31/03/2017

Logiciel de gestion de rendez-vous

Projet de structures de données

Contributeurs du projet :
AGUDO-PEREZ Olivier
GHERARDI Sylvain
HUOT Gaël
CHOLE Nicolas

Professeur référent :
CORDIER Fréderic

Licence informatique 2ème année
à la FST de MULHOUSE

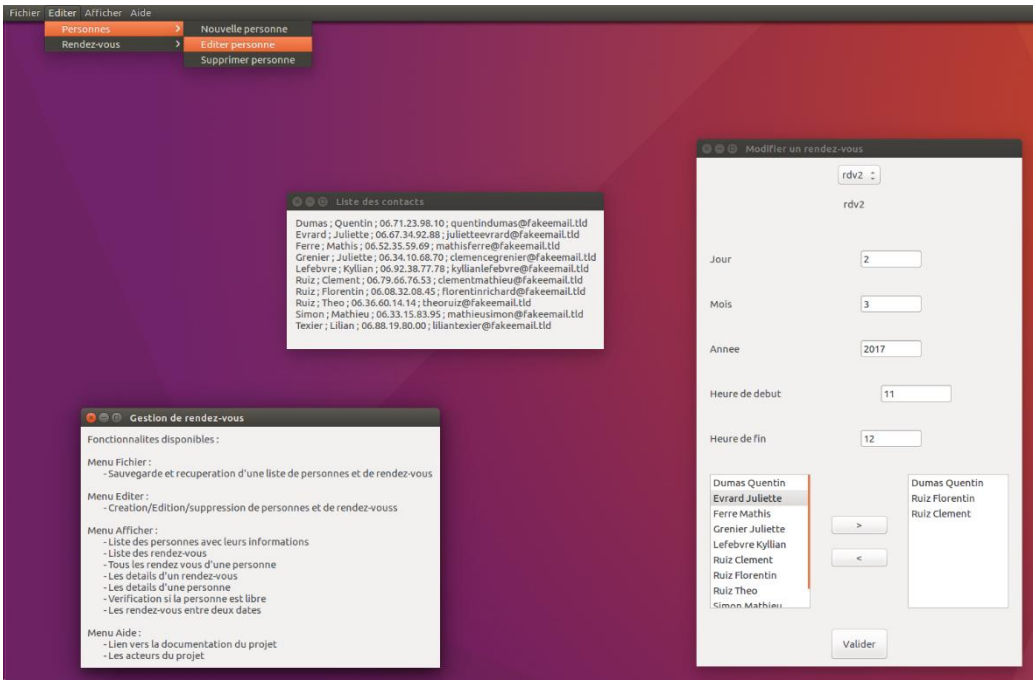
I. Introduction

Dans le cadre de l'unité d'enseignement « structures de données » nous avons réalisé un projet qui a pour but de créer une application permettant à un utilisateur final de gérer un répertoire de personnes et de rendez-vous.

Ce rapport se divise en trois grandes parties :

- La présentation de l'interface
- La présentation de la structure de données
- Les fonctions supplémentaires

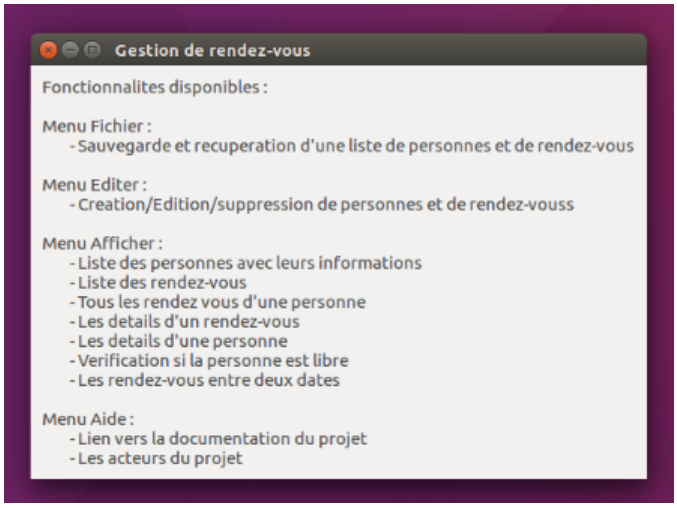
II. Interface utilisateur



L'interface que nous proposons est composée d'une fenêtre principale qui se lance à l'exécution du programme, dans cette fenêtre une barre de menus est disponible et c'est à partir de celle-ci que toutes les fonctions définies dans le cahier des charges sont disponibles.

La présentation de chaque fenêtre et des méthodes associées est réalisée ci-dessous.

A. Fenêtre principale et constructeur par défaut



Fichier : cadre.cpp
Méthode : cadre::cadre() : wxFrame{nullptr, wxID_ANY, "Gestion de rendez-vous", wxDefaultPosition}
Lignes : 20-120

La fenêtre principale est directement créée par le constructeur de la classe cadre, celle-ci contient la barre de menu ainsi qu'une description des fonctionnalités disponibles.

