Développement d'applications avec IHM :

Rendu Final

SOMMAIRE :

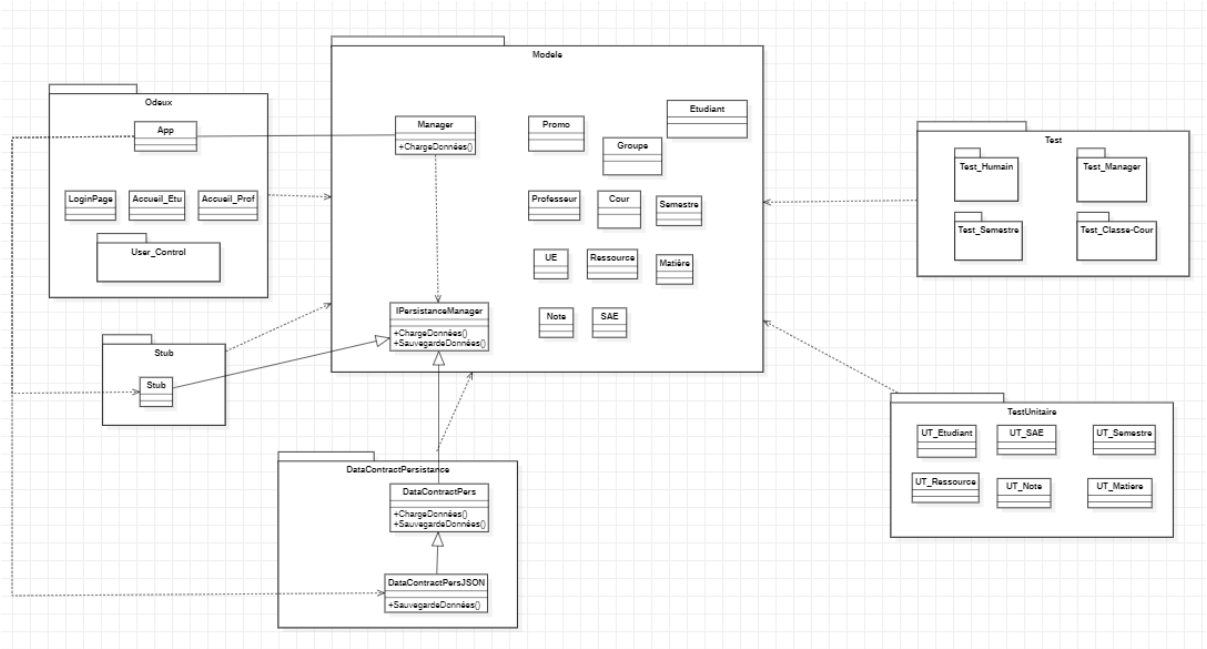
Nom de l'application : Odeux

Thème de l'application : Espace numérique de travail

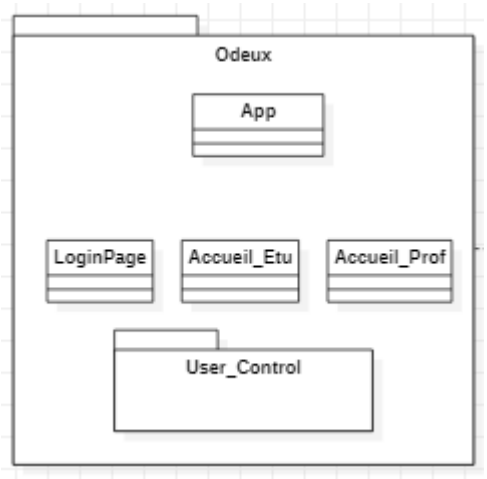
Récapitulation de notre application :

Nous cherchons à réaliser une application qui reprend "Odin", site web interne de l'IUT développé par Cédric Bouhours et son équipe avec de nouvelles options et une page d'accueil bien plus complète et intuitive, où l'on retrouve les salles et horaires du prochain cours, des notes directement affichées, les dernières absences ... Nous cherchons à ce que les utilisateurs (étudiants, professeurs) se sentent bien organisés, en prenant en compte les retours d'expérience de différents élèves sur la conception, pour créer de nouvelles fonctionnalités comme des pop-ups de rappels pour déclarer sa présence, par exemple.

