

DOCUMENTATION

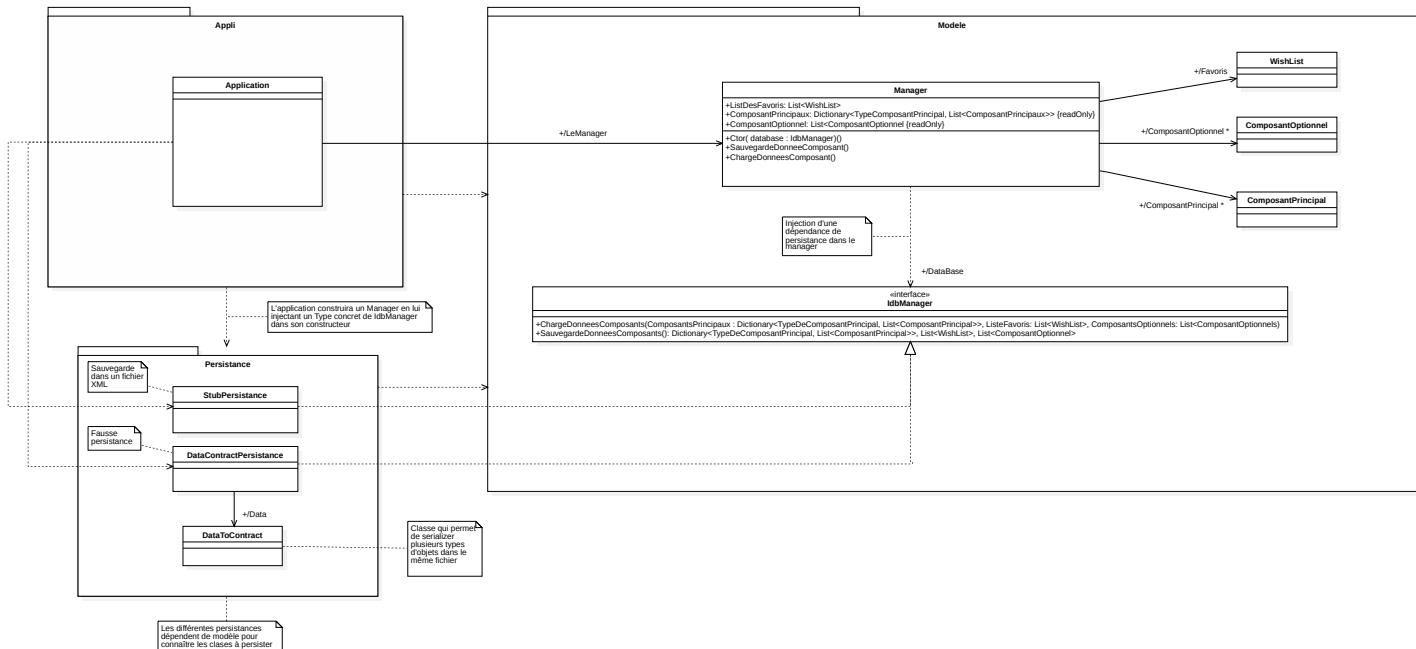
DIAGRAMMES ET ARCHITECTURE

PROJET-IHM FIN 1ÈRE ANNÉE

SOMMAIRE :

- 1- Diagramme de classe mettant en avant la persistance
- 2- Diagramme de classe mettant en avant l'architecture globale
- 3- Diagramme séquence Persistance
- 4- Diagramme de paquetage
- 5- Patron de conception et Architecture

1-DIAGRAMME DE CLASSE (PERSISTANCE)