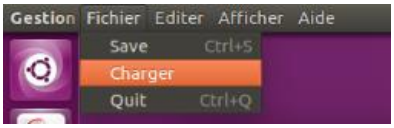
Tous les traitements suivants y sont réalisés :
- construction de la wxFrame principale
- création des menus
- construction de la barre de menus
- affichage du texte
- liaison de chaque menu avec les méthodes correspondantes

B. Constructeur

Fichier : cadre.cpp
Méthode : cadre::cadre(string c_nomFrame) : wxFrame{nullptr, wxID_ANY, c_nomFrame, wxDefaultPosition}
Lignes : 122-127

Ce deuxième constructeur prend en paramètre un string, nous utilisons celui-ci pour construire des nouvelles fenêtres qui se superposent à la fenêtre principale et dans lesquelles nous pouvons mettre différents éléments pour réaliser toutes les fonctions nécessaires.

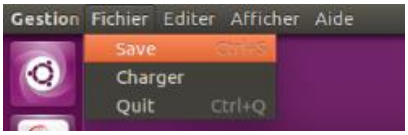
C. Charger



Fichier : cadre.cpp
Méthode : void cadre::OnCharger(wxCommandEvent& e)
Lignes : 122-147

Ce menu permet de charger les répertoires de personnes et de rendez-vous déjà enregistrés dans le dossier du programme. Pour ce faire, la méthode vide complètement les éventuels répertoires en cours d'utilisation, puis fait appel à la classe fromJson qui remplit les deux répertoires avec les informations sauvegardées précédemment dans un fichier json.

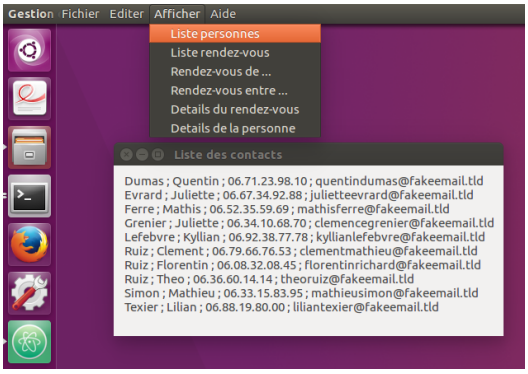
D. Sauvegarder



Fichier : cadre.cpp
Méthode : void cadre::OnSave(wxCommandEvent& e)
Lignes : 149-161

Ce menu permet de sauvegarder les répertoires créés. Pour cela, la méthode fait appel à la classe fromJson qui sauvegarde le contenu dans un fichier json.

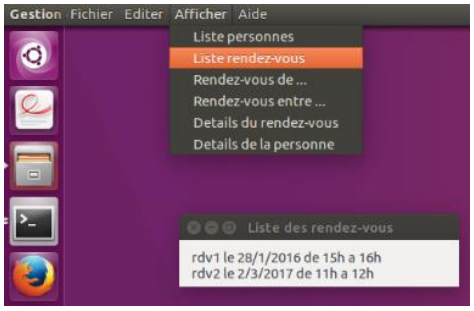
E. Afficher → Liste de personnes



Fichier : cadre.cpp
Méthode : void cadre::OnAfficherPersonnes(wxCommandEvent& e)
Lignes : 172-205

Cette méthode appel le constructeur de la classe cadre avec un attribut de type string, cela permet de créer une nouvelle fenêtre dans laquelle nous affichons ensuite la liste de toutes les personnes avec leurs informations.

