VAMPIRE SURVIVORS

COMPAGNON

Documentation et diagrammes

Mielcarek Félix - De La Fuente Axel - Astolfi Vincent

10/06/2022 - 1ère année - Groupe 10

Sommaire

Explications relatives aux rendus

Explication de l’API et de la partie note

Notre application se base sur un jeu se trouvant sur le distributeur de jeu en ligne numéro 1 Steam. En effet, il est possible d'acheter des jeux en ligne sur cette plateforme, cela permet l’accès au jeu que l’on possède sur tous les appareils où l'on souhaite jouer. C’est la cas du jeu que nous avons choisis pour développer notre application “compagnon”.

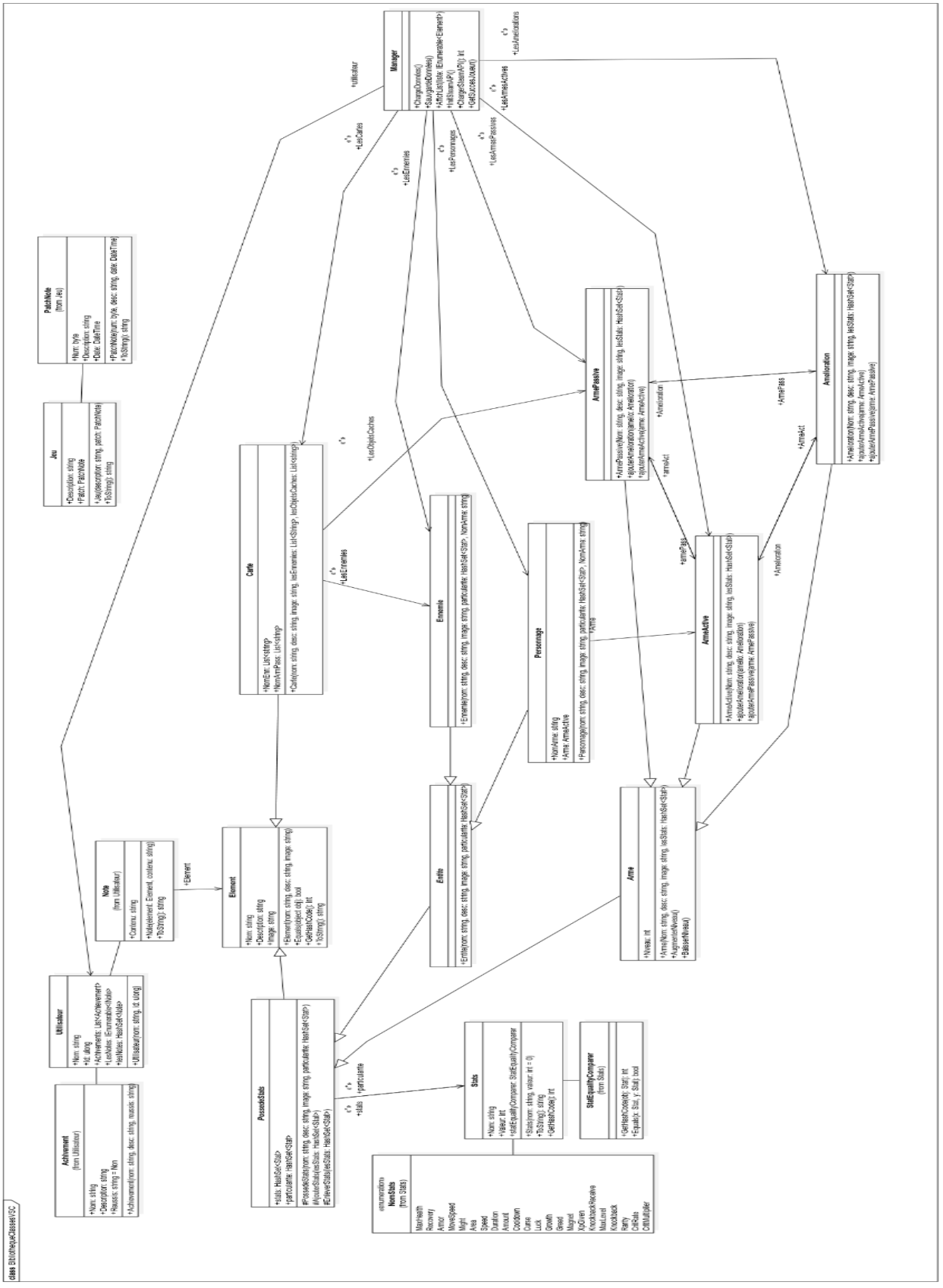
Sur notre application nous avons décidé d'ajouter, en ajout personnel, une API qui va nous permettre de réceptionner les informations du compte Steam utilisé par la personne qui utilise l'application. Cela permet que l'utilisateur ait accès, directement depuis l'application, à ses données en jeu tel que, les succès qu'elle a réalisé sur celui-ci. Ainsi, l'utilisateur une fois connecté à son compte Steam pourra laisser des notes aux objets qu'il regarde afin de pouvoir les retrouver dans le futur. Ces notes seront toutes visibles depuis la partie visuelle réservée à son compte. Les notes pourront lui permettre d'écrire des informations dont il aimerait se souvenir à l'avenir par rapport aux objets qu'il consulte comme par exemple un combo d'arme fort. Ou bien, au contraire, une arme qui ne fonctionnait pas bien avec une autre voir qui serait incompatible toutes les deux.

