Rapport de projet
Sujet : logiciel de gestion d'un Aéroport
Analyse Orientée Objet

Auteurs :

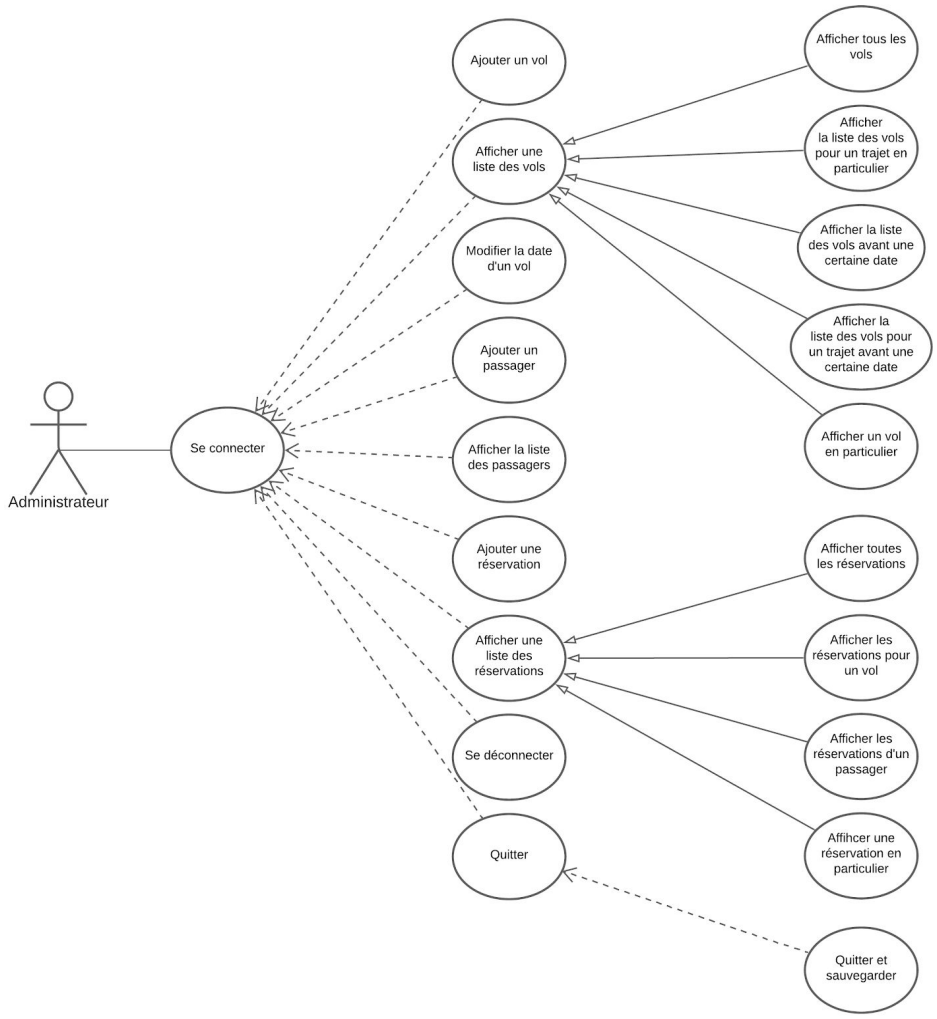
- **Tom Boumba**
- **Antoine Zaug**

Sommaire:

I. Fonctionnalités du logiciel

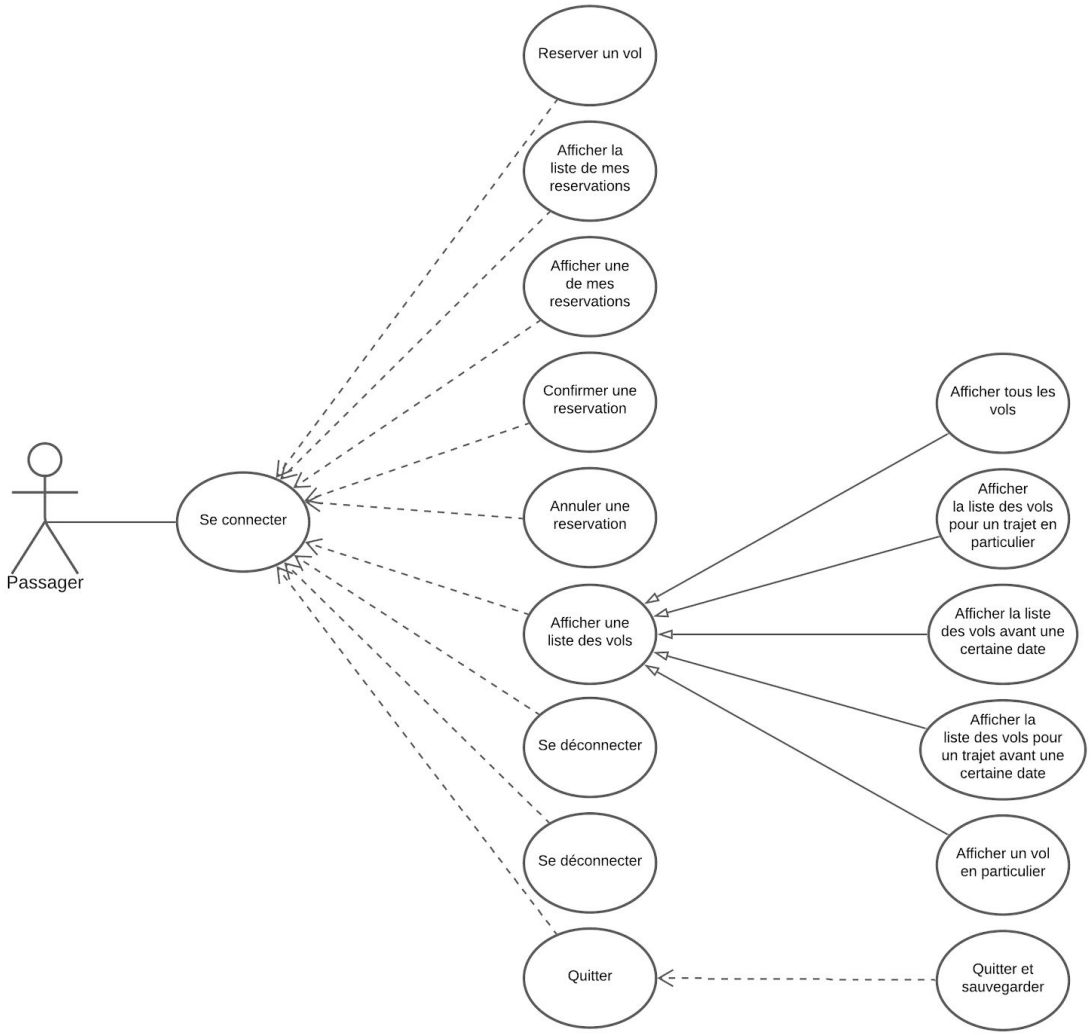
Le logiciel conçu et implémenté permet la gestion des vols et des réservations d'un aéroport. Il y a une partie "back office" qui permet l'administration, à l'aide de laquelle un administrateur peut gérer les vols et les réservations, et une partie "front office", permettant à un passager (client d'une compagnie aérienne) de gérer ses réservations de vol.
Pour être identifié en temps qu'administrateur, il faut rentrer son identifiant ainsi que son mot de passe. Évidemment s'ils sont corrects alors nous nous connectons sinon il y a un message d'erreur (les administrateurs sont ajoutés en dehors de l'exécution du logiciel, à la main dans le fichier texte "Administrateurs.txt").
Le principe est le même pour un passager mais on demande plutôt son numéro de passeport (les passagers sont ajoutés par les administrateurs).