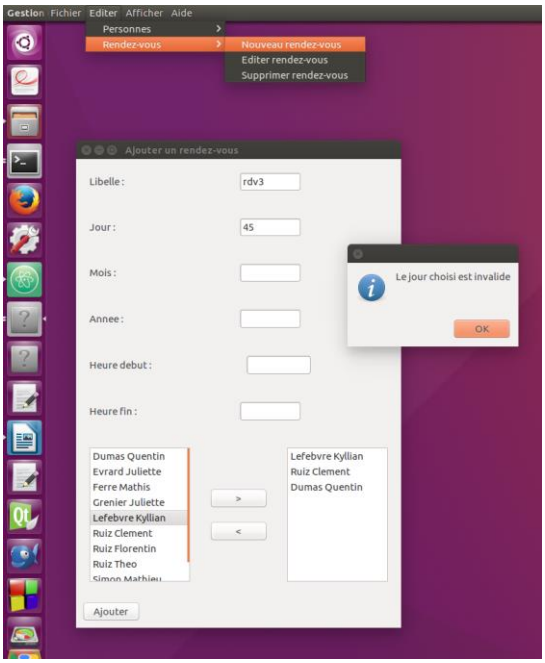
F. Afficher → Liste de rendez-vous

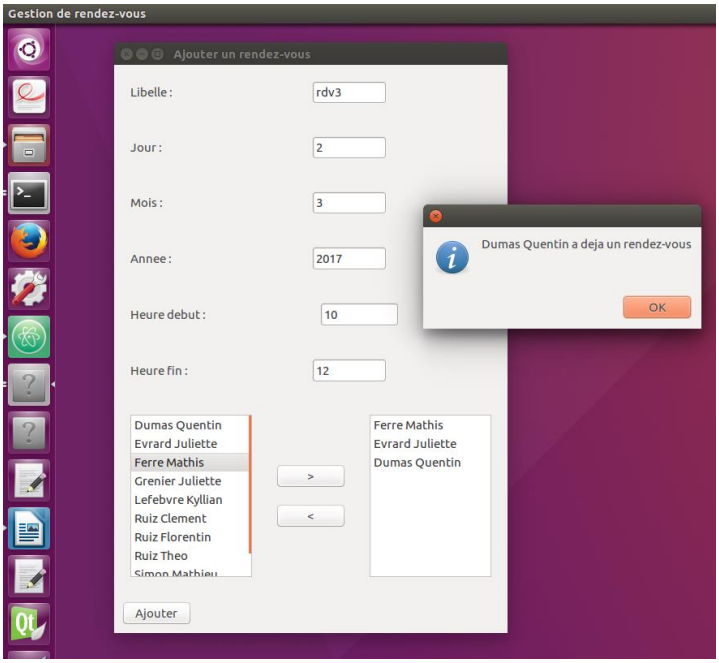


Fichier : cadre.cpp
Méthode : void cadre::OnAfficherRdvs(wxCommandEvent& e)
Lignes : 207-241

La fenêtre est créée de la même façon que pour l'affichage de la liste de personnes, puis nous affichons la liste de tous les rendez-vous.

G. Ajouter un rendez-vous





Fichier : cadre.cpp
Méthodes :
void cadre::OnAjouterRdv(wxCommandEvent& e)
void cadre::OnAjoutListeParticipants(wxCommandEvent& e)
void cadre::OnRetirerListeParticipants(wxCommandEvent& e)
void cadre::OnBoutonAjouterRdv(wxCommandEvent& e)
Lignes : 301-536

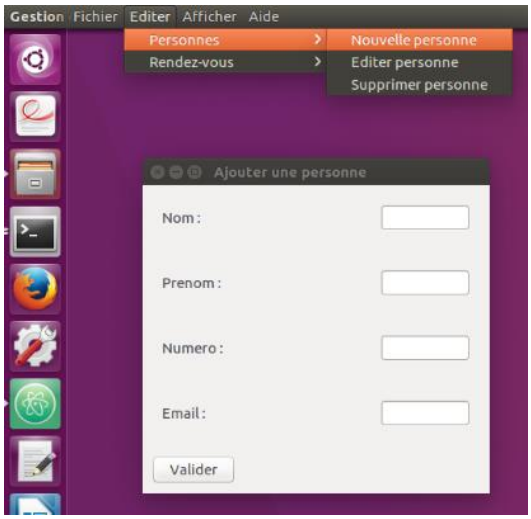
Méthode OnAjouterRdv :
C'est la première méthode qui est appelée via le menu de la fenêtre principale, dans celle-ci nous créons la fenêtre, nous ajoutons les zones de texte, les zones de saisie de texte et créons la liste de choix avec toutes les personnes existantes. Cette liste de choix est créée avec le répertoire de personnes courant et ne peut pas être modifiée.

Méthode OnAjoutListeParticipants :
Lorsque l'utilisateur appuie sur « > » pour ajouter un participant, cette méthode est appelée, elle vérifie tout d'abord qu'une sélection a été réalisée sur la liste de gauche, puis, ajoute la personne sur la liste de droite, si elle n'a pas encore été ajoutée.

Méthode OnRetirerListeParticipants :
Lorsque l'utilisateur appuie sur « < », si une personne est sélectionnée dans la liste de droite, elle en est retirée, aucune vérification n'est réalisée à ce moment.

Méthode OnBoutonAjouterRdv :
Lorsque l'utilisateur appuie sur « Ajouter », cette méthode réalise plusieurs choses :
- Elle vérifie que le rendez-vous n'existe pas déjà.
- Elle vérifie que les informations entrées par l'utilisateur : jour, mois, année et heure sont cohérents, dans le cas contraire un message d'erreur est affiché, de même si une personne dans la liste de participants a déjà un rendez-vous à la date choisie.
- Elle crée un vecteur contenant les noms et prénoms des participants à partir de la liste créée par l'utilisateur.
- Elle appelle la méthode de LCRdv permettant de créer un nouveau rendez-vous.

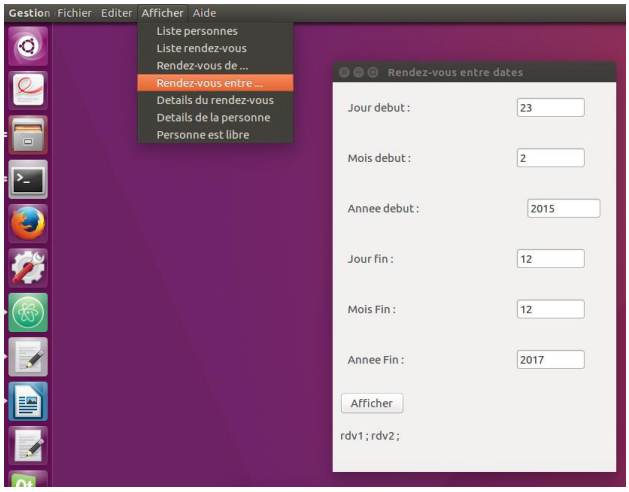
H. Ajouter une personne



Fichier : cadre.cpp
Méthodes :
void cadre::OnAjouterPersonne(wxCommandEvent& e)
void cadre::OnBoutonAjouterPersonne(wxCommandEvent& e)
Lignes : 243-319

L'ajout d'une personne est réalisé comme l'ajout d'un rendez-vous, seulement ici, il n'y a pas de liste de participants à créer, et nous avons choisi de ne pas réaliser de tests sur les données entrées par l'utilisateur.

