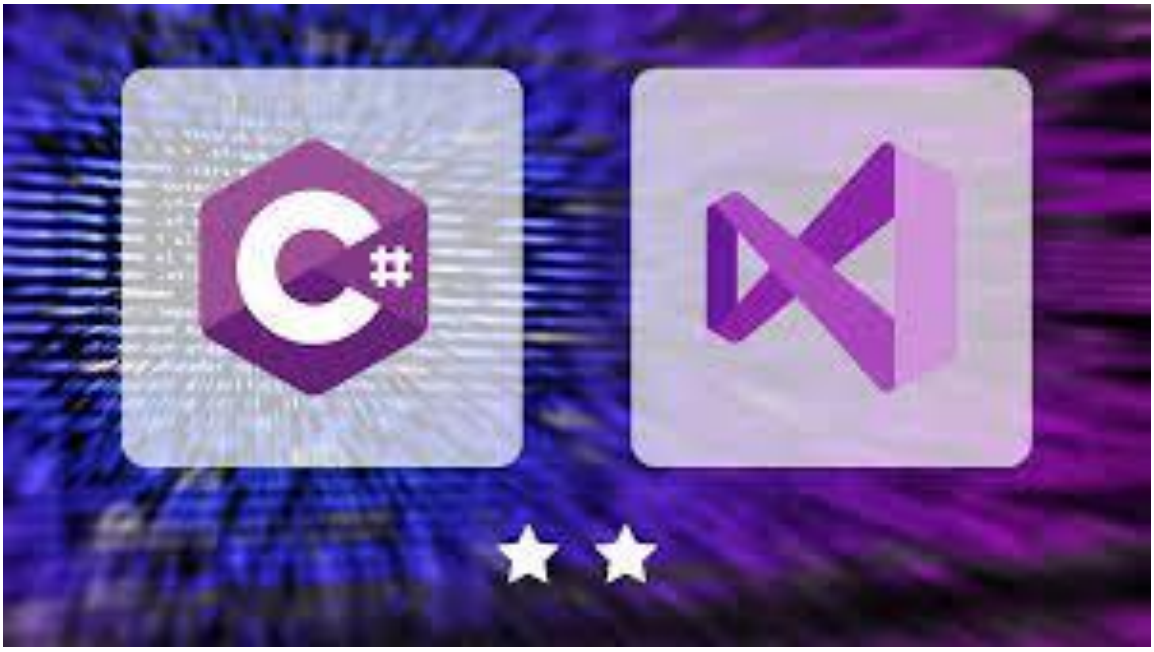


Conception et Programmation Orienté Objets (C#, MAUI, XAML)