Voici ci-dessous le diagramme de cas d'utilisation côté administrateur :



(Cette image est disponible dans l'archive)

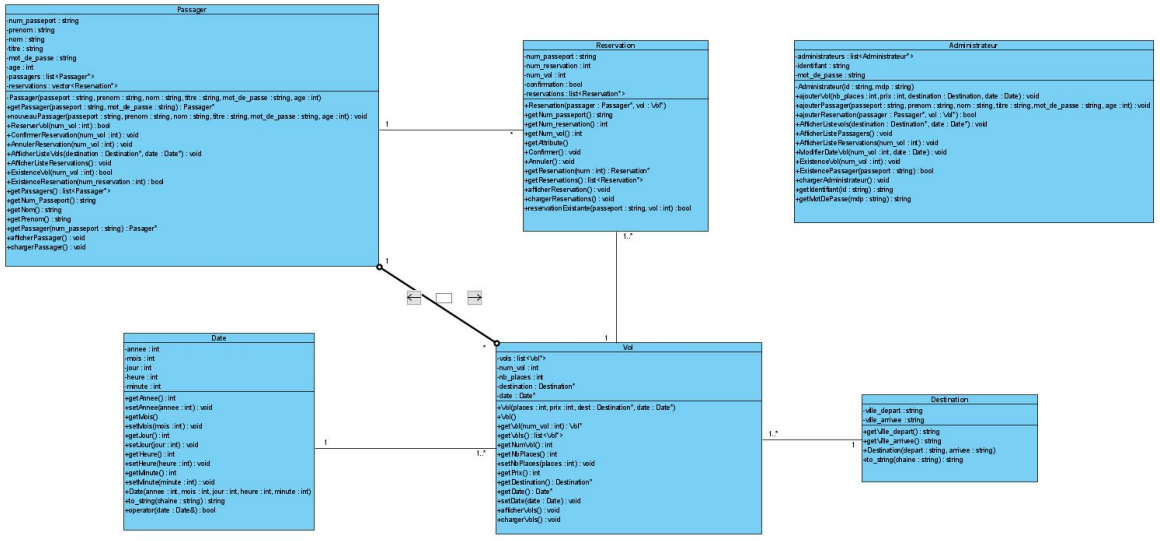
Et le diagramme de cas d’utilisation côté utilisateur :



(Cette image est disponible dans l’archive)

II. Structure du modèle orienté objet

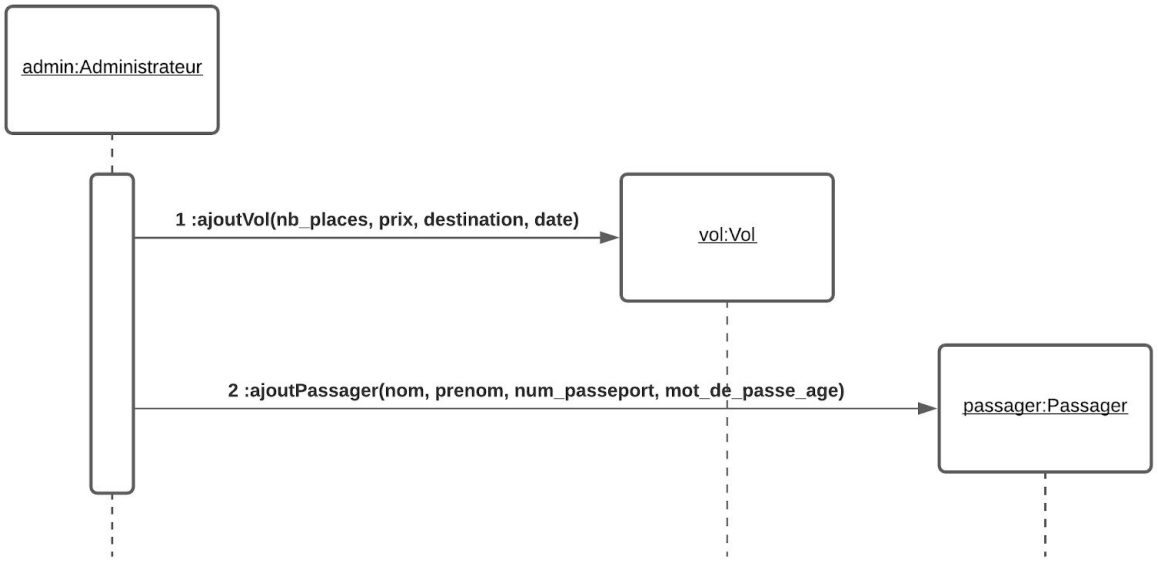
Voici ci-dessous le diagramme de classe de la solution orientée objet conçue.