I. Affichage des rendez-vous entre deux dates



Fichier : cadre.cpp
Méthodes :

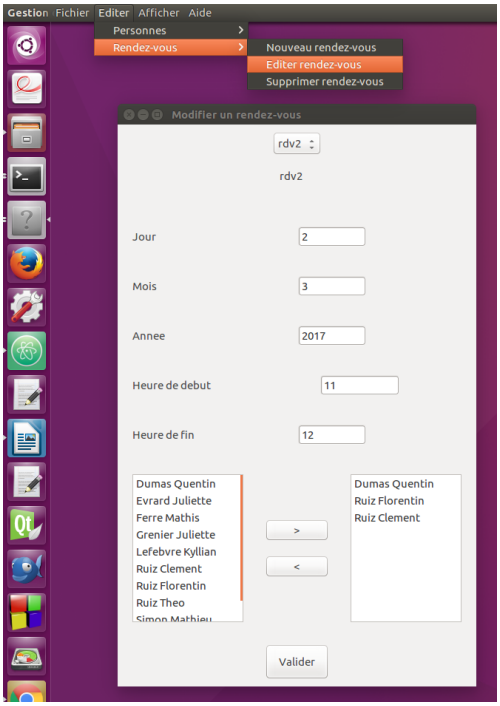
void cadre::OnAfficherEntreDates(wxCommandEvent& e)
void cadre::OnRefreshAfficherEntreDates(wxCommandEvent& e)

Lignes : 538-657

La méthode OnAfficherEntreDates crée la fenêtre, les zones de textes, les zones de saisies et le bouton de validation.

Ensuite, la méthode OnRefreshAfficherEntreDates fait appel à la méthode afficherEntreDates de la classe LCRdv qui lui retourne la liste des rendez-vous en question. Puis elle modifie la valeur du texte en dessous du bouton « Valider » pour afficher la liste de rendez-vous.

J. Modification d'un rendez-vous



Fichier : cadre.cpp
Méthodes :

void cadre::OnModiferRdv(wxCommandEvent& e)
void cadre::OnAjoutListeParticipants(wxCommandEvent& e)
void cadre::OnRetirerListeParticipants(wxCommandEvent& e)
void cadre::OnSelectionModifierRdv(wxCommandEvent& e)
void cadre::OnValiderModifierRdv(wxCommandEvent& e)

Lignes : 801-1028 et 427-458

Ici on réutilise les mêmes éléments que précédemment, la nouveauté est la liste déroulante.

Méthode OnModiferRdv :
C'est dans cette méthode que la liste déroulante est créée.

Méthode OnSelectionModifierRdv :
Lorsque l'utilisateur sélectionne un rendez-vous cette méthode est appelée et met à jour les valeurs de texte dans la fenêtre pour afficher celles du rendez-vous en question.

Méthode OnValiderModifierRdv :
Lors de l'appui du bouton « valider » cette méthode appelle les méthodes modifierDate, modifierHeure et modifierListePersonnes de la classe LCRdv. Tout comme dans l'ajout d'un nouveau rendez-vous, on vérifie si toutes les informations entrées sont correctes et que les nouveaux participants n'ont pas de rendez-vous à la date choisie.

Comme dans « G) Ajout d'un nouveau rendez-vous » la sélection de la liste de personne se fait via une double liste de sélection, on réutilise par ailleurs les méthodes OnAjoutListeParticipants et OnRetirerListeParticipants.

K. Modification d'une personne

Fichier : cadre.cpp
Méthodes :

void cadre::OnModifierPersonne(wxCommandEvent& e)
void cadre::OnSelectionModifierPersonne(wxCommandEvent& e)
void cadre::OnBoutonModifierPersonne(wxCommandEvent& e)

Lignes : 659-738

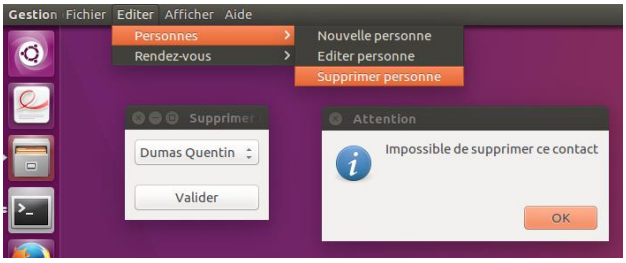
Dans cette fenêtre l'utilisateur sélectionne la personne à modifier dans une liste déroulante.

La méthode OnModifierPesonne crée la fenêtre avec la liste déroulante et des valeurs par défaut pour les champs de texte.

Ensuite, lorsque l'utilisateur sélectionne une personne, la méthode OnSelectionModifierPersonne met à jour les valeurs de texte pour afficher ceux de la personne en question.