SAE 2.01 - IHM

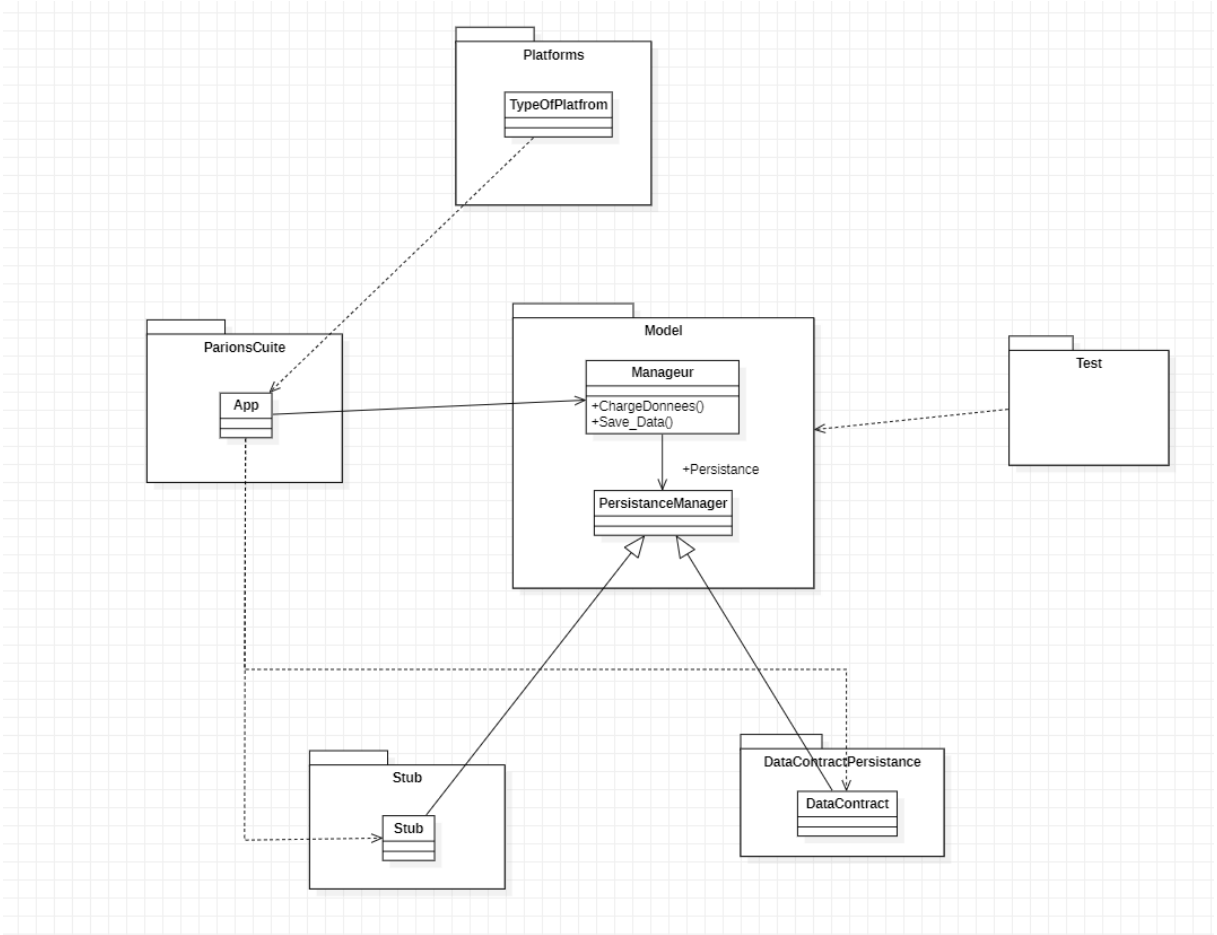
FAGES Tony | MUZARD Thomas | 06 / 06 / 2023

Sommaire :

