

Présentation du projet « Musée » :

- Description du contexte :

Nous avons eu l'idée de créer un programme informatique nommé « Musée ». Son objectif est de permettre à un organisme, tel qu'un musée, de gérer son inventaire de manière numérique et organisée. En partant de ce principe qui est certainement déjà exploité, nous avons voulu étoffer nos possibilités. Pour cela, le programme qui sert d'inventaire pourrait être mis à la disposition des visiteurs, à l'aide de bornes dans les salles ou par le biais des smartphones. L'avantage de mettre ce programme à la disposition du public pourrait être d'éviter une surcharge des guides en période d'affluence touristique.

On pourrait aussi imaginer que l'administration d'un tel organisme utilise le programme uniquement pour que les guides aient un support afin d'étoffer leurs connaissances sur les éléments de l'organisme, même ceux parfois temporaires comme lors d'expositions.




Une autre utilisation envisageable serait de donner le choix aux visiteurs de pouvoir visiter le musée à leur rythme à l'aide de ce programme. Il leur donnerait toutes les informations principales que l'administrateur jugerait utile de partager au public concernant chaque élément. Cela pourrait attirer plus de visiteurs, ceux qui ne visitent pas de lieux culturels car ils estiment qu'ils ont besoin d'un guide. En effet, un guide est intéressant car il connaît chaque élément d'un musée et peut donner des informations très détaillées sur ces derniers. Ce logiciel serait donc une solution adéquate pour certains visiteurs qui ne peuvent pas s'offrir un guide quelle que soit la raison.

Nous voulons que les utilisateurs aient tout de même une certaine interaction avec l'application, pour cela, l'intégration d'une galerie de médias devrait être disponible pour chaque élément ajouté à la base de donnée. Ainsi, les visiteurs pourront donc avoir des photos d'animaux, des images de reconstitution pour des animaux ayant vécu il y a longtemps, des sons ou même des vidéos afin que le visiteur puisse s'immerger dans la visite d'un musée. L'intégration d'une page dédiée aux commentaires et à la notation de l'organisme et/ou de l'application pourrait être envisagée.

Nous espérons aussi nous pencher sur l'accessibilité du programme afin d'élargir la cible des potentiels utilisateurs avec pourquoi pas, l'ajout d'un choix de la langue ? L'objectif est, qu'à partir de ce programme, n'importe quel organisme comme un parc animalier puisse aussi utiliser ce programme et le rendre accessible aux soigneurs par exemple. Élargir les capacités du programme tout en gardant une interface intuitive pour tous est important.

La partie d'administration ne sera pas montrée comme l'élément principalement disponible au lancement du programme. Il ne s'agit pas du but principal mais l'administration est nécessaire car elle est la base du programme si l'on veut qu'il fonctionne. Pour cela, le ou les administrateurs enregistrés devront s'authentifier à l'aide d'un identifiant et d'un mot de passe pour ensuite gérer la base de données comme il le souhaiteront.

- Sketch :

| Musée | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| British Museum 1) | |
| Adresse: Great Russell St, London WC1B 3DG | Téléphone: +44 20 7323 |
| Recherche rapide : <div>Saisir ici 2)</div> <div>Collection 3)</div> <div> Abeille Ane Ara Antilope Autruche Baleine Relette 4) </div> | <div>Abeille 5)</div> <div> Nom : Abeille Nom Scientifique : Antophila Espèce : Apoïdes Origine : Europe Nourriture : Flore </div> <div>    6) </div> <div> Le mot « abeille » est attesté en français pour la première fois au xive siècle⁴. D'abord mentionné sous les formes abueille, abele, aboille ou encore abeulle, ce mot est un emprunt à l'occitan abelha⁴. E. lui-même issu du latin āpīlla. </div> |
| Le British Museum (qui veut dire « Musée britannique », mais on utilise généralement le nom original), est un musée de l'histoire et de la culture humaine, situé à Londres, au Royaume-Uni. Ses collections, constituées de plus de sept millions d'objets, sont parmi les plus importantes du monde et proviennent de tous les continents. Elles illustrent l'histoire humaine de ses débuts à aujourd'hui. 7) | |
| Administration 8) | |