Diagramme de paquetage :



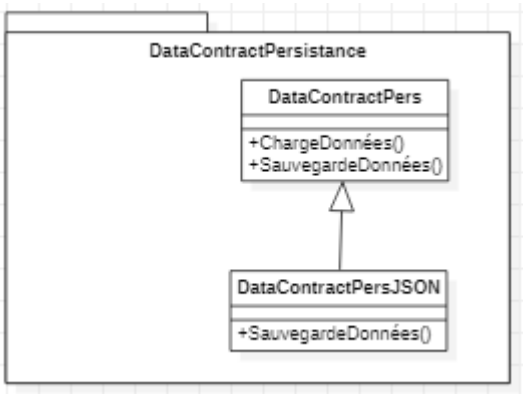
- Description :



Le package **Odeux** est l'ensemble de toutes les vues et de l'outil App du framework DOTNET qui gère l'ensemble des vues, elle possède une dépendance vers Modele, qui fait le lien entre le code XAML et les classes en C# pour permettre la liaison de données (Data Binding) ou pour les ajouts et suppression d'une liste, les événements, navigation…

Le **Manager** va récupérer ses données grâce à la fonction **ChargeDonnées**, qui va récupérer dans le stub les données qui y sont fournies, pour les instancier à la liste des personnes, liste des cours et à la promo actuelle.

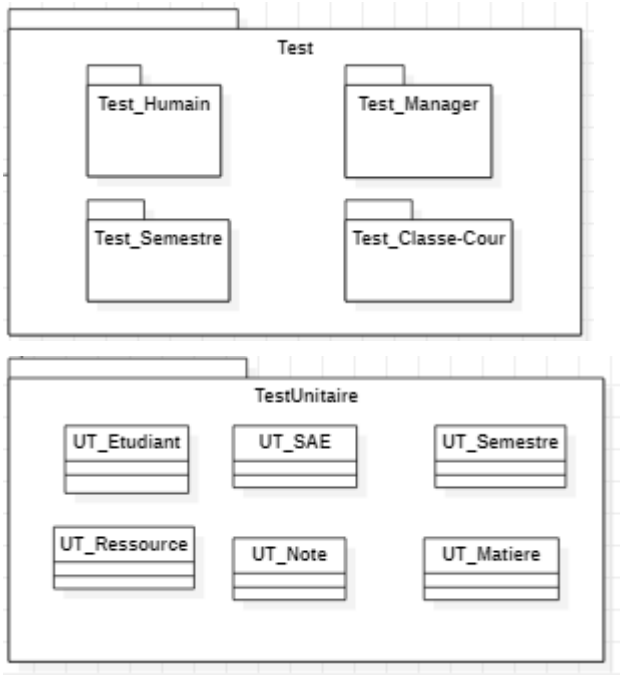
La classe **Stub** et **DataContractPers** vont être des dérivées de la classe IPersistanceManager qui a été codée dans le Modele, et qui vont reprendre les méthodes qu'elle possède.



Quant à lui le package **DataContractPersistance,** qui assure à la pérennité et sauvegarde des données après avoir fermé l'application, possède deux sous-classes essentielles à la persistance des données, elle possède deux méthodes :

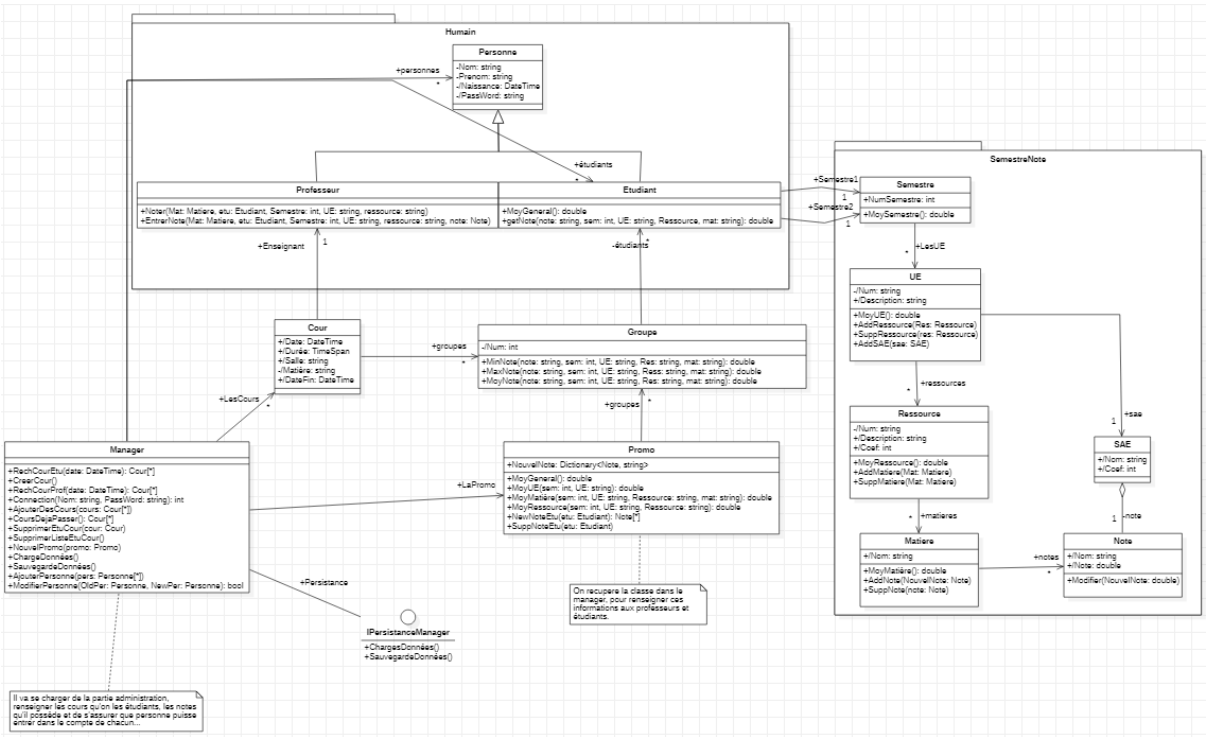
- *Charge Données* : charge les données via un fichier .json dans le manager .
- *Sauvegarde Données* : sauvegarde les données dans un fichier .json .