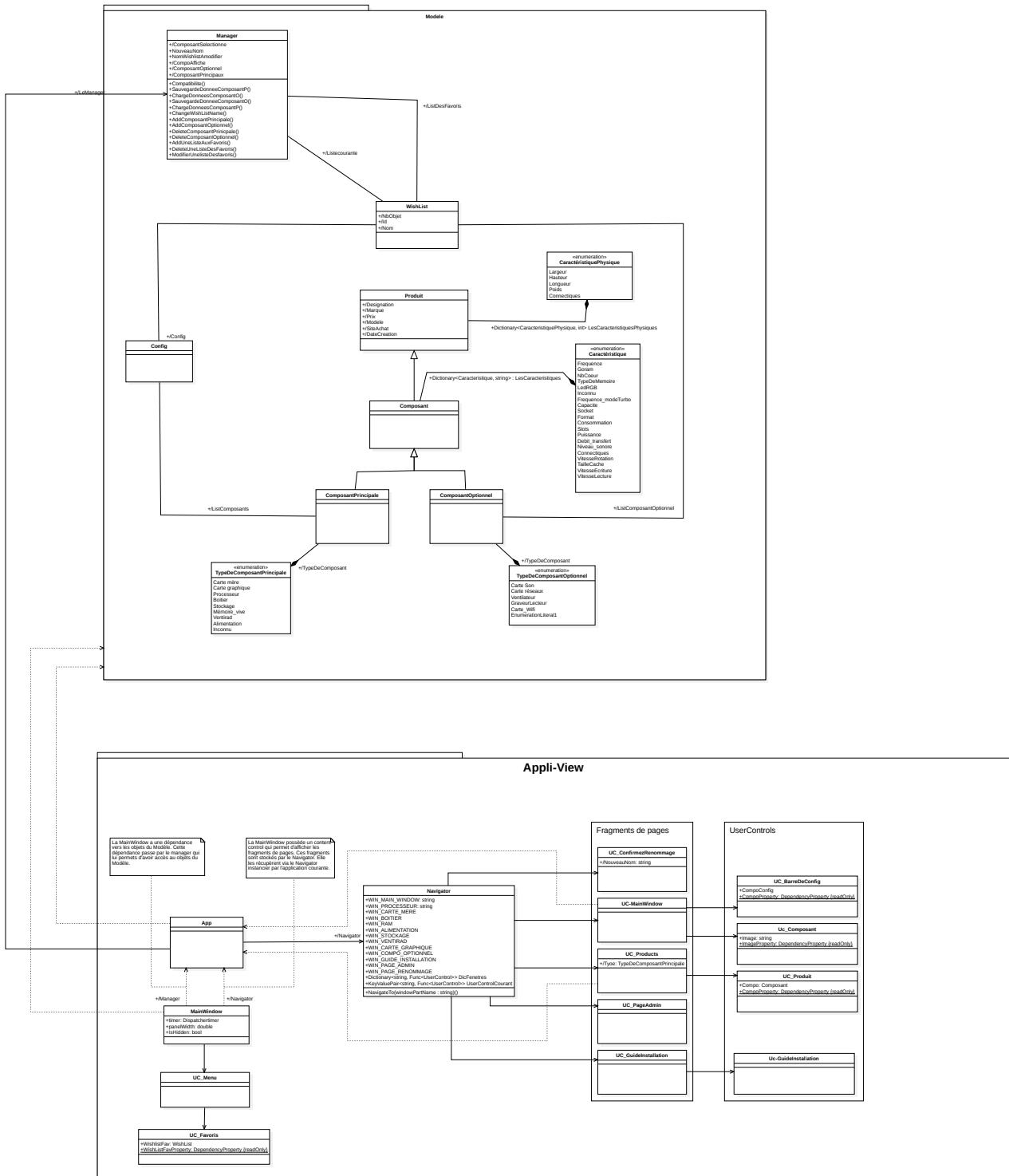


Ce diagramme montre uniquement les classes importantes au bon fonctionnement de la persistance.

L'application courante possède et instancie un Manager qui a pour rôle de gérer tous les objets du Modèle. Le manager quant à lui, possède une dépendance vers un type de persistance. Elle est injectée dans son constructeur. Ce système est générique et modélisé par une interface nommée : "IdbManager". Cette interface sera ensuite implémentée concrètement soit par un stub soit par unDataContract. Le Stub fera une fausse persistance en créant des objets à sa création tandis que leDataContract utilisera un fichier XML comme stockage de données.