1) Bandeau de présentation

Dans ce bandeau de présentation générale, l'administrateur peut rentrer des informations essentielles pour un visiteur : le nom du musée, son numéro de téléphone (pour demander des renseignements divers) et son adresse (afin de s'y rendre). La modification des éléments contenus dans ce bandeau de présentation est accessible depuis le menu Administration (voir n°8) par les administrateurs des musées.

2) Barre de recherche rapide

Cette barre de recherche va permettre à un visiteur de rechercher rapidement un élément dans l'ensemble des différentes collections. En effet, quand il commence à taper les premières lettres de l'élément recherché, alors la liste des éléments (n°4) se met à jour.

3) Choix des collections

Ce choix s'effectue via une liste déroulante qui contient une quantité de collections. Une fois une collection sélectionnée, la liste (n°4) se met à jour.

4) Liste des éléments

La liste des éléments contient l'ensemble des éléments présents dans les différentes collections. Cette liste se met à jour à l'aide de la recherche rapide (n°2) ou bien à l'aide de la liste déroulante (n°3).

5) Vue de l'élément

La vue de l'élément va permettre d'obtenir l'ensemble des informations relatives à un élément sélectionné dans la liste des éléments (n°4). Elle est composée d'un premier cadre contenant le nom scientifique, l'espèce, l'origine, la nourriture (pour un animal) ou les matériaux utilisés (pour une œuvre d'art). Puis d'une galerie d'image (n°6) illustrant l'élément. Enfin, elle est composée d'un bandeau de description pour compléter l'ensemble des informations données auparavant.

6) Images (ou galerie) d'un élément

Cette galerie est contenue dans la vue de l'élément (n°5). Elle est composée d'une image principale puis d'un menu de navigation avec des miniatures des images suivantes. L'agrandissement d'une image se fait par un simple clic sur cette-dernière.