La classe **DataContractPersJSON** est une classe fille/dérivée de **DataContractPERS**, elle est utilisée pour sauvegarder les données en format ".json" et non ".xml" en reprenant l'opération du **DataContractPERS.**



Le Package **Test** et **TestUnitaire** sont deux packages à part, qui assurent le bon fonctionnement du Modele en testant différents cas possibles sur quasiment toutes ces classes.

Diagramme de classes :



- Description :

Le diagramme de classes est un schéma utilisé pour présenter les classes et les interfaces des systèmes ainsi que leurs relations. Pour le diagramme ci-dessus qui reprend le **Modele** de notre application, on retrouve en classe essentielle le Manager qui va récupérer les informations des autres classes (Liste d'étudiant, Liste de cours…) .

Au total dans ce diagramme nous retrouvons 13 classes, on retrouve tout un côté qu'on nomme **Semestre** qui compose tout les attributs liés à l'étudiant et ses notes personnelles, où chaque Semestre (L'étudiant possède deux Semestre) est composé d'une liste d'UE, toutes les UE se voient accompagnées d'un nom et d'une Description qui permettent de les différencier entre eux, chaque UE de cette liste possède une Liste de ressource et une SAE, chaque ressource possède en plus un coefficient (qui montre plus ou moins l'importance de chaque ressource dans l'UE). Mais aussi chaque Ressource possède une liste de matières et qui chaque matière de cette liste possède une liste de Notes, une note se voit attribuée d'un entier qui est la note que l'étudiant a obtenu dans la matière en question et d'un nom (facultatif). Les moyennes de chacune de ces classes vont parcourir l'ensemble des listes pour additionner toutes les notes et diviser par le nombre de notes trouvées, pour la classe UE la moyenne va prendre en compte le coefficient de chaque ressource.

Les classes UE, Ressource et Matière ont une fonction Add (Add+nom de l'attribut de la liste qu'elle prend) et Supp, qui permet d'ajouter et supprimer à la liste qu'elle possède.