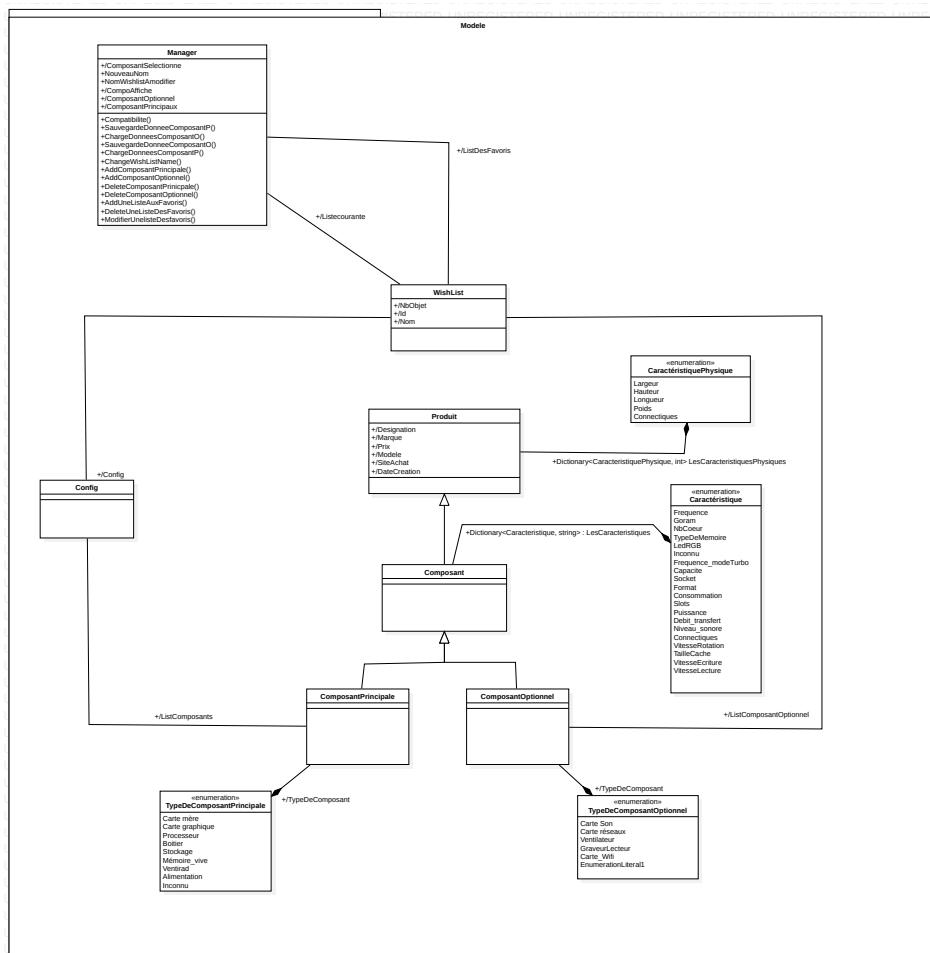
L'application a donc une dépendance envers un système de persistance qui a lui-même une dépendance vers les objets du modèle.

2 - DIAGRAMME DE CLASSE (ARCHITECTURE GLOBALE, AJOUTS PERSONNELS ET VIEWS)



Ce diagramme montre l'architecture globale de notre application, que ce soit le modèle avec nos ajouts personnels ou l'architecture des views.