CONTEXTE

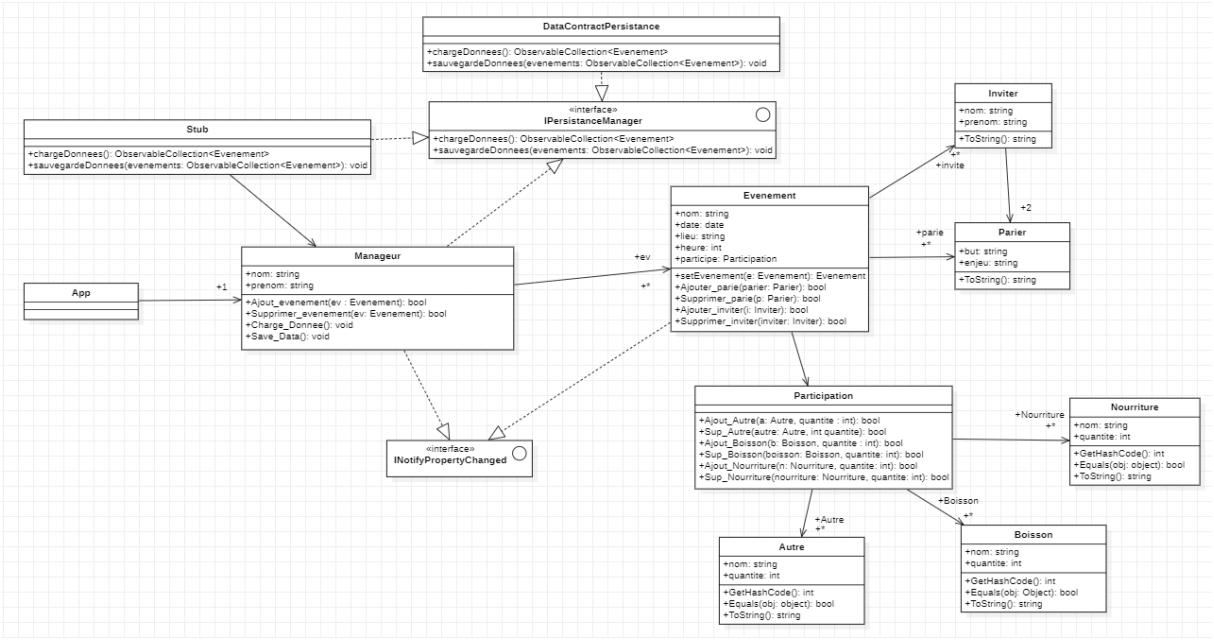
Dans la cadre de la SAe 2.01, nous avons développé une application qui a pour objectif d'organiser des évènements festifs. L'application permet de gérer les invités, tout ce qui est nécessaire pour la soirée (Nourriture, Boisson, Alcool, Matériel, …), toutes les informations qui sont nécessaire de l'évènement (lieu, date, …). Toutefois, la fonctionnalité principale de notre application et la gestion des paries effectuer durant la soirée. Il est possible pour 2 invités de parier sur ce dont il le souhaite, définir l'enjeu et le perdant devra réaliser un gage.