Lors de l'appui du bouton « valider » la méthode OnBoutonModifierPersonne appelle les méthodes modifierNumero eet modifierEmail de la classe LCPersonne

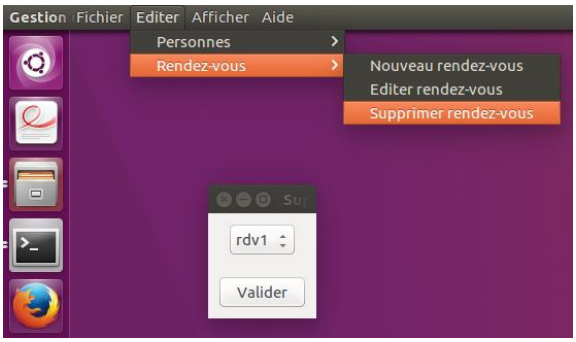
L. Suppression d'une personne



Fichier : cadre.cpp
Méthodes :
void cadre::OnSupprimerPersonne(wxCommandEvent& e)
void cadre::OnBoutonSupprimerPersonne(wxCommandEvent& e)
Lignes : 740-799

Ici on utilise de nouveau la liste déroulante de personnes, lorsque l'utilisateur clique sur « valider » la méthode OnBoutonSupprimerPersonne fait appel à la méthode supprimer de la classe LCPersonne qui supprime la personne uniquement si elle n'a pas de rendez-vous. De plus la méthode supprimée renvoie un booléen qui permet de savoir si la personne a été supprimée et donc d'afficher un message d'erreur si besoin.

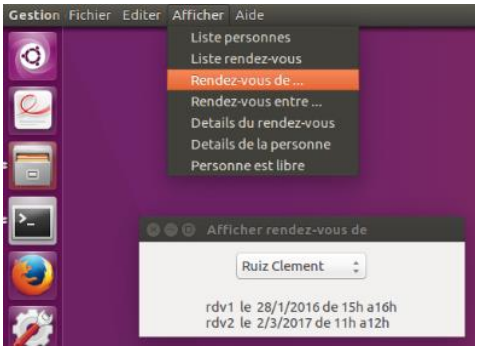
M. Suppression d'un rendez-vous



Fichier : cadre.cpp
Méthodes :
void cadre::OnSupprimerRdv(wxCommandEvent& e)
void cadre::OnValiderSupprimerRdv(wxCommandEvent& e)
Lignes : 1112-1162

Le même principe que pour la suppression d'une personne est utilisé, mais cette fois-ci aucun test n'est réalisé avant la suppression.

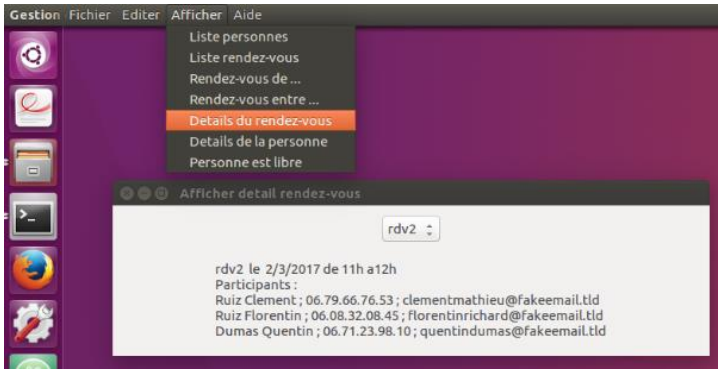
N. Affichage des rendez-vous d'une personne



Fichier : cadre.cpp
Méthodes :
void cadre::OnRdvDe(wxCommandEvent& e)
void cadre::OnSelectionRdvDe(wxCommandEvent& e)
Lignes : 1030-1109

La méthode OnRdvDe permet à l'utilisateur de sélectionner une personne, puis la méthode OnSelectionRdvDe trouve et affiche tous les rendez-vous correspondants.

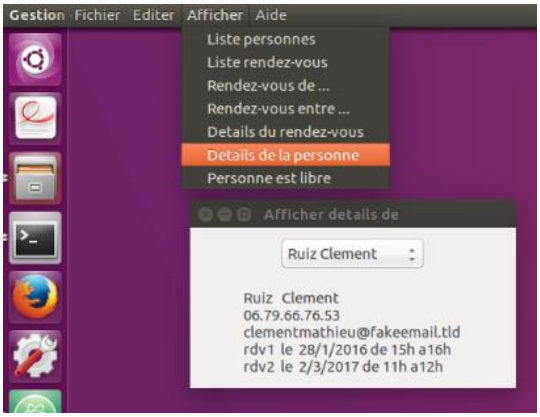
O. Affichage des détails d'un rendez-vous



Fichier : cadre.cpp
Méthodes :
void cadre::OnDetailRdv(wxCommandEvent& e)
void cadre::OnSelectionDetailRdv(wxCommandEvent& e)
Lignes : 1415-1501

La méthode OnDetailRdv permet à l'utilisateur de sélectionner un rendez-vous, puis la méthode OnSelectionDetailRdv trouve et affiche toutes les informations du rendez-vous ainsi que des informations de tous les participants.