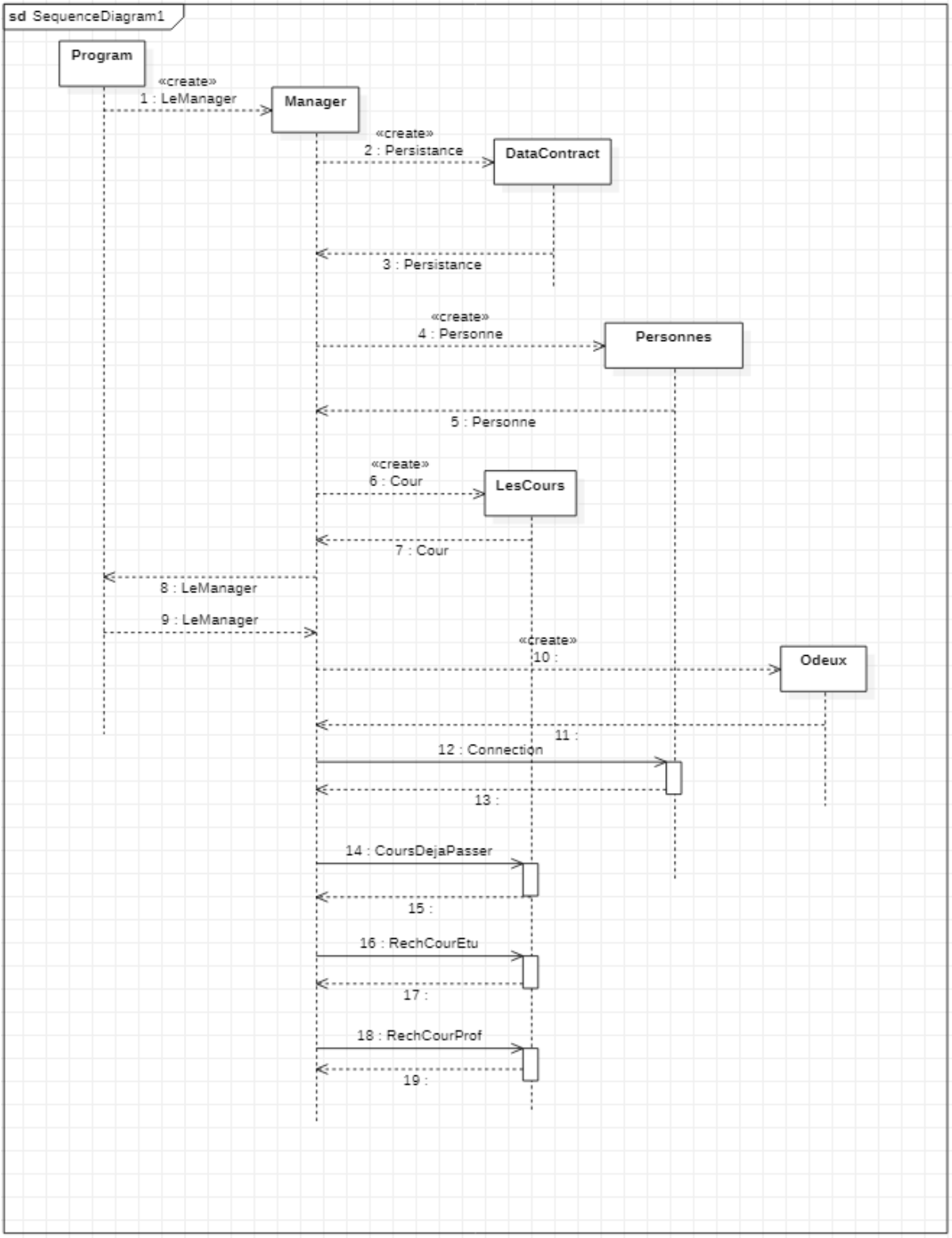
De l'autre côté nous retrouvons l'**Humain** avec la classe Personne qui prend le nom, prénom, date de naissance et mot de passe d'une personne (qui va servir l'authentification de la personne dans l'application et que n'importe qui ne puisse pas avoir accès à ses informations), la classe Etudiant et Professeur est une dérivée de cette classe, le Professeur n'est pas composé de Semestre et a la capacité d'ajouter une note à l'élève dans le semestre ou par exemple l'UE qu'il souhaite grâce à l'opération *Noter*.

Ensuite, nous avons un cours entre un Enseignant et une liste d'étudiant (Groupe) qui a pour spécificité la date de son déroulement, sa durée, la salle où il a lieu et la matière qui va être enseignée. L'attribut DateFin est juste la somme de la date et la durée pour estimer quand le cours sera fini, toutes ces classes vont permettre d'identifier les personnes qui veulent se connecter à l'application en tant que professeur ou étudiant.

La classe Groupe va juste prendre une liste d'étudiants et va être reprise par la classe cour et Promo. La Classe Promo n'est là que pour faire une moyenne des informations de tous les étudiants que ce soit sur leur moyenne générale (Les deux Semestres) ou une UE, Ressource, matière en particulier, qui va être utilisée dans l'application pour informer les avancements des étudiants (les profs et étudiants pourront voir ces informations), elle possède également les qui prend le nom d'un élève pour que les élèves concernés puissent être au courant des nouvelles notes qui leur sont attribuées dans la partie XAML.