Information par rapport aux schémas

Tous les diagrammes et schéma présents dans ce document sont disponibles en version SVG dans le dossier “Schema_version_SVG” présent avec ce dossier ou que vous pourrez retrouver dans le dépôt GitLab dans la partie documentation.

Diagrammes

Diagramme de classes (bibliothèque de classes)



Explications et détails du diagramme de classe

Le jeu Vampire Survivors est très complet et possède énormément de données. Afin de les répertorier au sein de notre application, nous avons dû mettre en place une architecture de classes assez complexe.

Nous sommes partis d'une classe mère abstraite "Element" qui possède en attributs un nom, une description et un lien vers une image. Toutes les données relatives à Vampire Survivors que nous allons traiter par la suite, sont des filles de cette classe. (Armes, Personnages, Ennemies, etc…).

Cependant chacune des classes citées ci- dessus possèdent des statistiques qui lui sont propres. Nous avons donc dû mettre en place une classe abstraite intermédiaire, permettant d'ajouter des statistiques aux classes Armes, Personnages et Ennemies. La classe Carte ne possède pas de statistiques, elle est donc une fille directe de la classe "Element". Cette dernière possède tout de même un attribut, liste, composé de d'armes passives.

Pour revenir plus en détail sur la classe abstraite "Stat", on peut voir qu'elle possède deux classes en elle. Le première classe "NomStat" est une "enum" qui permet de répertorier les noms de toutes les statistiques qui existent dans le jeu, afin d'y accéder, rapidement et efficacement. La deuxième classe "StatEqualityComparer" est un protocole d'égalité qui permet de comparer si deux "Stat" sont égales.

La classe "PossedeStat" permettant l'attribution de statistiques, est une classe abstraite et une classe mère de le classe "Arme" et la classe "Entite". Ces deux dernières sont des classes abstraites. Ce sont les classes mère des classes finales, qui sont : ArmeActive, ArmePassive, Amelioration, Personnage et Ennemie.

