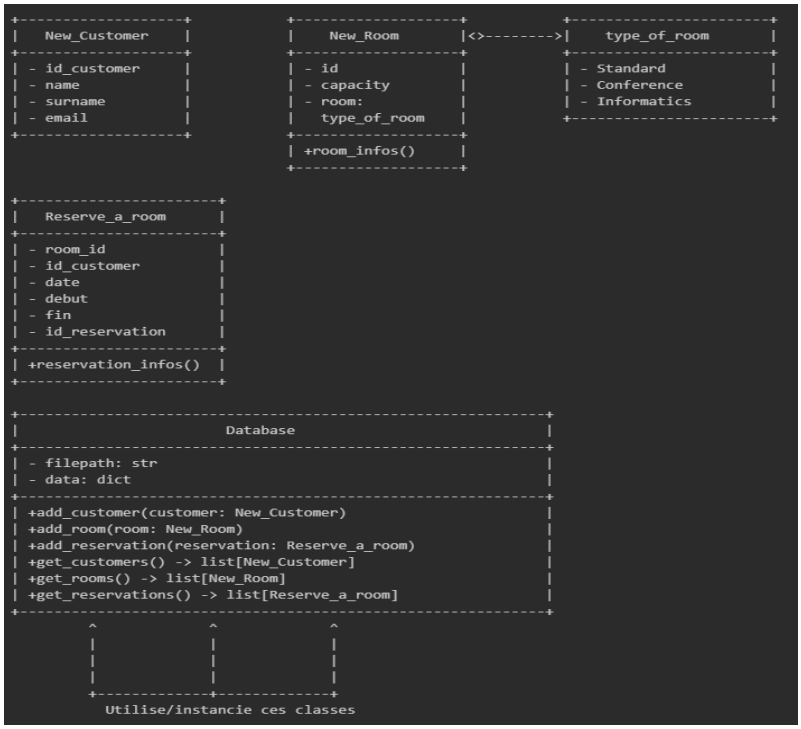
Dorian Bismuth 1A ASE
Ahmad Hassan 1A ASE

Rapport sur le projet MeetingPro App fait en AOOPython :

La démarche que nous avons utilisé fût dans un premier temps de d'organiser les différentes tâches que nous avions à faire afin de mener à bien ce projet. Pour cela, on a restreint le code à deux partie, le back (tous ce qui est data, création de classes et de fonctions), et le front (l'interface). Puis dans le back, afin d'avoir une idée des différentes liaisons que devais avoir chacune des classes, on a créé un diagramme de Classe avec les relation entre chaque classes. Cela à permit d'avoir une idée de comment découper le travail au plus efficacement, et aussi cela nous a donné en même temps des idées des différentes fonctions devant être ainsi créé. On retrouvâmes au finale un diagramme des classes ressemblant à peu près à celui-là :



Une fois cela fait, on a due comprendre et savoir utiliser les différents packages qui nous avait été mis à disposition, puis donc créer 3 fichiers python dans le dossier src comprenant chacune d'entre elles une des trois classes qui fût utile au projet. Ainsi qu'un autre fichier python pour la sauvegarde des datas des utilisateurs, ce qui est concrètement le passages d'informations contenu dans une library, à des informations enregistrées dans le json au format txt. Ceci est stocké dans un autre fichier python du répertoire src nommé « database ». Et ce fichier permettait alors d'enregistrer dans le json les informations récupérer des clients, des salles, et des réservations.
Et une fois chacune de ces étapes faite, on a enfin pu s'attaquer au plus long qui fût le fichier python GUI (Graphical User Interface), qui comme son nom l'indique permettait justement la création de toute la partie front de code, soit son interface.
Une fois tous cela finit, on s'attaqua aux tests fait à l'aide de « pytest ». Dans un répertoire à part nommée tests, on a créé un fichier python pour chaque fichier python contenu dans le src, à l'exception de tests pour GUI car il est inutile de tester si une interface est bien créée ou si elle a bien fait appelle à d'autres fonctions ou classes des autres fichiers. Et dans chacun de ces fichier, on a testé les fonctions et les classes intéressante à être testé afin de vérifier qu'elles renvoyaient bien ce qu'elles étaient censées renvoyer à l'aide de « pytest » que l'on met dans le terminale python.

Ensuite, on a créé le fichier «pyproject.toml» renseignant sur certaines infos du code ainsi que sur nous. Et le fichier«.gitignore» renseignant sur certaine infos de la versions de Python utilisé ainsi que sur certains packages utilisables (ou utilisés). Et enfin le fichier README.md que l'on a complété à la toute fin une fois le code fonctionnel.