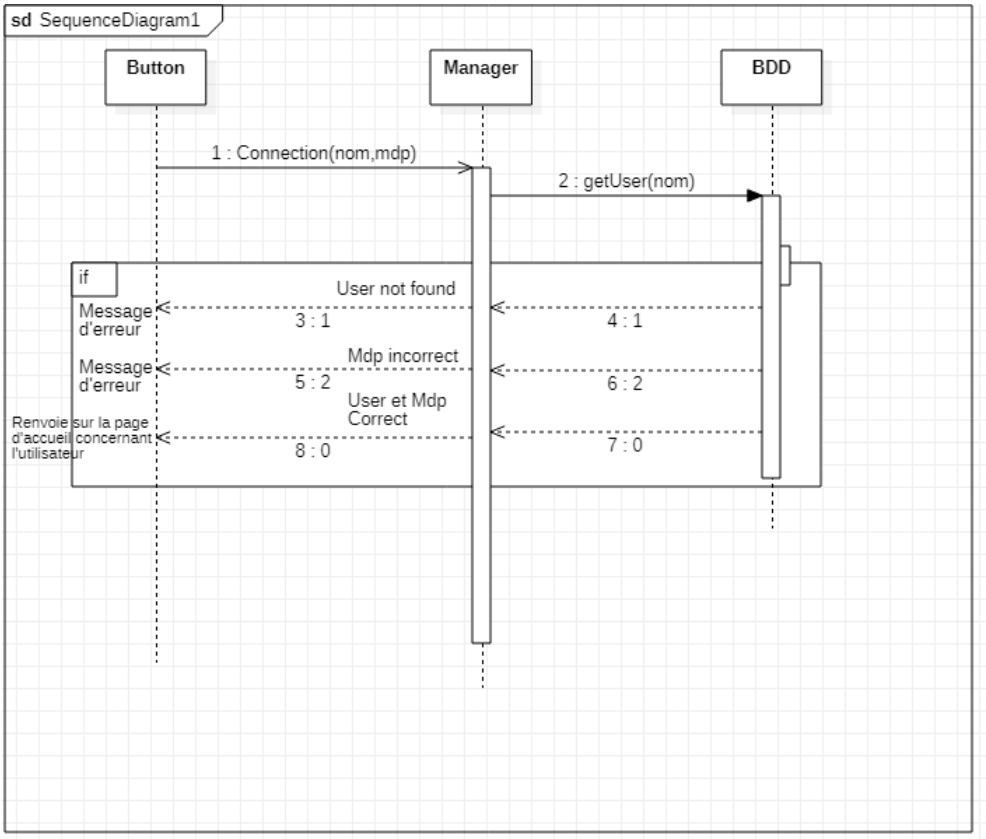
IPersistanceManager est une interface, c'est-à-dire que les classes qui en hériteront devront forcément posséder ses méthodes. Ces sont *SauvegardeDonnées()* et *ChargeDonnées()*, qui sont nécessaires au bon fonctionnement de la persistance.

Diagramme de séquence :



- Description :

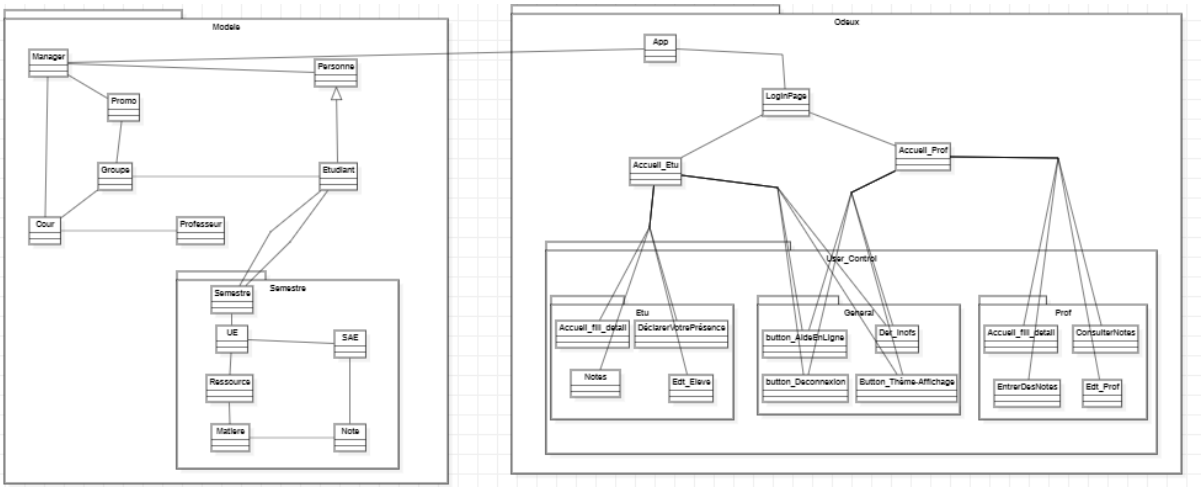
L'**utilité** du **diagramme de séquence** est de décrire comment se déroulent les interactions entre les acteurs ou objets. Pour le diagramme ci-dessus, dans un premier temps le programme va instancier un Manager qui se nommera **LeManager,** ce dernier instancie un Data Contract, une Liste de personne, une Promo et une liste de cours qui ira dans le programme, le manager va connecter une personne à l'application **Odeux** grâce à l'opération *Connection()*. Pour la liste des cours nous avons trois fonctions qui vont permettre à l'utilisateur si c'est un étudiant de savoir quels cours qui il a manqué et dans lesquels il n'a pas déclaré sa présence, puis de connaître pour un étudiant grâce à la fonction *RechCourEtu(DateTime date)* les cours qui ont pour date la valeur entrée en paramètres et de même pour les professeur grâce à la fonction *RechCourProf(DateTime date).*



