter un RDV”, dans le menu “Editer”, permet la même chose que l'ajout de personnes décrites au-dessus, avec les mêmes avantages à la différence qu'ici, pour ajouter un rendez-vous, il suffit de renseigner la date et le créneau horaire sans saisir de participants. En outre, les deux boutons suivants respectivement appelés “Retirer une Personne” et “Retirer

un RDV", permettent la suppression de personnes ou de rendez-vous existants. Une boîte de dialogue est ouverte et, pour supprimer des personnes, il suffit d'entrer le nom et prénom de ces dernières. Quant aux rendez-vous, il suffit d'entrer le nom de ces derniers supposés uniques. Une fois les sélections faites, il est possible à l'utilisateur de confirmer son choix avec le bouton 'Terminer" ou, au contraire, de l'annuler avec le bouton "Annuler". Ce menu offre donc une manipulation, à travers l'ajout et la suppression de personnes et de rendez-vous, très intuitive pour un utilisateur quelconque.

4.1.3. Affichage

Le troisième, et dernier, menu contient deux boutons respectivement nommés "Afficher la liste de Personnes" et "Afficher la liste de RDV". Le premier offre la possibilité d'afficher, comme son nom l'indique, la base de données contenant toutes les personnes enregistrées jusqu'à là. Le second permet l'affichage de l'autre base de données, à savoir celle contenant les rendez-vous. Lors de l'exécution de l'application, l'affichage est, par défaut, composé des deux bases de données (visible en bas de l'interface). Il est donc possible de visionner uniquement une des deux et de revenir à cet affichage initial en activant tout simplement les deux boutons. Cette activation est visible de par la couleur de fond bleu des icônes correspondants aux deux bases de données.

4.2. Boutons principaux

Il y a également six boutons principaux permettant des actions particulières notamment dans la modification de données ou encore dans l'affichage. Ces derniers sont situés sous la barre de menus.

4.2.1. Recherche de RDV par Date

Ce bouton, qui est situé le plus à gauche parmi l'ensemble des boutons principaux, permet d'effectuer une recherche parmi tous les rendez-vous existants dans la base de données à partir d'une date que l'utilisateur fourni par le biais de la fenêtre qui s'affiche après la sélection du bouton. Une fois la date saisie et valide, les rendez-vous correspondant à la recherche s'affichent. Toutefois, si aucun rendez-vous ne correspond à la recherche, alors un message renseignant cela s'affiche dans la fenêtre. En outre, si la date saisie est invalide, c'est-à-dire qu'elle ne correspond pas au format attendu, alors un message d'erreur apparaît à l'écran.

4.2.2. Liste de RDV d'une Personne

Ce bouton, comme l'indique son intitulé, permet l'affichage de l'ensemble des rendez-vous auxquels la personne donnée participe. Pour cela, l'utilisateur, une fois le bouton pressé et la fenêtre correspondante ouverte, renseigne le nom et prénom de la personne voulue. Une fois le bouton "Rechercher" cliqué, la liste de rendez-vous correspondant apparaît dans la fenêtre. Toutefois, si le nom et/ou prénom d'une personne ne correspondant à aucune personne présente dans la base de données courante, une message d'erreur apparaît signalant le problème. En outre, si les données de la personne sont valides mais qu'aucun rendez-vous ne correspond, alors un message disant que la personne sélectionnée ne participe à aucun rendez-vous s'affiche.

4.2.3. Liste de Personne d'un RDV

Le bouton "Liste de Personne d'un RDV" permet l'affichage de toutes les personnes participant au rendez-vous donné. Pour cela, l'utilisateur, après avoir appuyé sur le bouton correspondant, voit apparaître une fenêtre. Cette dernière permet la saisie du nom du rendez-vous dont on veut voir les membres. Une fois cette saisie faite et valide, la liste des personnes apparaît à l'écran. Toutefois, si le nom saisi ne correspond à aucun des rendez-vous de la base de données courante, alors un message d'erreur apparaît. Pour rappel, il ne peut y avoir de rendez-vous ne contenant aucun ou un seul participant.

4.2.4. Modifier une Personne

Ce bouton permet d'ouvrir la fenêtre qui offre à l'utilisateur la possibilité de modifier le téléphone ou l'adresse mail d'une personne dont le nom et prénom est saisi. Ceci dit, il n'est pas possible de modifier le nom ou prénom de ce dernier si ce n'est en le supprimant puis en le recréant. En outre, si l'utilisateur veut changer le numéro de téléphone uniquement, il n'a pas besoin de renseigner l'adresse mail de ce dernier pour que la modification soit valide et vice versa. Si la personne dont le nom et prénom a été saisi ne correspond à aucune personne de la liste de la base de données courante, alors un message d'erreur est affiché invitant l'utilisateur à corriger sa saisie. En revanche, si tout se passe bien, un message de confirmation apparaît.