P. Affichage des détails d'une personne



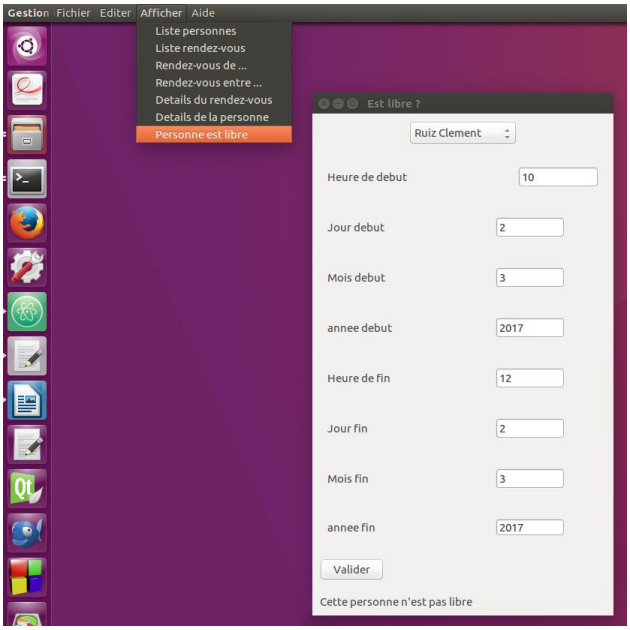
Fichier : cadre.cpp
Méthodes :

void cadre::OnDetailsPersonne(wxCommandEvent& e)
void cadre::OnSelectionDetailsPersonne(wxCommandEvent& e)

Lignes : 1164-1252

La méthode OnDetailsPersonne permet à l'utilisateur de sélectionner une personne, puis la méthode OnSelectionDetailsPersonne trouve et affiche toutes les informations de la personne ainsi que des informations des rendez-vous auxquels elle participe.

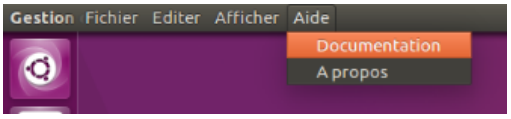
Q. Déterminer si une personne est libre



Fichier : cadre.cpp
Méthodes :
void cadre::OnPersonneEstLibre(wxCommandEvent& e)
void cadre::OnSelectionPersonneEstLibre(wxCommandEvent& e)
Lignes : 1254-1413

La méthode OnPersonneEstLibre permet à l'utilisateur de sélectionner la personne et de choisir une date de début et de fin. Ensuite la méthode OnSelectionPersonneEstLibre fait appel à la classe LCRdv pour déterminer si la personne est libre.

R. Documentation



Fichier : cadre.cpp
Méthodes :
void cadre::OnDocumentation(wxCommandEvent& e)
Lignes : 1503-1512

Cette méthode permet d'ouvrir la documentation créée avec Doxygen dans un navigateur web.

S. Initialisation de l'interface

La classe « appli » est nécessaire à wxWidget, c'est elle qui est appelée par la bibliothèque au lancement du programme. Dans la méthode OnInit nous créons donc le cadre principal, et nous l'affichons.

III. Structure de données

La structure de donnée contient toutes les méthodes permettant de réaliser les fonctions demandées dans le cahier des charges. Celle-ci est indépendante de l'interface graphique, ce qui permet de modifier les deux structures indépendamment. La description détaillée de la structure de données interne est réalisée ci-dessous.

A. Liste chaînée de personnes

Fichiers :

lcpersonne.h / lcpersonne.cpp
chainonpersonne.h / chainonpersonne.cpp

La structure permettant de réaliser la liste chaînée de personnes es indépendante du reste de la structure de donnée.

Dans la classe LCPersonne les méthodes suivantes sont disponibles :
- LCPersonne() : le constructeur
- ~LCPersonne() : le destructeur
- void ajouter(string nom, string prenom, string numero, string email) : ajoute une nouvelle personne à la liste

L'ajout à la liste chaînée est différent de la structure classique car elle fait appel à la méthode de tri qui renvoie l'adresse du chaînon après lequel le nouveau chaînon doit être ajouté. Après cela, le parcours de la liste et l'insertion se fait de façon classique.

- bool supprimer(string nom, string prenom, LCRdv* listeRdv) : supprime une personne de la liste
- void modifierNumero(string nom, string prenom, string numero) : modifie le numéro d'une personne
- void modifierEmail(string nom, string prenom, string email) : modifie l'email d'une personne
- chainonPersonne* getTete()
  string getNom(chainonPersonne* crt)
  string getPrenom(chainonPersonne* crt)
  string getNumero(chainonPersonne* crt)
  stringgetEmail(chainonPersonne* crt)
  chainonPersonne* getSuivant(chainonPersonne* crt)
  Les accesseurs de tous les paramètres d'un chainonPersonne.
- chainonPersonne* trier(string& nouveauNom, string& nouveauPrenom) const : cette méthode prend en paramètre le nom de la personne à ajouter et recherche la place que doit prendre cette personne dans la liste chaînée existante.

Nous avons réalisé la méthode de tri sans utiliser les opérateurs « < » et « > » pour comparer l'ordre alphabétique des deux chaînes de caractère, notre méthode fonctionne donc de la manière suivante :

1) Si la liste chaînée existante est vide, on renvoie le pointeur nul
2) On parcourt la liste chaînée et pour chaque personne on fait :
    a) On compare les strings caractère par caractère
        - Si le nouveau caractère est plus grand, on passe au mot suivant.
        - S'il est plus petit, on a trouvé la place du nouveau nom.
        - S'il est égal, on passe à la lettre suivante.
    b) Si les deux noms ont les mêmes lettres, mais pas la même taille, on place le plus petit en premier.
    c) Si les deux personnes ont le même nom, on refait les étapes a et b sur les prénoms.
    d) Si on a atteint le bout de la liste, on peut placer le nom.