Pour le diagramme de la fonction login, cela est représenté par 3 grands objets, tout d'abord le button qui est celui qui va faire la requête pour se connecter à l'application en prenant un nom et un mot de passe, puis le Manager qui va vérifier en faisant appel à la base de données si le nom et mot de passe correspondent à un utilisateur et renvoie *1,2 ou 0*, dans le cas où il va renvoyer 1,l'utilisateur n'est pas trouvé et un message d'erreur va s'afficher sur la personne, puis dans le cas où il est trouvé (*0)*, on retrouve la possibilité qu'il soit un professeur ou étudiant, dans le cas

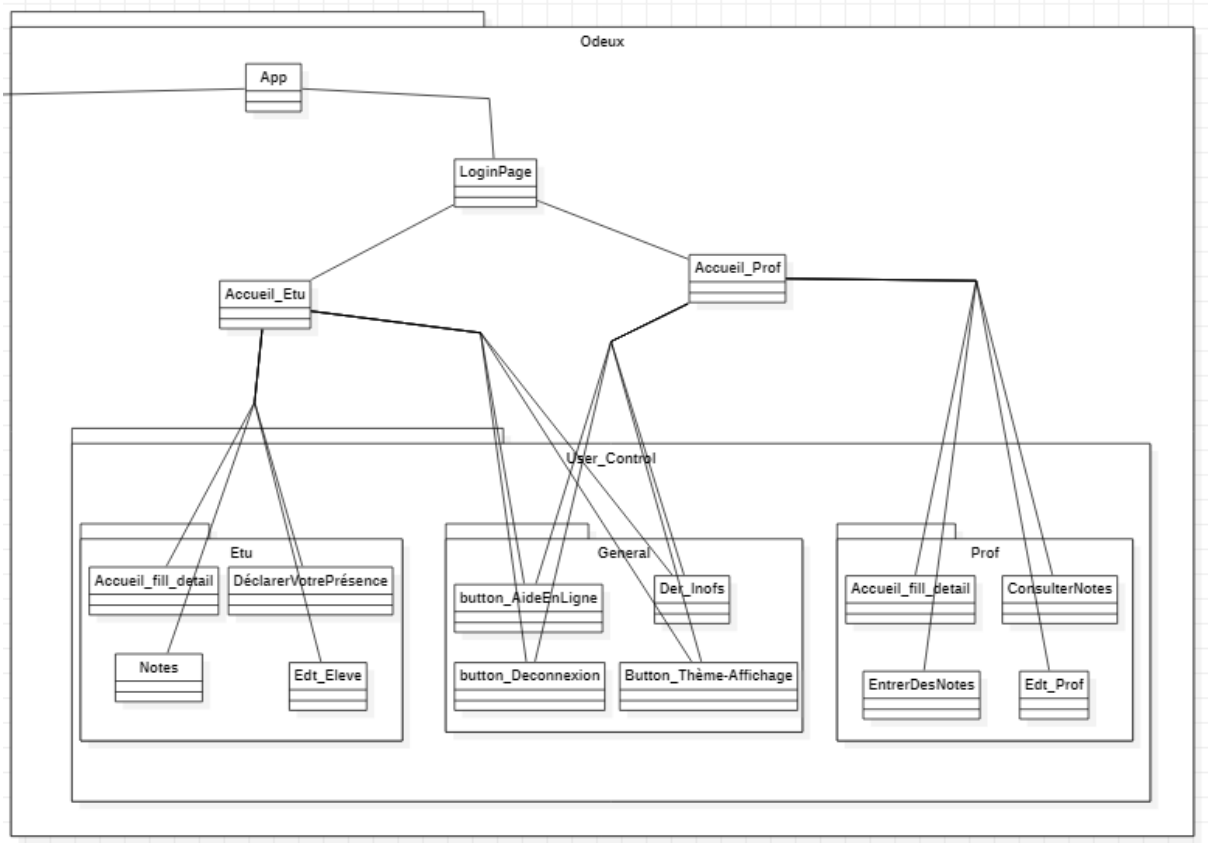
où c'est un étudiant est renvoyé sur la page Elève sinon si il s'agit d'un professeur qu'on renvoie sur la page Professeur.