DIAGRAMME DE PAQUETAGE



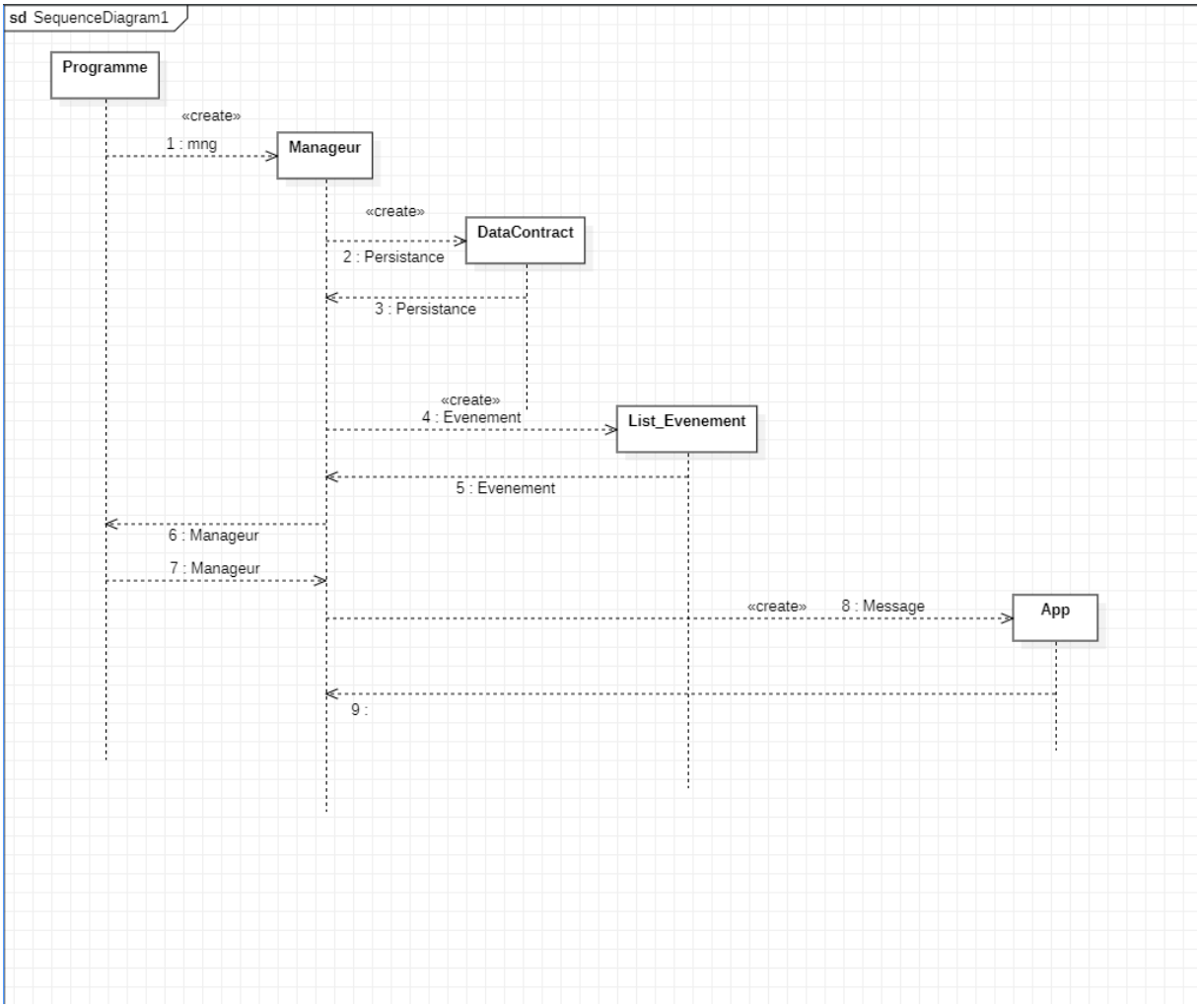
Dans ce diagramme de paquetage, nous avons dans un premier temps ParionsCuite qui contient l'App de notre application. Elle permet de gérer l'ensemble des vues XAML pour l'aspect visuel de notre application. L'app à une dépendance avec le Manageur qui se trouve dans le package Model. Ce package nous permet de faire le lien entre les Classe C# et le code XAML, ou plus simplement, nous permet de lier nos fonctionnalités d'application avec nos views. Lorsque l'application est utilisée, des données doivent être stocker, ou encore lorsqu'on lance notre application, les données que nous avions précédemment, doit apparaitre. Dans notre cas, nous avons 2 possibilités, soit nous passons par des données en brut que l'on met dans un stub, très pratique pour tester nos fonctionnalités ou encore faire nos tests dans notre Package Test. Soit, nous utilisons le DataContract qui a pour objectif de charger les données de l'application via un fichier (dans notre cas xml) que nous pourrons retrouver grâce au Persistance Manager, lorsque le manageur va charger les données c'est-à-dire, prendre les données qui sont dans le fichier xml, ou encore les sauvegarder, lorsque nous allons quitter l'application encore dans notre fichier xml. Enfin, avec Maui qui nous permet de développer des applications multiplateformes, nous avons une package Platform qui contient plusieurs builder permettant de build l'application sur différente plateforme tel qu'Android, IOS, Windows, …