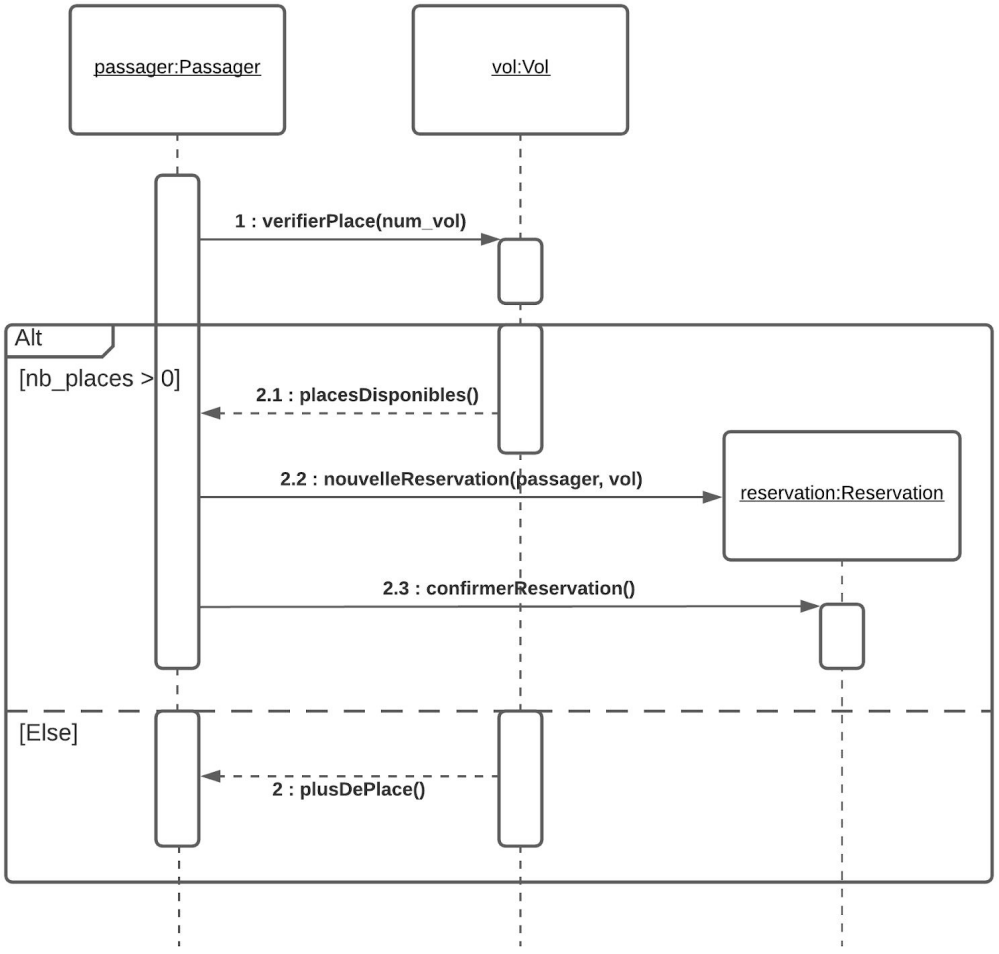
(Cette image est disponible dans l'archive)

La classe "Administrateur" n'a pas de relation avec les autres classes car un administrateur ne fait que de la modification de données, il ne réserve pas de vols.