Description écrite de l'architecture:



- Description :

Commençons par la partie graphique, qui est le Xaml:



Il se situe dans le package **Odeux** et a une dépendance avec le package **Modele** pour pouvoir faire le binding. Nous avons dans cette partie graphique le "coeur" de l'application, la page "LoginPage". C'est une page classique qui contient

le titre de l'application, sa partie centrale est composé de deux TextBox (le deuxième est une passwordBox), et un button connection, qui va permettre l'action de pouvoir se connecter à l'application si la première textBox possède un nom existant et le deuxième le mot de passe correspondant au nom, puis nous avons l'appel à l'User Control "Button Thème-Affichage" (dans le package General) qui est le changement de couleur sur la page.

Puis si la connexion est établie et que le nom correspond à un professeur, on se retrouve sur la page "Accueil_Prof" ; si c'est un élève on se retrouve sur la page "Accueil_Etu". Cette page contient tous les user controls qui lui sont attachés(package Etu dans le package User_Control) comme:

- Accueil_fill_detail: c'est l'accueil pour l'étudiant, où l'on retrouve un bouton qui supprime tout la liste des cours dépassés, une listbox contenant les cours du jour (à la date actuelle) et une autre listbox contenant les nouvelles notes que les professeurs ont entré pour l'utilisateur connecté.
- Notes: affiche une liste de toutes les notes de l'élève selon l'UE, le semestre choisi.
- Edt_Eleve: contient une listbox qui affiche une liste des cours de la date entrée sur le datepicker.
- DéclarerVotrePrésence: possède une listbox où sont affichés tous les cours manqués (avant la date d'aujourd'hui), où l'on peut sélectionner un de ces cours et cliquer sur le bouton déclarer qui va déclarer notre présence et le supprimer de la liste.

Mais nous utilisons aussi les user controls (comme le professeur) du package General:

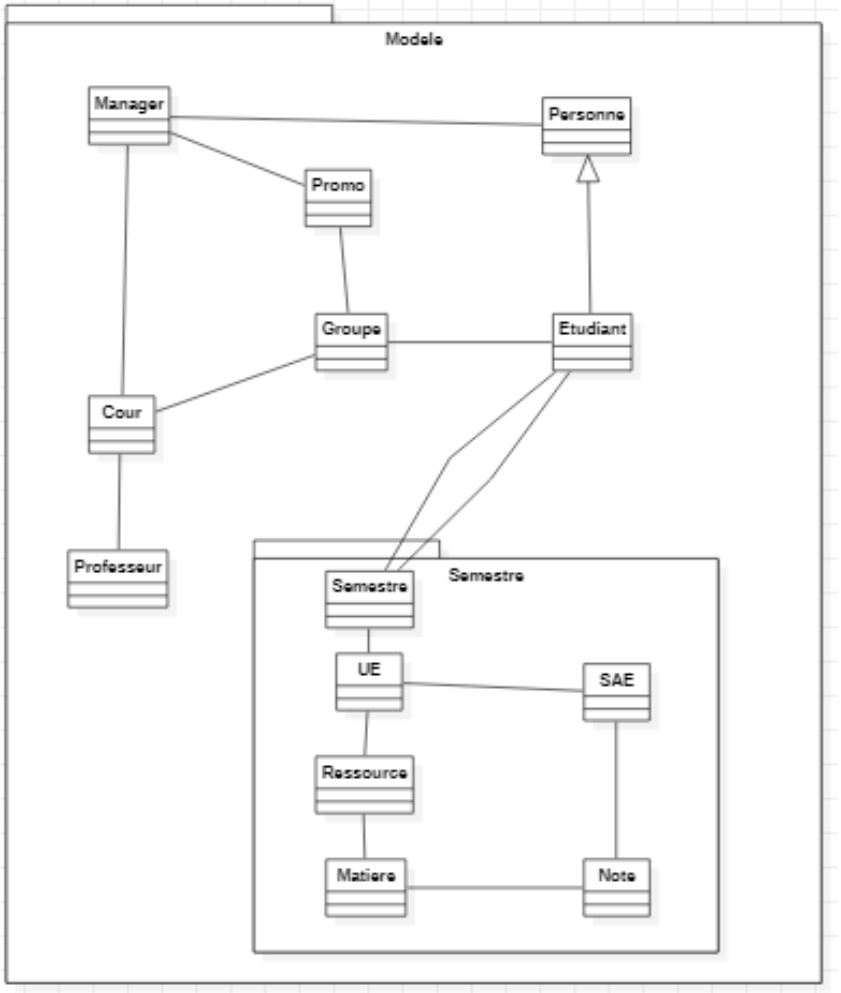
- Button_AideEnLigne : bouton qui renvoie sur la page d'assistance d'Odin/Odeux:https://jira.dsi.uca.fr/servicedesk/customer/portal/29/user/login?destination=portal%2F29
- Button_Deconnexion : bouton qui déconnecte l'utilisateur de sa session pour le renvoyer sur la page Login (Un message est envoyé pour confirmer le choix de l'élève)
- Der_Infos: Contient qu'une ListBox avec des informations banales et non modifiables.