ARCHITECTURE GLOBALE ET AJOUTS PERSONNELS

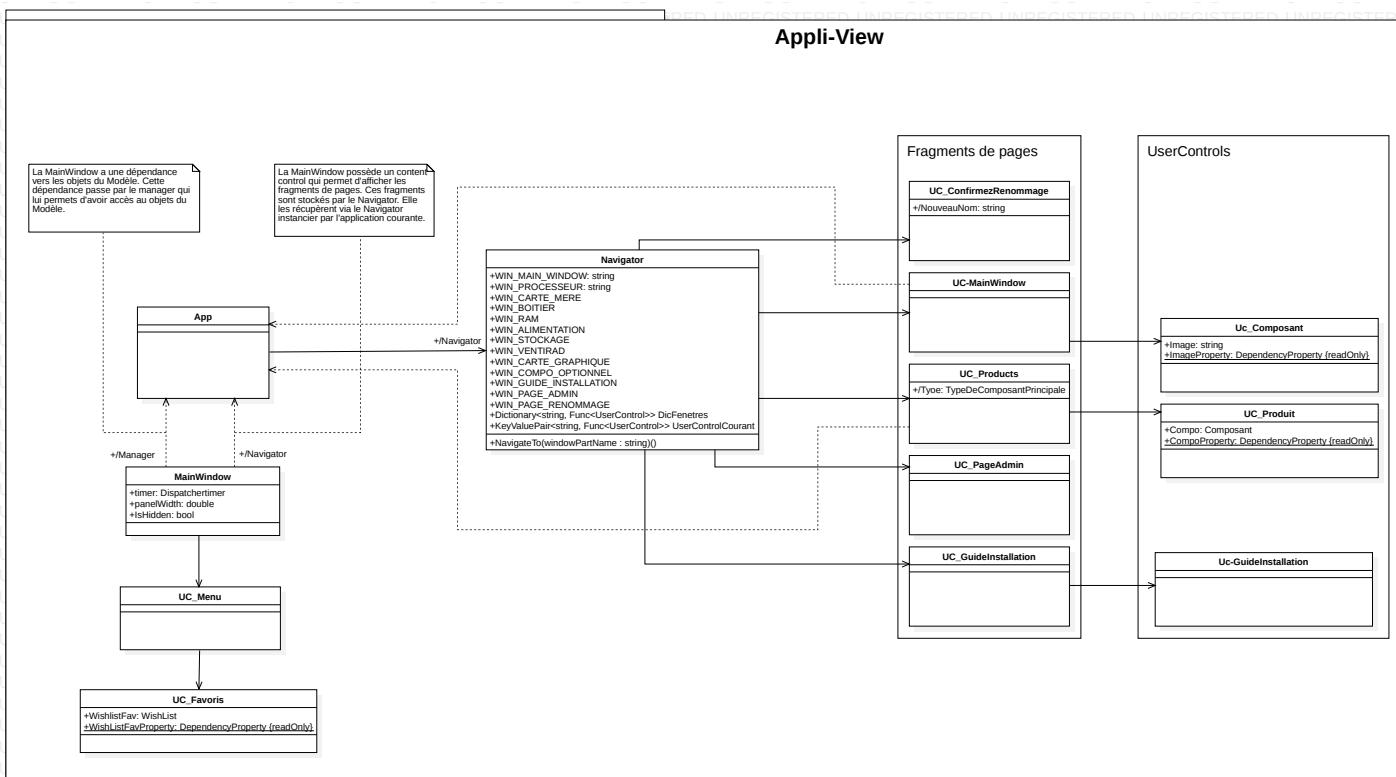


Nos objets principaux sont des composants, il y en a deux types, les principaux et les optionnels. C'est sur ces objets que nous effectuons un Master-Details.

Nous avons également décidé de rajouter la possibilité à l'utilisateur de se créer une configuration pour monter son ordinateur. Cette option se modélise par la classe **Wishlist**. Une wishlist possède une configuration qui est composée de tous les composants principaux choisis par l'utilisateur ainsi qu'une liste de composants optionnels qui viendront compléter son ordinateur. L'utilisateur peut enregistrer ses Wishlist favorites afin de les retrouver ultérieurement. Il peut également les renommer et les modifier. Nous avons également voulu rajouter un test de compatibilité pour simplifier les recherches de l'utilisateur. Pour que l'utilisateur soit guidé pour le montage, nous lui avons mis à disposition un petit guide de montage détaillé en 12 étapes.

Nous avons un Double Master-Details, le premier est la possibilité d'afficher tous les composants d'un certain type et l'autre d'accéder aux caractéristiques des différents produits de ces composants.