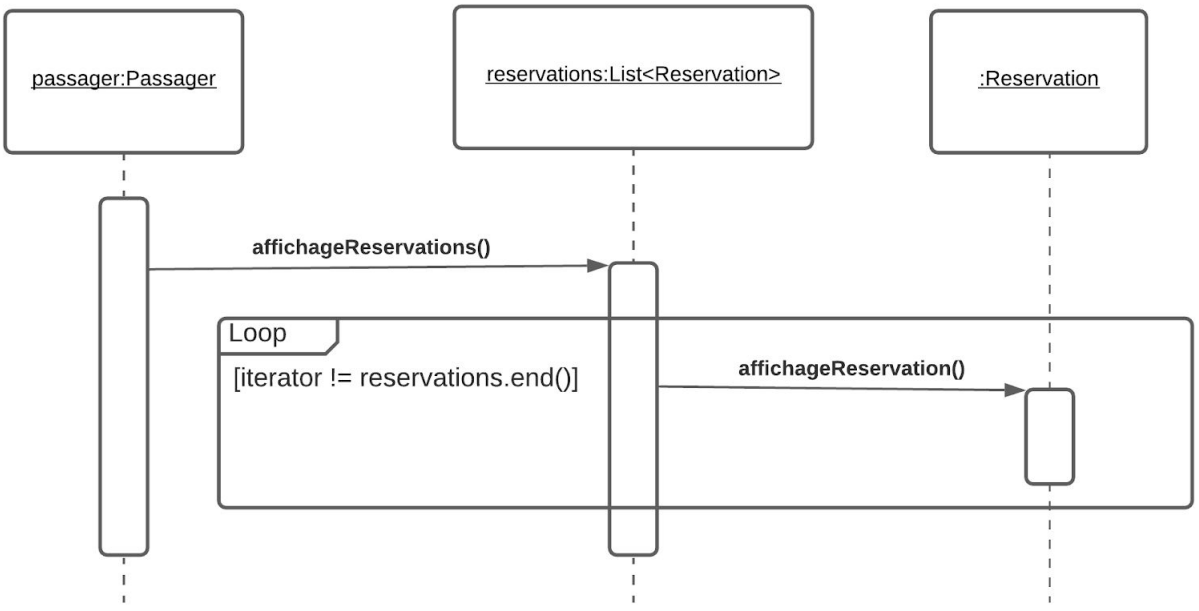
Il existe des schémas typiques d'interaction entre les instances de ces classes, voici ci-dessous ces interactions illustrées par des diagrammes de séquences.



Ajout d'un vol et d'un passager par un administrateur



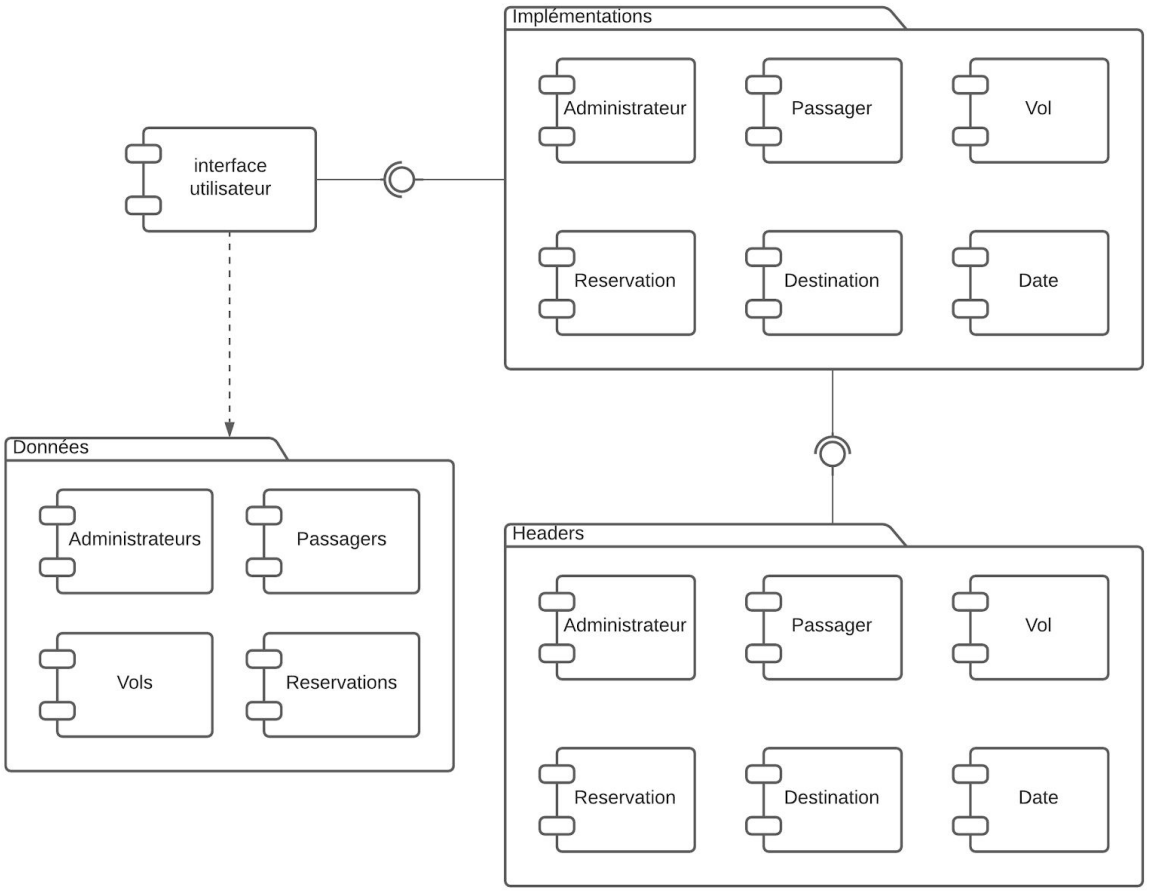
Réservation d'un vol par un passager, puis confirmation de cette réservation.