Si c'est un professeur, il a de son côté (en plus que le package General) le package Prof, qui contient les User-Controls:

- Accueil_fill_detail: l'accueil pour le professeur, où l'on retrouve un bouton qui confirme la présence de tous les élèves, une listbox contenant les cours du jour (à la date actuelle où est la personne connectée).

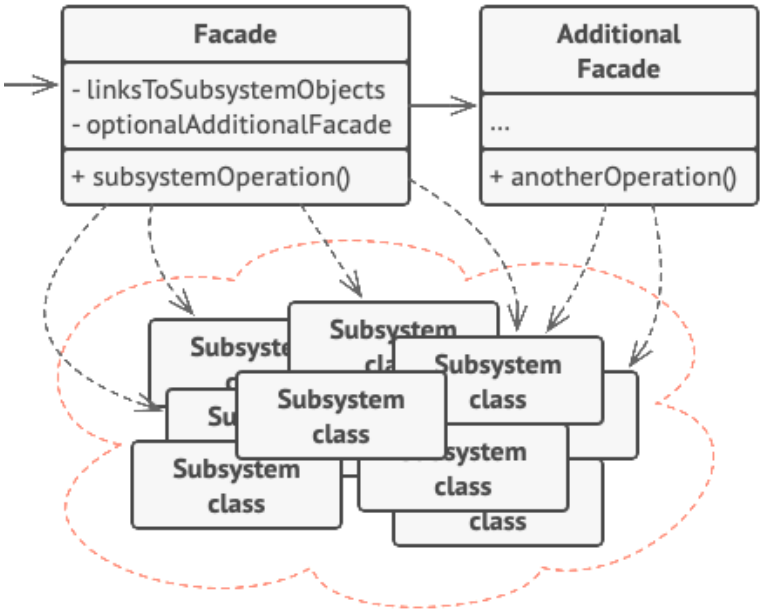
- ConsulterNotes: Va permettre au professeur de choisir le groupe d'élèves et l'élève de ce groupe, pour voir l'entièreté de ces notes, de ces moyennes à chaque UE, Ressource…
- EntrerDesNotes: Elle reprend l'architecture et les fonctionnalités de Consulter Notes mais a la capacité d'ajouter des notes dans la matière qui est choisi.e
- Edt_Prof: Reprend le même concept que Edt_Eleve mais s'adapte pour afficher l'emploi du temps du professeur qui a ouvert sa session.

Passons au côté C#, plus précisément le Modèle de l'application :



Le choix de cette conception de diagramme a été réalisé de sorte à ce qu'il puisse répondre aux attentes dans la partie graphique.

Passons maintenant aux patrons de conception utilisée. Dans mon diagramme, nous en avons utilisé un : la façade. Le but du patron façade est de simplifier l'accès à un système complexe. L'utilisateur du système, nos vues, doit alors passer par la façade pour accéder aux éléments du modèle



Dans mon modèle, cette façade est bien évidemment Manager. Il est celui qui va être relier à la partie Xaml (via la classe App), il permet de modifier, d'ajouter ou de supprimer des cours, personnes, etc. Pour cela le manager a une dépendance avec les classes: Cours, Promo, personnes et étudiants, pour pouvoir afficher les cours actuels d'un étudiant ou professeur, regarder dans la promo la Moyenne Général que se soit pour un semestre, une UE, une Ressourc, en sélectionnant à chauqe fois un élève précis. Cela permet aussi de regarder dans la liste des personnes si l'utilisateur existe dans la page Login et que si sa combinaison id/mdp correspond à un professeur ou un étudiant. Éétant que la classe Prof et Étudiant vont hériter de la classe Personne, la classe Etudiant contient en plus des informations d'une personne, deux semestres qui vont permettre l'affichage dans l'User-Control "Notes", répartie en UE, Ressource, Matière…

Un professeur a seulement des opérations d'ajout de notes qui vont se faire dans la partie graphique via l'user control "Entrer des notes". Le Modele a été réalisé de façon à pouvoir grouper les élèves en plusieurs groupes différents pour que chaque cours ne soit attribué pas qu'à un seul ou tous les élèves, mais à une partie d'élèves qui sont dans le même groupe.

Ps: Chaque Document présenté ci-dessus à été mis sur le gitlab et rendu dans un zip à part (Documents_Odeux.zip)