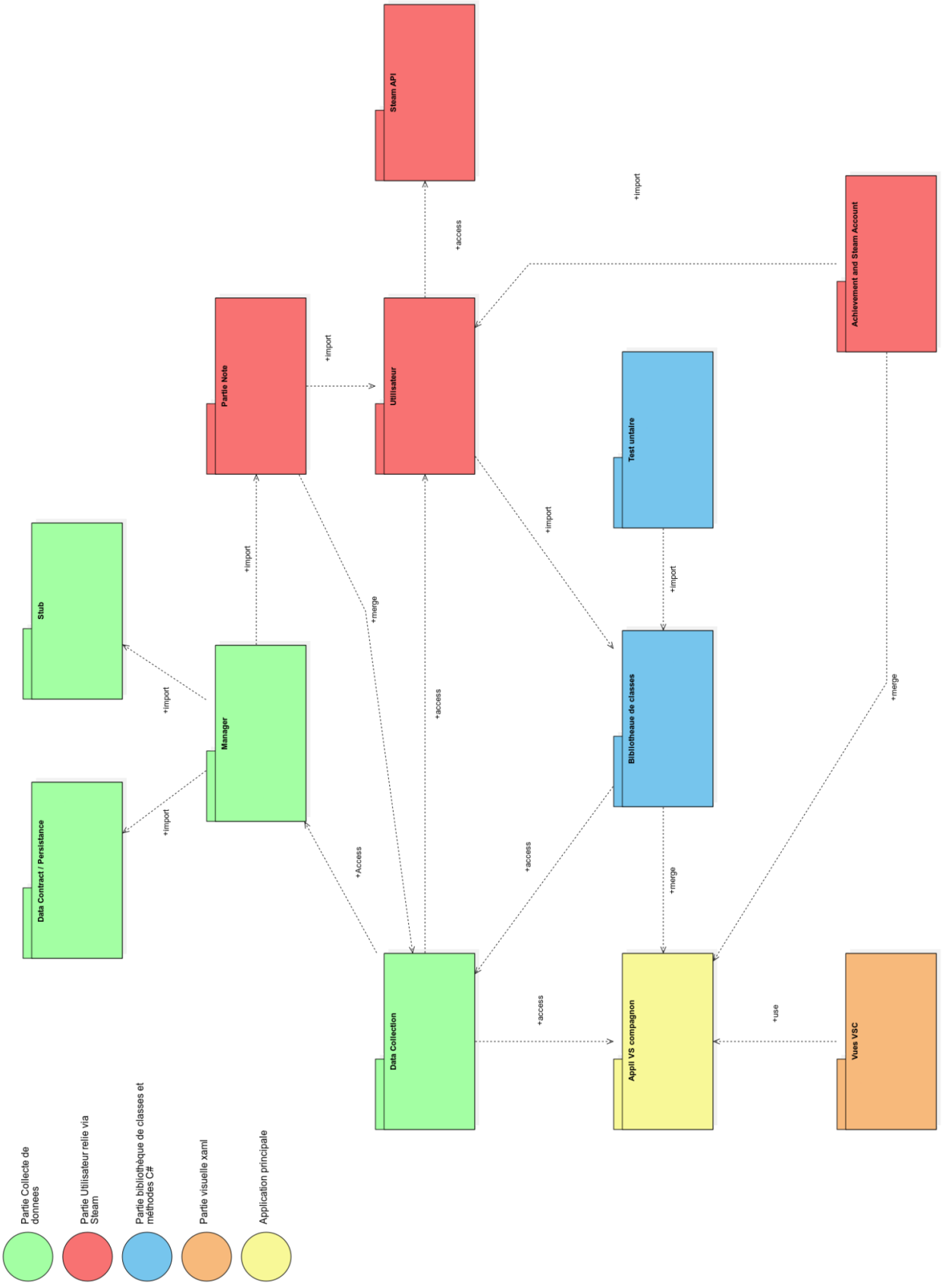
Le diagramme nous laisse voir que chacune de ces classes sont reliées les une aux autres. Cela est justifié par le fait que chaque arme possède un attribut de l'amélioration possible qu'elle peut faire. Chaque personnage possède un attribut d'une arme, afin de savoir quelle arme il possède en début de partie.

Pour finir nous avons les classes "Jeu", "Utilisateur" et "Manager". En ce qui concerne la classe jeu, il s'agit d'une classe concernant les notes de mise à jour du jeu.

La classe "Utilisateur", est une classe permettant de stocker des informations sur un utilisateur. La classe possède une classe interne, appelée "Note". La classe "Note" possède des attributs permettant de lier une note écrite par l'utilisateur, à un "Element" de l'application (Armes, Personnages, Cartes…).

Enfin la classe "Manager" est la classe permettant de charger les données du stub ou de la persistance et d'effectuer le binding correctement. Elle possède des listes de chaque "Element" de l'application, ainsi qu'un utilisateur, si ce dernier est connecté.

Diagramme de paquetage



Explications et détails du diagramme de paquetage

Notre application Vampire Survivors Compagnon se décompose en plusieurs sous parties.