A. Liste chaînée de rendez-vous

Fichiers :
    lcrdv.h / lcrdv.cpp
    chainonrdv.h / chainonrdv.cpp

La structure de données permettant de réaliser la liste chaînée de rendez-vous est composée des classes Lcrdv et chainonRdv, contrairement à la liste chaînée de personnes, celle-ci n'est pas indépendante.

Pour mettre en avant les relations entre les deux listes chaînées les attributs et les méthodes de la classe avec leur explication est disponible ci-dessous :

- Pour réaliser la liste de participants de chaque rendez-vous nous utilisons un vecteur à deux dimensions qui contient les noms et prénoms de tous les participants. Nos méthodes dans la liste chaînée de rendez-vous prennent en paramètre les strings des noms et prénoms et recherchent elles même le chaînon correspondant, cela permet de faciliter l'appel vers les méthodes de la liste de personnes.

- void modifierListePersonnes(string libelle, vector<vector<string>> participants)

Cette méthode remplace simplement la liste de participants du rendez-vous par la liste passée en paramètre. Comme pour la liste de personnes, la méthode prend en paramètre un string contenant le libellé du rendez-vous et recherche elle-même le chaînon correspondant.

Nous avons choisi de remplacer la liste complète à chaque fois et non pas de pouvoir ajouter et supprimer des personnes car notre interface permet d'ajouter ou de supprimer des personnes de la liste et la renvoie. Voir point II/G) et II/J).

- vector<string> afficherEntreDates(int jour1, int mois1, int annee1, int jour2, int mois2, int annee2)

Cette méthode crée et renvoie un vecteur contenant le libellé de tous les rendez-vous programmés entre la date 1, incluse, et la date 2, incluse.

Pour vérifier si le rendez-vous est compris entre deux dates, deux booléen sont créés, dans un premier temps on vérifie si la date du rendez-vous est supérieure à la date 1, puis on vérifie si cette même date de rendez-vous est inférieure à la date 2. Enfin, si les deux booléens ont pris la valeur vraie, on peut ajouter le libellé du rendez-vous au vecteur et passer au rendez-vous suivant.

- bool estLibre(string nom, string prenom, int jour1, int mois1, int annee1, int heure1, int jour2, int mois2, int annee2, int heure2)

Cette méthode renvoie un booléen qui indique si une personne est libre entre la date 1 et la date 2 passées en paramètre.

Elle fonctionne sur le même principe que la méthode précédente mais celle-ci prend aussi en compte les heures.

Le but en réutilisant cette technique est de pouvoir tester si une personne est libre sur une période plus longue qu'une seule journée.

- bool avoirRdv(string nom, string prenom)

Cette méthode vérifie si une personne a un rendez-vous et renvoie un booléen. Elle prend en paramètre deux strings, cela permet de faire une recherche dans les vecteurs de participants des rendez-vous sans devoir faire appel à la liste chainée de personnes.

- chainonRdv* trier(string& nouveauLibelle) const

Cette méthode fonctionne de la même façon que la méthode de tri de la liste de personnes (voir III/A/) mais avec un seul mot au lieu de deux.

IV. Fonctions supplémentaires

- Test si date valide

Fichier : cadre.cpp
Méthodes :
void cadre::OnBoutonAjouterRdv(wxCommandEvent& e)
void cadre::OnRefreshAfficherEntreDates(wxCommandEvent& e)
Lignes : 486-534 et 623-655

Tous les tests sur les valeurs entrées par l'utilisateur sont réalisés dans l'interface, cela permet de ne pas faire appel à la structure de données si cela n'est pas nécessaire.

La vérification sur la date se fait de la façon suivante :

1) L'utilisateur entre la date voulue dans les zones de texte prévues à cet effet.
2) La valeur récupérée est un string, nous utilisons donc la fonction wxAtoi() qui transforme un string de la bibliothèque wxwidget en integer. Si le texte entré n'est pas transformable en integer, Atoi() renvoie 0.
3) On vérifie si la valeur n'est pas 0 et si elle est comprise entre 2 valeurs, par exemple 1 et 12 pour le mois.
4) On envoie un message d'erreur si une des valeurs n'est pas valide.

Cette méthode a un inconvénient, il n'est pas possible de créer un rendez-vous en l'an 0. Nous avons cependant choisi de garder ce défaut, ce qui permet de simplifier le code et de ne pas réaliser de traitements inutiles.

- Tri sur prénom et sans comparateur < >

Fichier : lcpesonne.cpp
Méthodes :
chainonPersonne* LCPersonne::trier(string& nouveauNom, string& nouveauPrenom) const
Lignes : 61-136

Le fonctionnement de cette méthode a été expliqué au point III/ A/.

- Enregistrer/charger depuis des fichiers

Fichiers : fromJson.cpp
fromJson.h

Méthodes :
void fromJson::getRepertoire()
void fromJson::saveRepertoire()
void fromJson::getRdv()
void fromJson::saveRdv()

Le répertoire de personnes ainsi que le répertoire de rendez-vous sont sauvegardés dans des fichiers distincts. Pour ce faire, nous avons utilisé la librairie Open Source Json (https://github.com/nlohmann/json) qui permet :

1) De créer des objets json et de les rediriger dans des flux.
2) De récupérer du json depuis un flux et d'en créer un objet.