ARCHITECTURE DES VIEWS

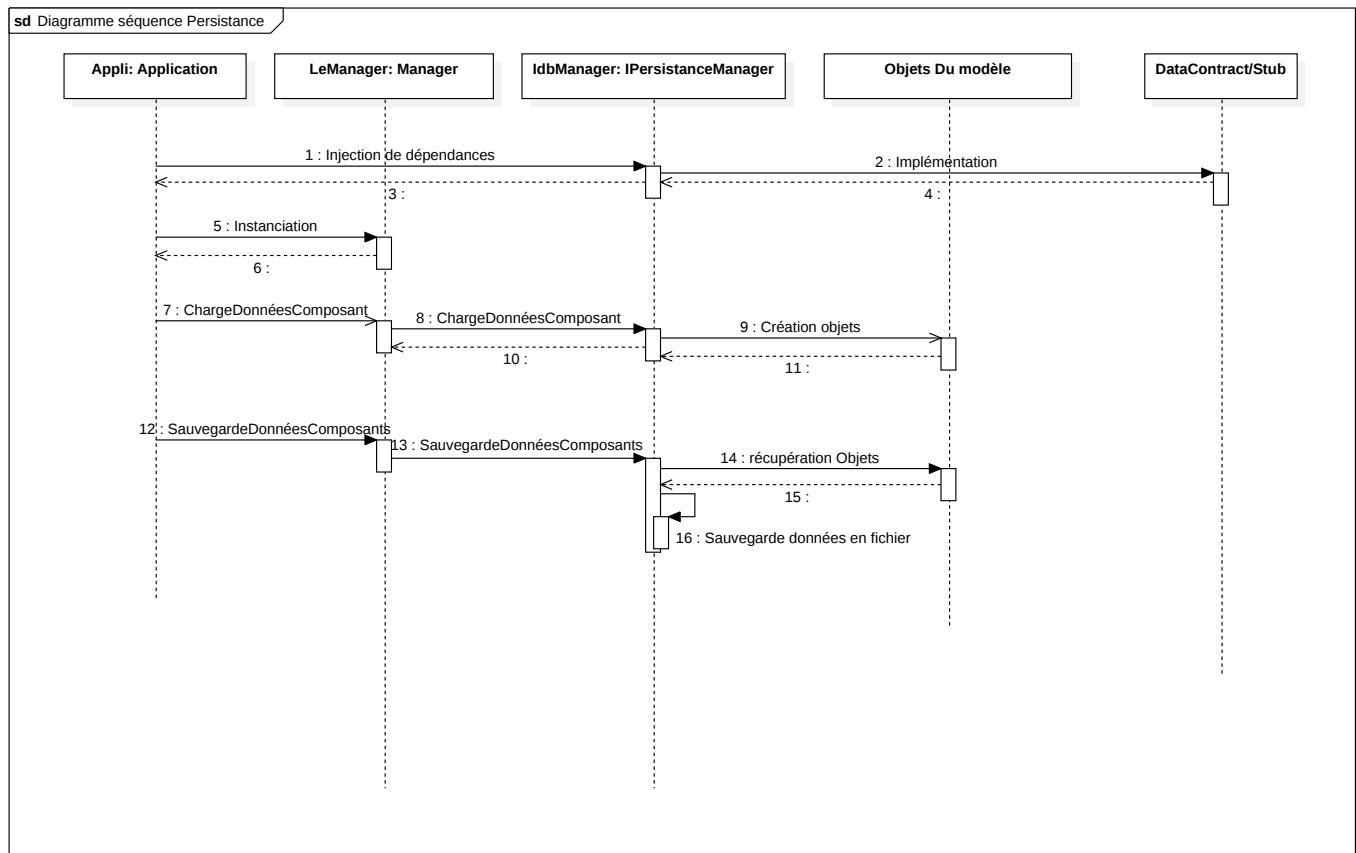


Ce double Master-Details passe tout d'abord par un choix de type de composants dans le fragment de page "UC_MainWindow". Une fois ce choix fait, l'utilisateur pour choisir d'afficher les détails d'un produit parmi ceux de ce type de composant.

L'architecture des views et la suivante :

Le Navigator instancié dans l'application courante possède tous les fragments de pages. La MainWindow possède un ContentControl qui va être rempli par les fragments de page du Navigator au fur et à mesure de la navigation de l'utilisateur. La MainWindow est donc en dépendance avec le Navigator de l'application. Pour afficher les données des composants, la MainWindow et certains fragments de pages ont besoin d'une dépendance vers le Manager de l'application courante qui possède toutes les données du modèle.

3- DIAGRAMME SÉQUENCE PERSISTANCE

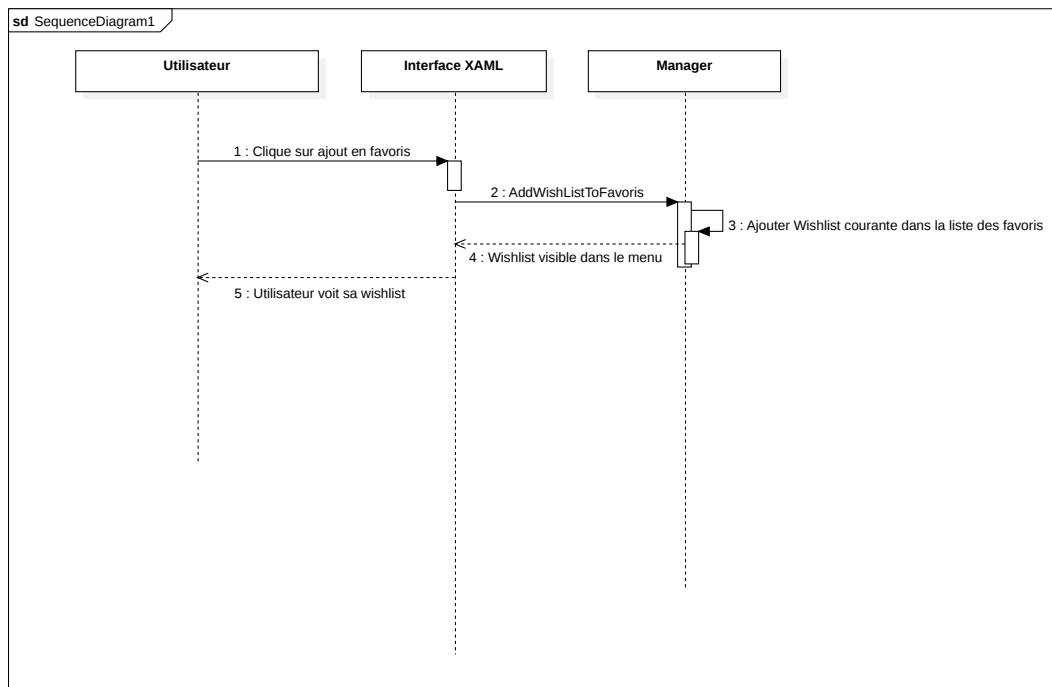


Ce diagramme de séquence met en avant les interactions des différentes classes du Modèle réalisant la persistance.