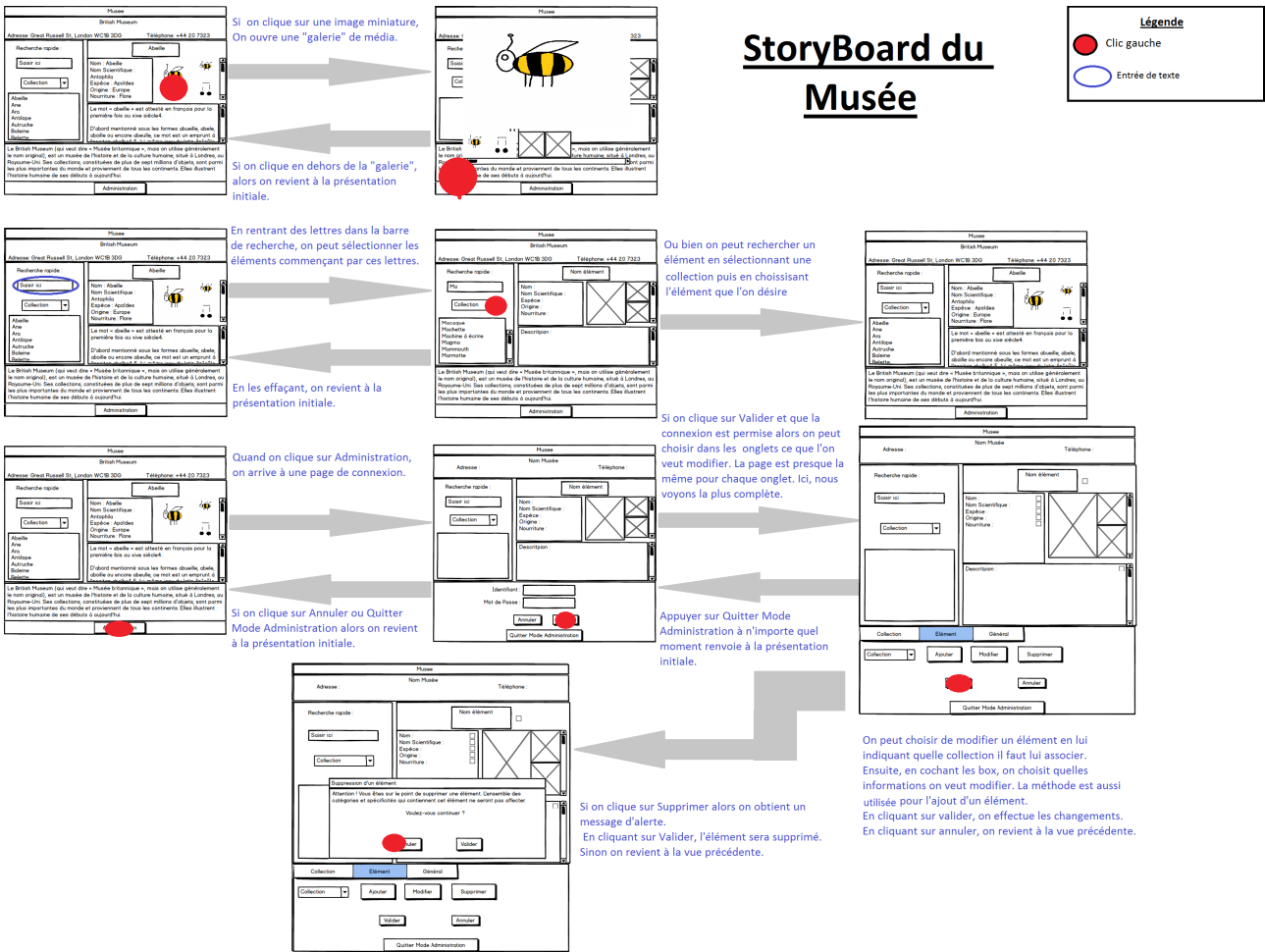
7) Bandeau de description du musée

Il s'agit d'un espace réservé à l'administrateur du musée afin qu'il puisse donner des informations complémentaires sur le musée (par exemple les expositions actuelles, les dernières nouveautés, les statistiques,...). Ce bandeau est situé entre la vue de l'élément (n°7) et le bouton Administration (n°8).

8) Bouton Administration

Ce bouton situé sous le bandeau de la description du musée (n°7) est le moyen d'accès au mode Administration pour un administrateur. Il permet, une fois l'identifiant rentré et le mot de passe correspondant, la modification (ou la mise à jour) de l'ensemble des informations données : éléments, collections, images, description, informations du musée. Attention, ce mode n'est disponible que pour les personnes accréditées Administrateur.

- Story-board :



- **Diagramme de cas d'utilisation :**



Explications :

Dans ce diagramme de cas d'utilisation, on remarque que l'utilisateur dispose de plusieurs utilisations qui s'offrent à lui au niveau du programme. L'administrateur est une spécialisation de l'utilisateur, il aura donc une utilisation supplémentaire du programme qui s'avère être le pilier du programme.

Cas « Faire une recherche » :

Description du cas : Le principe est que l'acteur aura différentes possibilités pour rechercher un élément du musée.

Auteur(s) : Utilisateur et Administrateur

Conditions requises : Sélectionner un élément dans l'ensemble des éléments proposés par le système.

Conséquences : Avoir accès aux informations complètes d'un élément comme : sa description, ses caractéristiques et la galerie de médias de ce dernier.

Séquence d'événements : L'acteur pourra directement sélectionner l'élément dans l'ensemble de tous les éléments existants, taper le nom associé à l'élément qu'il recherche ou bien sélectionner la collection dans laquelle se trouve l'élément.