La classe fromJson est divisée en 4 fonctions : 2 qui permettent de sauvegarder les listes chainées et de les stocker dans leurs fichiers json respectifs et deux qui permettent de récupérer des données JSON afin de les placer dans les listes chainées passées en paramètre lors de l'appel de la classe. Avant toute sauvegarde et toute récupération, une vérification est effectuée afin de vérifier si le fichier utilisé existe ou est utilisable.

Pour la sauvegarde, il était nécessaire de créer un objet JSON temporaire, qui fonctionne comme un tableau associatif, où l'on stocke les différentes informations des rendez-vous ou des personnes. Pour cela, on parcourt la liste chainée correspondante afin d'y récupérer les informations qui y sont stockées. On envoie ensuite les objets JSON dans le fichier grâce à la surcharge d'opérateur « << ».

Enfin, pour la récupération des données, on récupère l'objet JSON du fichier grâce à la surcharge d'opérateur. On parcourt ensuite l'objet JSON et l'on ajoute les entrées dans la liste chainée grâce aux fonctions « ajouter ».

- Interface graphique

L'interface graphique est présentée dans le point II/.

V. Organisation du projet

Pour nous organiser dans ce projet nous avons utilisé un gestionnaire de version, Git. Nous avons commencé à utiliser Git pour ce projet. Cependant c'était le premier projet relativement important en groupe que nous avons réalisé, nous n'avons donc pas forcément utilisé Git de la manière la plus optimisée.

Voici le lien de notre gitHub :

https://github.com/Agudolive/Projet_RDV

Voici la répartition du travail au sein de notre groupe :

Olivier :

Classe LCPersonne :

- void LCRdv::modifierDate
- void LCRdv::modifierHeure
- void LCRdv::modifierListePersonnes
- bool LCRdv::estLibre

Classe LCRdv :

- void LCPersonne::modifierNumero
- void LCPersonne::modifierEmail

Classe cadre :

- void cadre::OnAjouterPersonne / void cadre::OnBoutonAjouterPersonne
- void cadre::OnModifierPersonne / void cadre::OnSelectionModifierPersonne / void cadre::OnBoutonModifierPersonne
- void cadre::OnSupprimerPersonne / void cadre::OnBoutonSupprimerPersonne
- void cadre::OnRdvDe / void cadre::OnSelectionRdvDe
- void cadre::OnSupprimerRdv / void cadre::OnValiderSupprimerRdv
- void cadre::OnPersonneEstLibre / void cadre::OnSelectionPersonneEstLibre
- void cadre::OnDetailsPersonne / void cadre::OnSelectionDetailsPersonne

Sylvain :

Classe LCPersonne :

- LCRdv::~LCRdv
- chainonRdv* LCRdv::trier
- vector<string> LCRdv::afficherEntreDates
- bool LCRdv::avoirRdv

Classe LCRdv :

- LCPersonne::~LCPersonne
- chainonPersonne* LCPersonne::trier

Classe cadre :

- void cadre::OnCharger
- void cadre::OnSave
- void cadre::OnAfficherRdvs
- void cadre::OnAjouterRdv / void cadre::OnAjoutListeParticipants / void cadre::OnRetirerListeParticipants / void cadre::OnBoutonAjouterRdv
- void cadre::OnAfficherEntreDates / void cadre::OnRefreshAfficherEntreDates
- void cadre::OnDocumentation(wxCommandEvent& e)
- void cadre::OnDetailRdv(wxCommandEvent& e) / void cadre::OnSelectionDetailRdv(wxCommandEvent& e)

Olivier + Sylvain :

Classe LCPersonne :

- void LCRdv::ajouter
- LCRdv::supprimer

Classe LCRdv :

- void LCPersonne::ajouter
- bool LCPersonne::supprimer

Classe chainonRdv :

- chainonRdv::chainonRdv

Classe chainonPersonne :

- chainonPersonne::chainonPersonne

Classe cadre :

- cadre::cadre()
- void cadre::OnAfficherPersonnes
- void cadre::OnModiferRdv / void cadre::OnSelectionModifierRdv / void cadre::OnValiderModifierRdv

Classe appli :

- bool appli::OnInit()

Gaël :

Classe fromJson :

- void fromJson::getRepertoire
- void fromJson::saveRepertoire
- void fromJson::getRdv
- void fromJson::saveRdv

VI. Conclusion

Pour conclure, toutes les fonctionnalités demandées dans le cahier des charges ont été réalisées et nous avons pu implémenter des fonctionnalités supplémentaires. Tout cela a été réalisé en respectant la deadline.

Pendant le projet nous avons dû apprendre à collaborer dans le cadre d'un projet de programmation, ce qui était nouveau pour nous.

Nous avons dû nous adapter lorsqu'un membre de notre groupe a abandonné la formation.

La principale difficulté que nous avons rencontrée se situe au niveau de l'interface, en effet avant cela nous n'avions pas l'habitude de gérer plusieurs fenêtres pour le même programme.