La première d'entre elles est la partie VuesVSC, cette partie contient toute la partie graphique de notre application avec le code en xaml.

De plus notre application comporte aussi une partie Bibliothèques de classe qui correspond à toute la partie code behind de notre application avec les différentes classe en C#.

Les méthodes des classes présentes dans la bibliothèque de classe sont testées lors de tests Unitaires. Ces derniers utilisent donc uniquement le code présent dans celle-ci afin de les tester et de voir si elles sont bien toutes fonctionnelles.

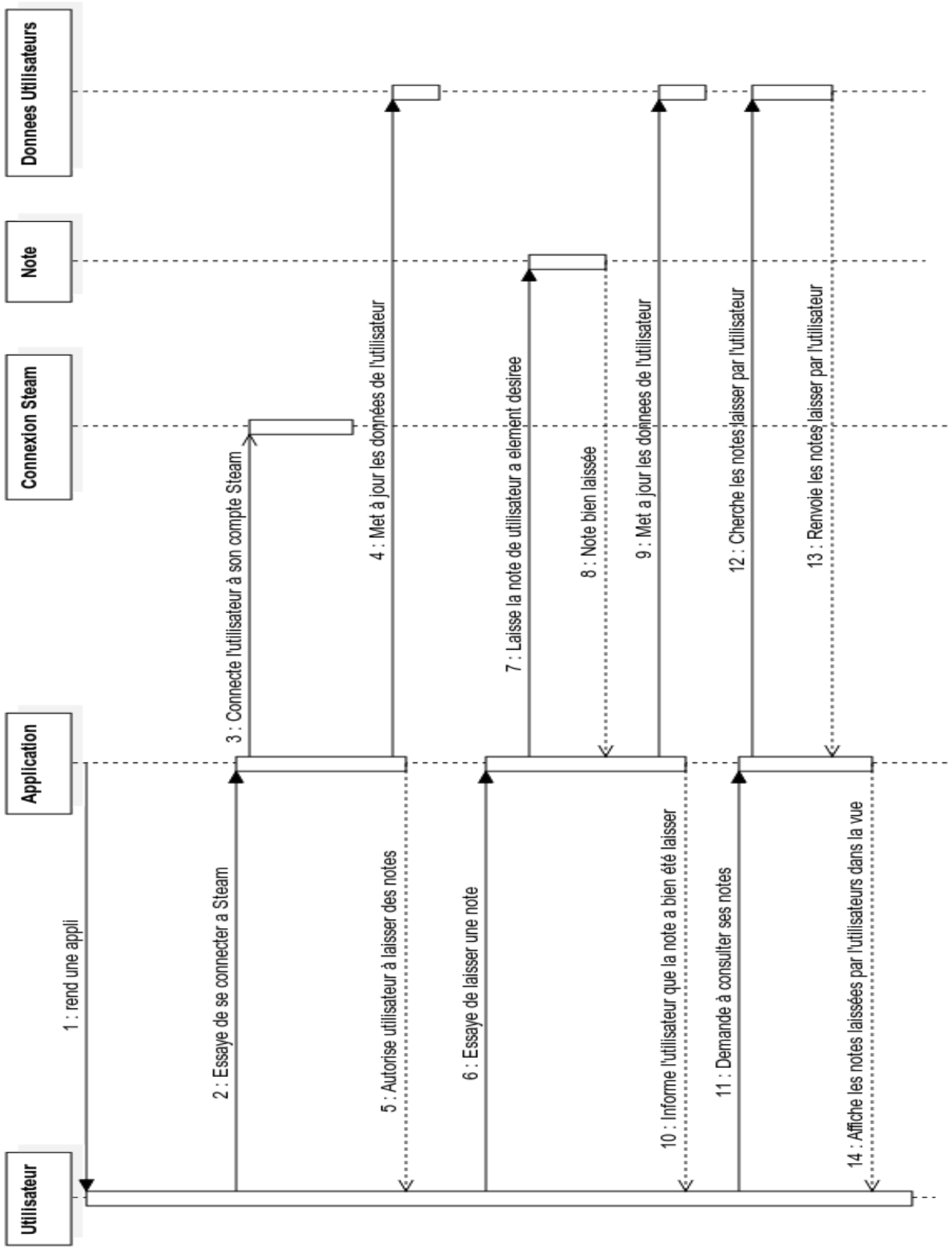
La partie donnée de notre application se fait grâce à un manager qui tient ses données d'un stub et d'un data contract. Ces derniers nous permettent de mettre en place la partie persistance de notre application qui vont nous permettre de garder les différentes données entre deux utilisations de l'application par l'utilisateur.