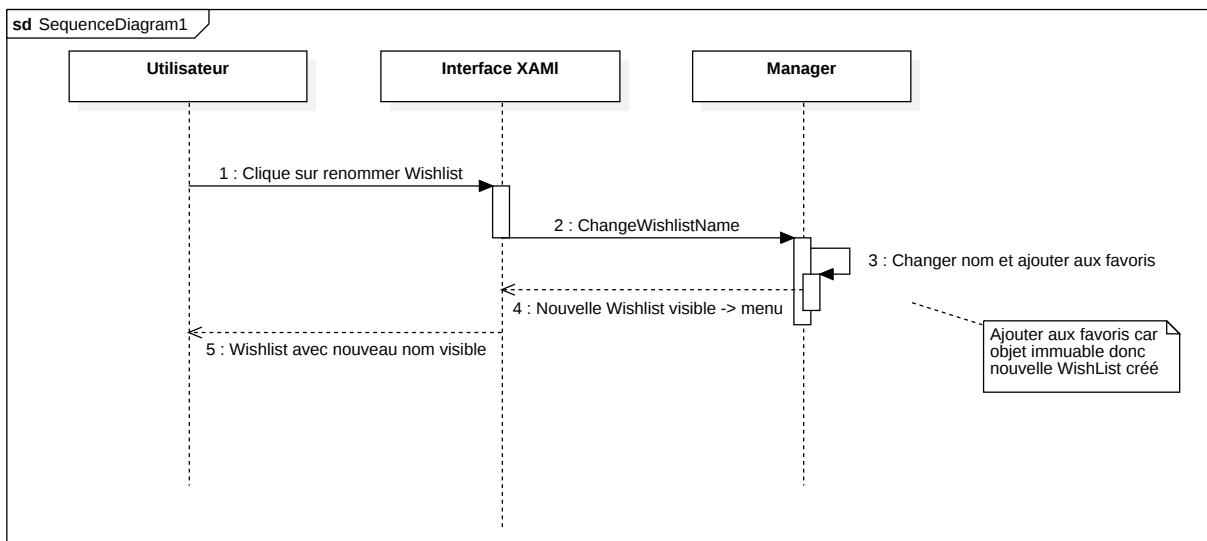
L'application instancie un Manager en lui injectant une dépendance de IPersistenceManager. Cette interface sera implémentée par un type concret qui sera soit un Stub soit unDataContract.

Lorsque l'application se lance, il faut charger les données. Pour se faire l'application appelle le IdbManager à charger les données, et ce, par le biais du Manager. Ce IdbManager, implanté par un type concret, créera des objets en lisant leurs données dans un fichier puis les retournera au Manager pour que les classes dépendantes de modèle puissent les utiliser.

Lorsque l'utilisateur enregistre une Whishlist en Favoris ou la renomme, il faut sauvegarder ses données. Pour se faire l'application appelle le IdbManager à sauvegarder les données, et ce, par le biais du Manager. Une fois de plus, le type concret de cette interface écrira les données en fichier.