DIAGRAMME DE CLASSE



Dans notre diagramme de classe, nous avons tout d'abord un manageur qui a pour objectif de gérer toutes les fonctionnalités de notre application, il représente l'utilisateur de l'application que l'on peut identifier via son nom et prénom. Pour cela, il a une liste d'évènement puisqu'un utilisateur peut organiser une ou plusieurs soirées, mais c'est aussi lui qui va avoir les méthodes implémenter dans l'interface PersistanceManager, qui a pour objectif, de charger au démarrage de l'application, toutes les données qu'il y avait d'enregistrer, mais aussi de les sauvegarder pour qu'a chaque ajout, suppression et ou modification, lorsque l'utilisateur redémarre l'application, qu'il puisse récupérer l'ensemble des ses données. Ainsi chaque évènement est composé d'un nom, d'une date, du lieu, de l'heure ainsi que d'une liste d'invités qui regroupera l'ensemble des invités qui seront présent à la soirée, d'une liste de paris, puisque, durant une soirée, il est possible de parier sur tout et n'importe quoi. Et enfin, une participation qui regroupe l'ensemble des listes de ce qui sera nécessaire d'amener durant la soirée. La classe Evènement a 5 méthodes, lui permettant de modifier les informations de l'évenement, d'ajouter ou de supprimer un pari et d'ajouter et supprimer des invités. Un invité se caractérise par son nom et prénom, et contient une méthode ToString pour définir son affichage. Un pari nécessite un invité qui sera le parieur , un autre invité qui sera celui qui reçoit le défi du pari, d'un but qui définira les termes du pari et un enjeu, qui définira le gage en cas de perte de l'un ou de l'autre. La classe Participation est donc notre classe qui contient la liste de nourriture avec ses méthodes d'ajout et de suppression dans la liste. La classe nourriture quand a elle se caractérise par un nom et une quantité. Nous retrouverons exactement les même classes la liste de boisson et l'objet Boisson et la liste Autre et l'objet autre, qui désigne l'ensemble du matériel nécessaire pour la soirée (chaise, table, lumière, …). Pour ce qui est du sauvegarde des données, ou l'on passe par le stub qui nous permet de charger des données en brut, pratique pour nos tests, soit nous passons par la persistance qui va sauvegarder nos données dans un fichier xml. Enfin, l'App correspond à l'exécutable de notre application qui créé donc un manager pour gérer toutes les fonctionnalités de notre appli.

DIAGRAMME DE SEQUENCE



Tout d'abord, nous initialisons un Manager sous le nom de « manager ». Il va quant à lui, instancier un DataContract via la Persistance, la liste des évènements du manager. Après tout ceci, l'adresse du manager ira au programme pour que le Manager puisse instancier l'App pour naviguer avec toutes les données précédemment chargées dans les listes.

ARCHITECTURE DE L'APPLICATION

Pour l'architecture de notre projet, nous avons un manageur qui représente l'utilisateur de notre application. Notre manageur aura accès a l'ensemble des méthodes pour naviguer sur l'application et utilisé les diverses fonctionnalités qu'elle propose. Ainsi il peut créer ou supprimer des évènements. Dans notre application, un évènement correspond à une soirée et contient tout ce qui est nécessaire pour organiser au mieux une soirée. Nous avons une liste d'invité permettant de gérer les individus qui seront présent ou non à la soirée, il est important de rentrer tous les invités présents pour pouvoir utiliser convenablement les autres fonctionnalités de l'application. Un évènement à aussi une liste de paris, qui quant à lui, s'occupe de tous les paris que l'on veut faire durant une soirée. Du moment qu'un pari a un but, un enjeu et 2 invités représentant celui qui parie et celui qui accepte le pari, le pari est enregistré dans l'application. A la fin, nous pouvons déterminer sur l'application si le pari est terminé, ou toujours en cours, et dans le cas ou il serait terminé, déterminé le vainqueur. Un évènement a pour finir une participation. Une participation dans notre application regroupe l'ensemble des objets nécessaire à l'évènement. Dans participation nous avons une liste de nourriture, de boisson et d'autre. Comme son nom l'indique, pour la liste de nourriture est une liste de nourriture contenant le nom et la quantité qui sera amené. Il en sera de même pour les boissons et l'objet Autre, qui correspond à tout le matériel qui sera nécessaire à l'évènement tel que des chaises, tables, lumières,… Dans notre appli, la classe manageur hérite de INotifyPropertyChanged, lui permettant d'être constamment en relation avec les données de l'application via la Persistance, ou le Stub dans le cas de nos essaie. Ainsi, durant l'utilisation des fonctionnalités de l'application, nous pouvons faire des modifications en temps réel et elles seront sauvegarder lorsque nous quitterons l'application. Bien évidemment, au chargement de l'application, toutes les données sont aussi chargées pour que l'utilisateur puisse récupérer tout ses données créer, supprimer ou modifier dans une utilisation précédente.