Affichage par un passager de ses réservations de vol.

III. Détails d'implémentation

Voici ci-dessous un diagramme de composants esquissant la structure de l'implémentation qui se trouve dans le répertoire "src".



(image disponible dans l'archive)

L'interface utilisateur proposant les fonctionnalités du logiciel est entièrement écrite dans le fichier "main.cpp".

Au début du programme, les données sont chargées depuis le répertoire "donnees" vers les listes statiques des classes du projet, puis pour fonctionner, le code fait appel à des fonctions implémentées dans les fichiers ".cpp" du répertoire "Implementations" (les headers se trouvent dans le répertoire "Headers").

Lorsqu'on quitte le programme en sauvegardant, les données contenues dans les listes statiques des classes sont transcrites en texte (des lignes de champs séparés par le délimiteur ":"). On peut également se déconnecter pour utiliser la partie client et administrateur sans quitter le logiciel.

Les saisies utilisateur sont traitées à l'aide de la fonction "saisieChamp<T*>(string message, char flag)". Cette fonction permet de demander à l'utilisateur de saisir la prochaine entrée avec le "message", puis effectuer une vérification sur sa saisie selon le "flag" fourni. Un pointeur vers un objet "T" contenant la saisie de l'utilisateur est ensuite renvoyé. Cela permet de traiter aussi bien les saisies de chiffres que de caractères, chaînes, etc…
La saisie des dates et des destinations est gérée à part dans deux fonctions dédiées. Ces fonctions captent l'information saisie par l'utilisateur et fabriquent un objet puis renvoient un pointeur vers l'objet (date ou destination). La saisie de date gère le fait que la date saisie doit être inférieure à la date actuelle.

Les dates sont d'ailleurs comparables entre elles, ce qui est utile pour afficher les vols ayant lieu avant une certaine date.

Nous avons choisi de ne pas sauvegarder les dates et les destinations en tant que telles, mais plutôt en tant que partie intégrante des vols dans lesquelles elles sont présentes. De cette façon, on peut quand même retrouver des informations selon la date et la destination des vols en se référant aux objets "Destination" et "Date" internes aux vols.

Nous avons également choisi d'empêcher la création d'administrateurs dans le code. Le seul moyen de créer des administrateurs est de les rajouter à la main dans le fichier texte. Les passagers, vols, et réservations ne sont toujours créés que pour être directement stockés dans les listes statiques correspondantes. De la même façon, les destinations et dates sont également créées seulement pour être associées à un vol (on crée quand même aussi une date dans le but de comparer la date saisie par l'utilisateur avec les autres lorsqu'on affiche les vols avec ce tri).

Conclusion:

Pour conclure, ce projet fut très intéressant, tant d'un point de vue technique que d'un point de vue organisationnel. En effet, il a fallu mettre à profit notre enseignement sur le C++ et réfléchir aux meilleurs moyens d'implémenter une solution, mais il a aussi fallu s'organiser, travailler en binôme et mener à bien ce projet. Nous sommes satisfaits du travail fourni, du rendu, et de notre participation à ce projet.