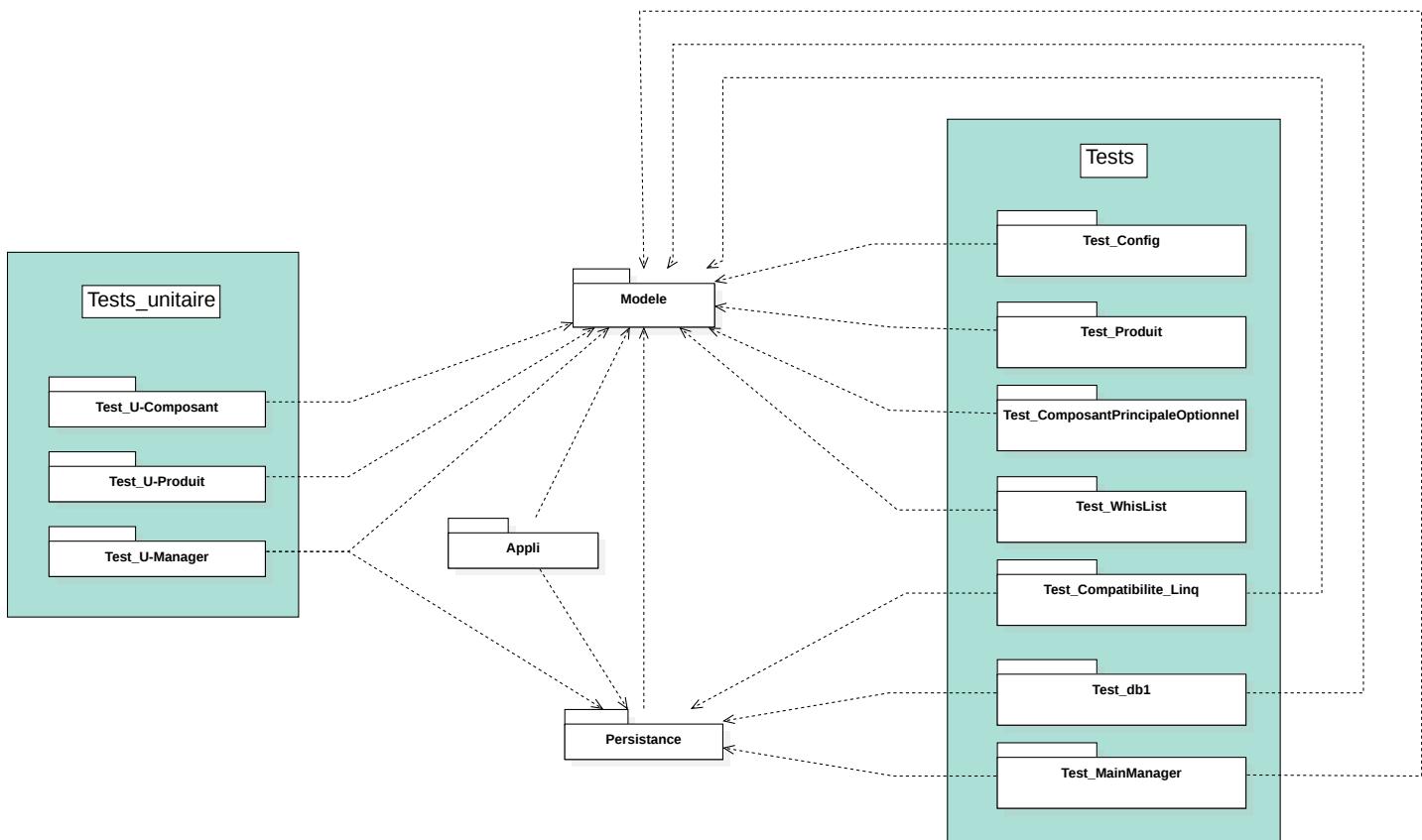


Ce diagramme de séquence met en avant le cas d'un ajout de Wishlist en favoris par un utilisateur. Lorsqu'il clique sur le bouton d'ajout au favoris, le Manager de l'application courante ajoute sa WishList courante dans la liste de favoris. À cet instant une sauvegarde en fichier de persistance est réalisée. Celle-ci est détaillée dans le diagramme précédent. L'utilisateur peut voir sa nouvelle WishList Favorite dans le menu.



Ce diagramme de séquence met en avant le cas d'un renommage de Wishlist en favoris par un utilisateur. Lorsqu'il clique sur le bouton de renommage, il peut modifier le nom. Ce nom sera modifié par le Manager de l'application courante qui recréa une nouvelle wishlist car tous les objets de notre modèle sont immuables. Une fois cette nouvelle Wishlist créée, il la verra dans ses favoris et elle sera sauvegarder en fichier.