4.2.5. Modifier un RDV

Le bouton "Modifier un RDV" permet, quant à lui, de modifier un rendez-vous existant. Ceci dit, tout comme la modification d'une personne, il est impossible de modifier le nom du

rendez-vous si ce n'est en le supprimant puis en le créant à nouveau. Sinon, il est possible, grâce à une fenêtre s'affichant lors du clic sur le bouton, de modifier la date et/ou le créneau horaire d'un rendez-vous. Tout comme la modification d'une personne, l'utilisateur n'est pas contraint de renseigner toutes les caractéristiques du rendez-vous. En effet, seul le nom et les données modifiées sont demandées, par conséquent si l'utilisateur ne souhaite pas modifier, par exemple, l'heure de début du rendez-vous, il ne la saisit tout simplement pas. De la même manière que le paragraphe précédent, si le nom de correspond pas à un rendez-vous existant dans la base de données courante alors un message d'erreur intervient. En revanche, lors de la modification du créneau horaire, on suppose que l'utilisateur saisit des heures valides, c'est-à-dire que l'heure de début ne sera jamais postérieure à celle de fin. Cependant, on vérifie que le nouveau créneau horaire saisi par l'utilisateur n'interfère pas avec un autre rendez-vous au sein des participants du rendez-vous modifié. Si cela est le cas, alors un message d'erreur s'affiche.

4.2.6. Modifier les participants d'un RDV

En cliquant sur ce bouton, une fenêtre apparaît. Il est alors possible d'ajouter et/ou de supprimer autant de personnes que l'utilisateur le souhaite. Toutefois, la personne sélectionnée doit être dans la base de données courante pour être ajoutée au sein du rendez-vous et la personne sélectionnée doit être dans ce rendez-vous pour être supprimée. Sinon, un message d'erreur intervient. L'utilisateur doit, pour cela, renseigner le nom et prénom de la personne cible afin d'y effectuer ses modifications. Un message de confirmation apparaît si l'opération d'ajout ou de suppression s'est bien passée. Cependant, un rendez-vous en est un si et seulement si il y a au moins deux personnes qui y participent. Cela dit, si l'utilisateur supprime une personne et qu'initialement il n'y en avait que deux, alors ce rendez-vous doit être automatiquement supprimé. Or, malheureusement, notre application ne le permet pas de façon automatique.

5. CONTRIBUTIONS ET REMERCIEMENTS

5.1. Contributions

En ce qui concerne les contributions, voici trois tableaux respectivement les contributions de Alexandre, Manuel et Logan :

5.1.2. BARRERE Manuel

Classe Date	Toutes les méthodes sauf *stoDate()* et *toString()*
Classe Hour	Toutes les méthodes sauf *stoHour()* et *toString()*
Classe LDCP	Uniquement la méthode *rechD()*
Classe LDCR	Uniquement la méthode *rechD()*
Classe Manager	Les méthodes : *addPersonne()*, *addPersonneToRDV()*, *addRDV()*, *changeDateAndHour()*, *changeMail()*, *changePhone()*, *changePhoneAndMail()*, *rechRdvDate()*, *removePersonne()*, *removePersonneFromRDV()*, *removeRDV()*
Class Personne	*addRDV()*, *removeRDV()*
Classe RDV	*addMember()*, *compareTo()*, *estImbriquee()*, *removeMember()*
La rédaction du rapport	
Création du PowerPoint & Diagramme	

5.1.1. JANON Alexandre

Github	Préparation du projet avec Github
Classe Date	*toString()*, *stoDate()*
Classe Hour	*stoHour()*

5. CONTRIBUTIONS ET REMERCIEMENTS

Classe LDCP	Toutes les méthodes sauf *rechD()* et surcharge de l'opérateur [ ]
Classe LDCR	Toutes les méthodes sauf *rechD()* et surcharge de l'opérateur [ ]
Classe Manager	Les méthodes : *isStringEmpty()*, *constructeur*, *loadPersonne()*, *loadRDV()*, *savePersonne()*, *saveRDV()*, *getListPersonnes()*, *getListRDV()*
Classe Personne	Les méthodes : *constructeur*, *surcharges d'opérateurs*, *afficher()*, *compareTo()*, *rdvToQString()*, *rdvToString()*, *toQString()*, *toString()*, *getters*
Classe RDV	Les méthodes : *constructeur*, *surcharge des opérateurs*, *afficher()*, *compareTo()*, *participantsToQString()*, *participantsToString()*, *toQString()*, *toString()*, *getters*, *setters*
Classe MainWindow	Toutes les méthodes
Classe ManagingDialog	Toutes les méthodes
Classe LoadingDialog	Toutes les méthodes
Le fichier main.cpp	L'ensemble des fonctions
La rédaction du rapport	

5.1.3. POMMIER Logan

Classe Date	La méthode *stoDate()*
Classe Hour	La méthode *stoHour()*
Classe LDCP	Surcharge de l'opérateur *operator[]()*
Classe LDCR	Surcharge de l'opérateur *operator[]()*

5. CONTRIBUTIONS ET REMERCIEMENTS

5.2. Remerciements

Pour la bonne réalisation du projet, nous avons utilisé Git. Git est un gestionnaire de projet extrêmement puissant et gratuit, qui nous a permis de travailler de façon efficace, chacun chez soi pour respecter les règles strictes imposées par la situation actuelle, mais également en vérifiant l'avancement du projet dans sa globalité. Nous remercions GitHub d'avoir hébergé gratuitement notre projet sur leur site internet.

Nous voulons remercier OpenClassrooms pour l'aide et les cours proposés gratuitement sur leur site internet, qui nous ont permis d'avancer dans notre projet, notamment sur l'utilisation des différents objets de Qt mais aussi sur l'utilisation de Git et Vim, qui est un éditeur de texte que nous avons utilisé pour Git.

Nous remercions également icones8 pour nous avoir permis de télécharger gratuitement les différentes icônes utilisées dans l'interface graphique de notre application, et ce depuis leur site internet.

Merci à tous les membres de l'équipe, qui se sont investis dans ce projet, qui ont permis de le rendre possible et surtout tel qu'il est aujourd'hui.