Cas « Visualiser/Écouter un média » :

Description du cas : Les différents acteurs auront l'occasion de visualiser des images ou bien d'écouter des pistes audio concernant l'élément de leur choix.

Auteur(s) : Utilisateur et Administrateur

Conditions requises : Avoir au préalable sélectionné un élément dans la liste.

Conséquences : Permettre une meilleure interactivité avec les utilisateurs afin qu'ils puissent profiter de sons et d'images pour l'élément. Les différents utilisateurs auront donc une sensation d'immersion appuyée par les images et sons de chaque élément.

Séquence d'événements : L'utilisateur peut donc choisir parmi une liste de médias celui qui l'intéresse et, selon le type de média, interagir avec. S'il s'agit d'une piste audio, l'utilisateur peut démarrer l'écoute en cliquant dessus. Il peut aussi agrandir le média sélectionné avec le mode galerie pour une meilleure résolution d'image.

Cas « Administrer les données » :

Description du cas : L'acteur dispose de la fonctionnalité la plus importante : la gestion des données.

Auteur(s) : Administrateur

Conditions requises : Il faut que l'acteur se soit authentifié et soit reconnu en tant qu'administrateur.

Conséquences : Cela permet de déléguer une administration totale à/aux administrateur(s) afin qu'ils puissent s'occuper à tout moment des données de l'application.

Séquence d'événements : L'acteur a accès aux informations concernant l'organisme et peut ainsi les éditer en fonction de son choix : le nom, l'adresse, le mail, le numéro de téléphone ou la description(fonctionnement) de son organisme. En dehors des données liées à l'organisme, il peut faire le choix parmi 2 autres types de données : les collections ou les éléments. À partir de ces 2 différents genres, il a la possibilité de créer ou de supprimer n'importe quelle donnée liée à un des genres. Puis, selon le genre, différentes options d'édition sont possibles comme pour une collection : l'ajout d'un ou de plusieurs éléments à celle-ci par exemple. Enfin, pour l'édition d'un élément, il peut choisir de modifier : son nom, sa liste de médias, sa description et/ou ses caractéristiques.

Cas « Gérer les collections » :

Description du cas : L'acteur peut gérer la liste des collections de son organisme afin de pouvoir le mettre à jour si nécessaire.

Auteur(s) : Administrateur

Conditions requises : Il faut que l'acteur se soit authentifié et soit reconnu en tant qu'administrateur.

Conséquences : L'acteur a alors toute possibilité concernant les données sur les collections.

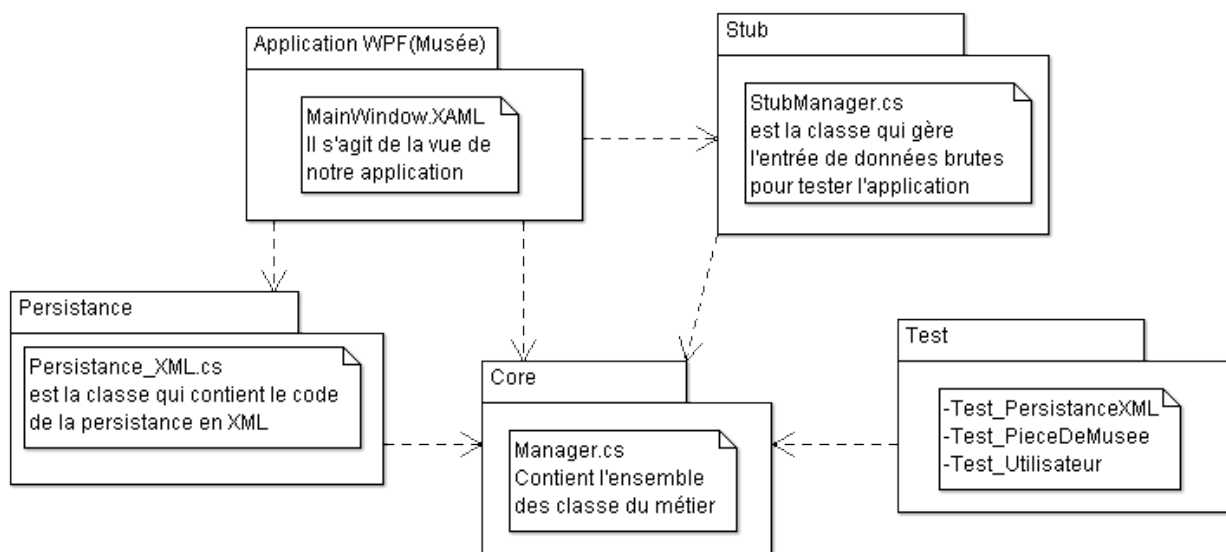
Séquence d'événements : Après s'être authentifié et une fois que le système aura vérifié son statut d'administrateur. L'acteur pourra donc choisir de gérer les collections. Il pourra donc faire le choix de créer une nouvelle collection en lui attribuant un nom et en lui ajoutant des éléments ou des collections qui existent déjà dans la base de donnée du système. Il peut aussi faire le choix de supprimer une collection car elle était temporaire ou bien qu'elle n'a plus lieu d'être dans l'application. Enfin, il peut aussi faire le choix d'éditer une collection et donc modifier son nom et la liste des collections et éléments qui la constitue.