L'utilisateur peut laisser des notes aux différents objets présents dans notre appli. Ces notes sont donc stockées dans la persistance de l'application grâce au manager.

Pour qu'un utilisateur puisse laisser des notes il a besoin de se connecter à son compte steam qui se fait grâce à la classe utilisateur de notre bibliothèque de classe. Cette classe permet donc de faire le lien entre une API connectée directement à Steam et un utilisateur.

Grâce à cette API on peut également accéder aux informations de l'utilisateur comme par exemple son compte steam ainsi que les succès qu'il possède sur le jeu Vampire Survivors. Partie succès que nous affichons dans l'application.

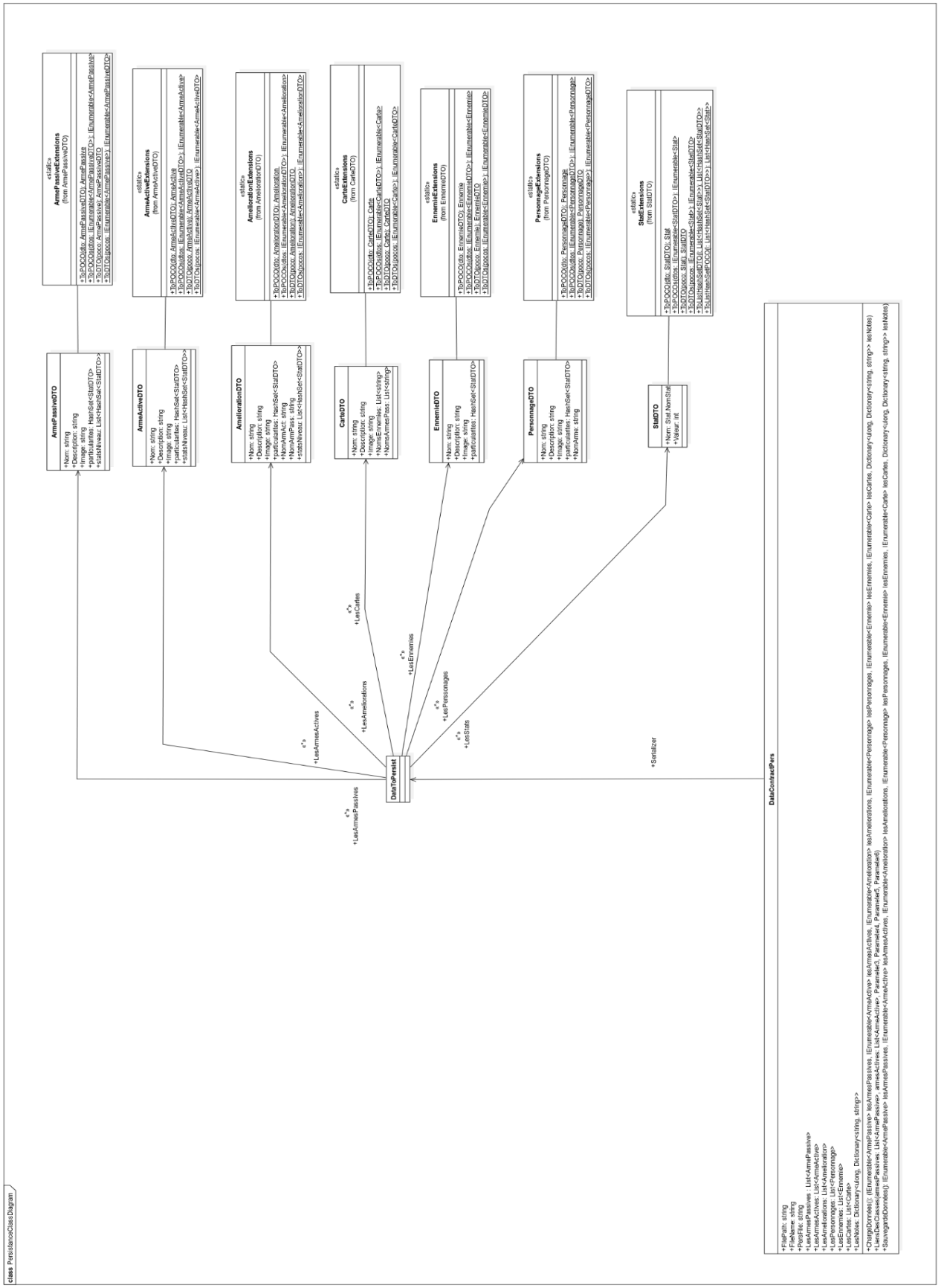
Diagramme de séquence



Explication du diagramme de séquence

La partie de notre application où l'utilisateur est en relation directe avec les données de celle-ci se trouve dans la partie des notes. En effet, lorsque l'application rend la partie
visuelle à l'utilisateur, celui-ci a la possibilité d'ajouter des notes aux objets qu'il consulte. Pour se faire il doit au préalable se connecter à son compte steam via le bouton
situé en haut à gauche de la page. Pour se connecter l'utilisateur demande à l'application de lancer une tâche asynchrones qui va, en tâche de fond, cherche si l'utilisateur possède steam,
si son compte est ouvert et si il possède le jeu Vampire Survivors dans sa bibliothèque de jeu. Si ces trois recherches sont fructueuse alors, l'utilisateur sera désormais connecté à steam
via l'application, il pourra ainsi consulter sa partie avec son nom d'utilisateur ainsi que toute les notes qu'il aura laissé au préalables qui auront ete sauvegarder dans l'application
grâce à la partie persistance de celle-ci. Une fois connecté, il peut aussi laisser de nouvelles notes. Pour cela il va pouvoir, directement depuis la page d'un objet, écrire dans une
boîte de dialogue situe en bas de la page sa note qui sera donc transmise à ses données utilisateurs via l'application et la classe utilisateur. Une fois sa note laisser elle est stockée