4- DIAGRAMME DE PAQUETAGE



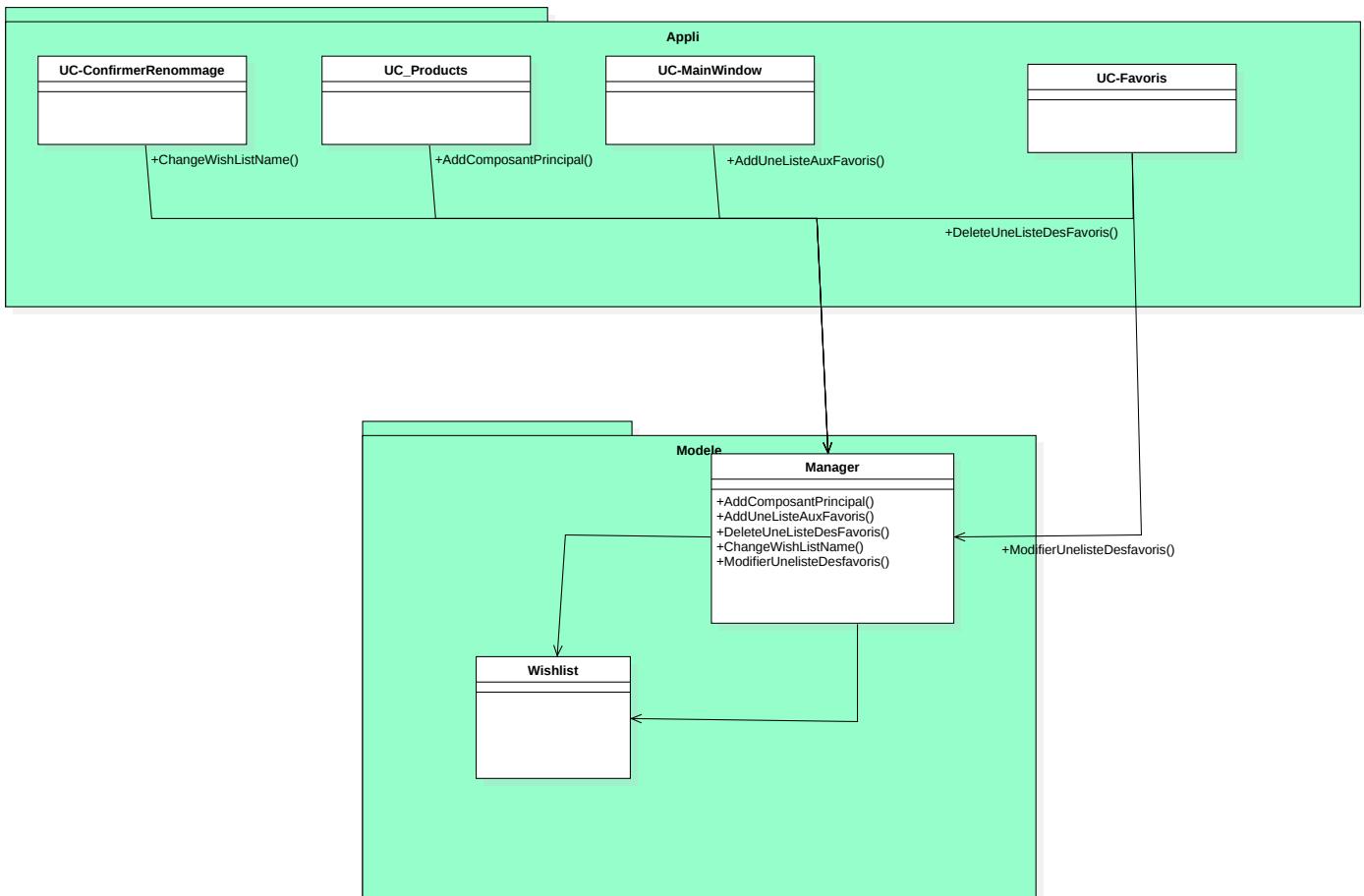
Voici notre diagramme de paquetage. Il nous permet d'avoir un aperçu visuel de l'architecture de notre projet. Les 2 rectangles en bleu représentent des répertoires : Un qui regroupe les tests fonctionnels (Tests) et l'autre, "Test_unitaire" qui regroupe l'ensemble de nos tests unitaires. A cela se rajoute trois autres packages : Appli, Modele et Persistance.

Pour commencer Modele regroupe presque l'ensemble des classes C# (Modele étant une bibliothèque de classes).

Ensuite nous avons appli, qui est regroupe la partie View. C'est une application Wpf. Cette partie contient aussi la classe "Navigator".

Enfin le dernier package "Persistance" qui contient l'ensemble des classes s'occupant de la persistance mais aussi de charger les données de l'application soit grâce à un stub soit en lisant dans un fichier.

5 - PATRON DE CONCEPTION ET ARCHITECTURE



L'architecture globale de notre application se base sur un modèle de conception dit "Facade". Notre classe Manager gère tout les objets et classes du modèle et les mets en relations avec celles hors de ce modèle. Le Manager est donc le pilier central. Il gère les données d'affichage, les données de la logique ainsi que la base de données. Toutes interactions entre objets passe forcément par des méthodes implémentées par le manager.