- **Considérations ergonomiques**

Notre application est issue d'un Master-Detail de base avec un système de navigation/recherche à gauche et une vue à droite de l'élément recherché lors de la navigation. La disposition des différents éléments de notre application a été réalisée par nos fins.

Nous avons décidé, pour la partie Administration, de la gérer dans un bandeau situé en-dessous du Master-Detail. En effet, nous avons trouvé plus judicieux et plus moderne d'effectuer la gestion des données dans la même fenêtre que l'application elle-même et ainsi éviter l'ouverture d'une multitude de fenêtre. Nous faisons seulement afficher une demande de validation via une boîte de dialogue pour valider ou non une modification sur les éléments d'un musée.

- **Diagramme de paquetage :**



Description :

Voici le diagramme de paquetage de l'application Musée. Pour l'instant, il comporte 5 assemblages différents qui vont interagir ensemble afin de faire fonctionner le programme principal. L'intérêt de créer différents assemblages est de pouvoir séparer le programme en plusieurs parties et de le rendre plus flexible.

Le premier assemblage présenté est celui qui a pour nom « Core », il s'agira du noyau central de l'application. A l'intérieur de cet assemblage réside tout le code regroupant le « Métier » du projet. Ce regroupement servira à assurer le bon fonctionnement de l'application. En effet, les classes majeures du Core sont les classes Manager, Utilisateur et Collection.

Le second assemblage est nommé « Application WPF », il comportera toutes les vues qui existent pour le projet. Ce regroupement permet d'avoir toutes les différentes vues que l'on peut donner au projet. L'assemblage a une dépendance envers un autre assemblage : le Core. Cette relation permet donc de relier la partie métier avec les différentes vues disponibles et d'interagir avec.