Pour ce qui est de la répartition des taches ;

- Dorian Bismuth s'est occupé du GUI, du database, du diagramme de classes ainsi que du README
- Ahmad Hassan s'est occupé du pseudo-code, de la création des classes ainsi que des tests

Afin d'utiliser les différents packages, on s'est principalement inspirer des documentations python trouvable sur le net ainsi que de vidéos Youtube. Et lors de certaines erreurs de chemins de data, ou sur l'interface, on a pu utiliser l'aide de certaines personnes de la classe, ainsi que de d'IA pour les problèmes que l'on arrivait vraiment pas à résoudre.

L'application MeetingPro, accessible en compilant le fichier GUI.py du dossier src (source), permet à l'utilisateur de gérer l'ensemble du processus de réservation de salles de manière intuitive. Dès l'ouverture, l'utilisateur accède à une fenêtre principale organisée en plusieurs onglets : « Home », « Add », « Book » et « Display ». Depuis l'onglet « Home », il peut naviguer rapidement vers les autres fonctionnalités. L'onglet « Add » permet d'ajouter soit un nouveau client, soit une nouvelle salle : pour un client, il suffit de renseigner le nom, le prénom et l'email (qui doit être unique) ; pour une salle, il faut spécifier un identifiant (unique), le type de salle (standard, conférence ou informatique) et la capacité. Toute tentative d'ajout d'un doublon (même email ou même identifiant de salle) génère un message d'erreur.

L'onglet « Book » guide l'utilisateur dans la réservation d'une salle : il sélectionne d'abord un client existant, puis saisit la date, l'horaire de début et de fin, le type de salle et la capacité souhaitée. Après avoir cliqué sur « Search available rooms », l'interface affiche la liste des salles disponibles correspondant aux critères ; l'utilisateur choisit alors une salle et valide la réservation, qui est enregistrée dans la base de données. Si des informations sont manquantes ou erronées, un message d'erreur s'affiche.

L'onglet « Display » permet de visualiser, sous forme de tableaux, la liste des clients, des salles, des réservations et les créneaux horaires disponibles pour chaque salle. L'utilisateur peut rafraîchir l'affichage à tout moment en cliquant à nouveau sur le bouton correspondant. L'interface signale également les erreurs de saisie ou d'informations manquantes par des messages d'alerte. Enfin, la suppression de clients, salles ou réservations doit se faire manuellement dans les fichiers JSON du projet, en respectant la structure du fichier. L'interface est conçue pour être agrandie afin d'afficher correctement tous les éléments, et il est déconseillé de lancer plusieurs fois le fichier GUI.py pour éviter d'ouvrir plusieurs fenêtres en même temps.

Enfin, j'ai rencontré beaucoup de problèmes durant ce projets mais il y en a particulièrement 3 qui fût compliqué à résoudre tel que :

– Pour les test, « pytest », ne fonctionnait sur aucun de nos ordis, nous avons passé du temps à tenter de résoudre le problème, et finalement, en renommant le package pip, on réussi finalement à faire marcher pytest.

– Ensuite, nous avons eu particulièrement du mal à nous adapter au package json car envoyer des listes de caractères aux formats str dans le json était très compliqué mais en plus, il y avait sans arrêt des erreurs car les datas ne trouvait pas le chemin à prendre pour être utilisé, donc on pouvait stocker des datas dans le load, mais on ne savait pas si ils étaient enregistré, et on ne pouvait pas les

utiliser. Finalement, par la modification de certaines variables et par l'ajout d'une fonction save et d'une autre fonction load, on a pu enfin faire marcher notre json et notre gestion d'information de la database. Cependant, changer tout cela à pris du temps car tout le fichier database en fût changé.

– Et enfin un autre gros problème, non pas par la difficulté (même si c'était aussi un peu à cause de la difficulté), mais par la longueur du programme, c'était par rapport au fichier GUI.py. Ce fichier a comporté un énorme nombre d'erreur, mise à part les problèmes de mise en page que l'on a pu rencontré, trouver comment faire certaines choses sur internet (comme des tableau, des menu déroulant, passer d'un page à une autres, les boutons qui deviennent bleu lorsque la souris passe dessus ainsi de suite), un autre gros problème que l'on a rencontré (toujours un peu en rapport avec les databases), c'était le faite que des informations ne s'enregistrait pas sur l'interface, et que certains liens entre des informations à cocher ne marchait pas forcément. Là il n'y avait pas de miracle, on a due écumer les centaines de lignes de codes que l'on avaient fait afin de comprendre que soit on avait mal importé des classes, soit on avait mal utilisé une fonction (c'est aussi comme ça que l'on s'est rendu compte du problème précédent avec le json). Et pour les derniers problèmes que l'on avait pas réussi à regler (alors qu'ils semblaient être les plus simple comme par exemple les données écrit en blanc au lieu d'être écrit en noir), on a alors été aidé de chat gpt afin de savoir d'ou pourrait venir l'erreur, et finalement, on a réussi à s'en débarrasser (sachant qu'il s'agissait encore une fois d'une mauvaise utilisation de fonction).