dans les données de l'utilisateur qui pourra donc la retrouver via sa page personnelle. Il peut donc ainsi consulter toutes les notes qu'il aura écrites auparavant.

Diagramme de classes (bibliothèque de persistance)



Explication du diagramme de classe de la persistance

La classe DataContractPers gère toute la sérialisation et la désérialisation, en effet c'est une interface IPersistanceManager.
Cette classe possède un serializer qui fonctionne avec des fichiers XML et qui prend des DataToPersist. Cette classe possède des collections des éléments DTO. Ce sont des classes semblables à celle du modèle initial mais adapté à la gestion des données.

Description de l’architecture et des choix d’implémentation

Pour la navigation dans nos vues nous avons fait le choix d’utiliser un Navigator car nous voulions que des boutons présents dans un UserControl puissent modifier le UserControl parent. Cela rend aussi le code plus clair.

Pour la persistance nous avons choisi de l’implémenter avec des conversions d’objets POCO en objets DTO pour éviter d’avoir un mélange du modèle et de la persistance.

Pour la sauvegarde des données nous avons choisi d’appeler cette fonction du manager lors de la fermeture de l’application et non pas à chaque modification des données à sauvegarder pour éviter des appels de cette fonction inutiles.
Dans la classe Utilisateur nous avons choisi d’implémenter une liste de INote créée à partir d’une liste de Note. Cela permet de pouvoir afficher les notes sans pouvoir les modifier sur la page de profil, tandis que la liste peut être modifiée dans les pages d’éléments.

Les collections sont wrap dans des ReadOnlyCollection car elles ne sont pas vouées à être modifiées, cette encapsulation profonde permet donc de ne les rendre que affichables.

Les statistiques ne sont égales que si le nom et la valeur sont égales, cependant pour garantir l’unicité des éléments dans les HashSets de stats nous avons implémenté un EqualityComparer qui ne se base que sur le nom de la statistique.