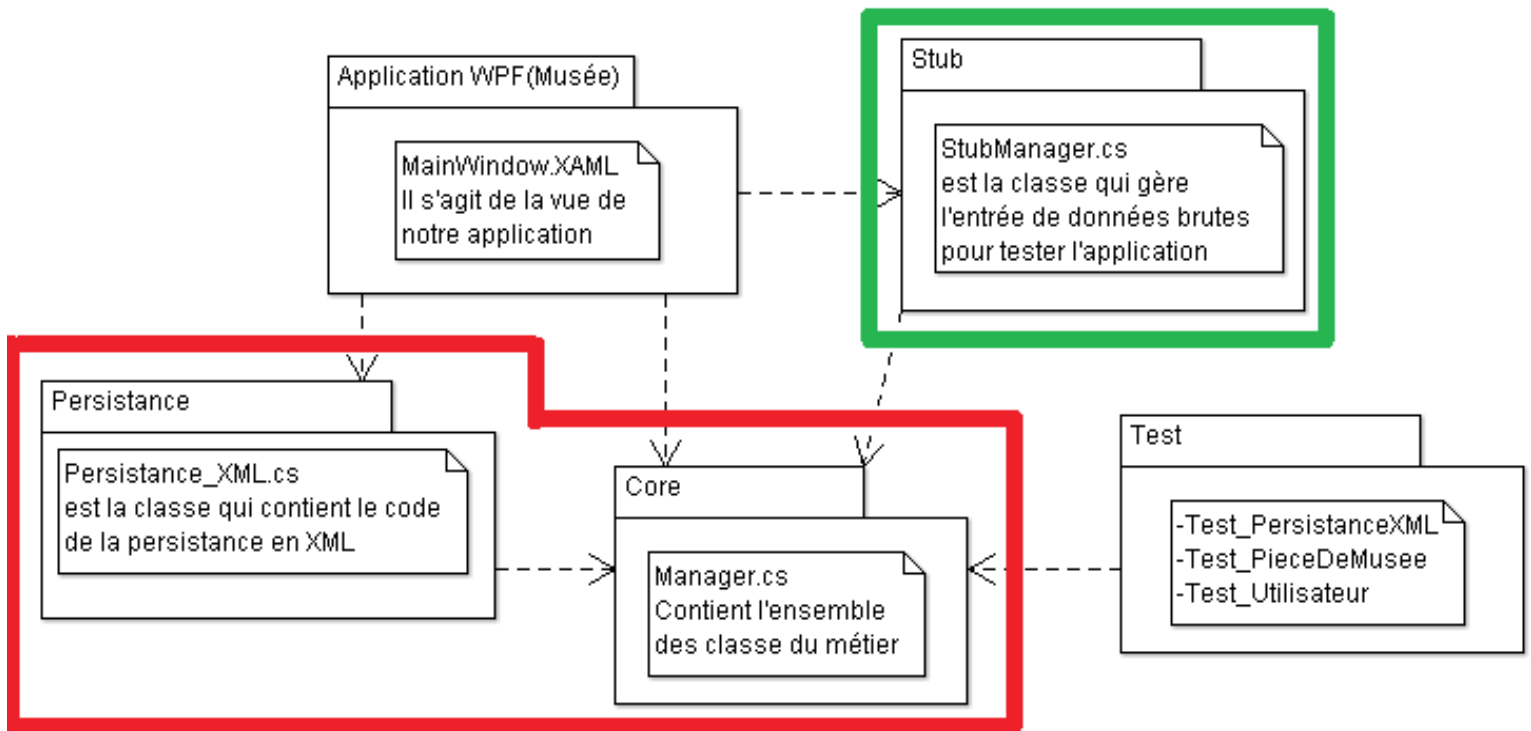
Le troisième assemblage ayant pour nom « Test » contient tous les tests applicatifs créés par le programmeur afin de vérifier le bon fonctionnement du programme. Il a une relation de dépendance avec le Core ce qui permet donc à tous les tests d'utiliser les classes du Core pour vérifier qu'il n'y ait aucun problème avec l'exécution de toutes les méthodes du Core. Cet assemblage est essentiel lors du développement de la partie métier du projet (Core) car il assure par de nombreux tests variés l'efficacité du programme et permet de détecter les potentielles sources d'erreurs.

Le quatrième assemblage contient l'entrée de données brutes à fin de pouvoir tester la vue et nos

classes en collaboration avec le package Tests.

Le dernier assemblage est composé de notre persistance en XML et contient la classe `Persistence_XML.cs`. Le but de cette classe est de sérialiser et dé-sérialiser nos objets afin de les sauvegarder en XML et de pouvoir les charger à tout instant.

- **Diagramme de paquetage mettant en avant la persistance :**



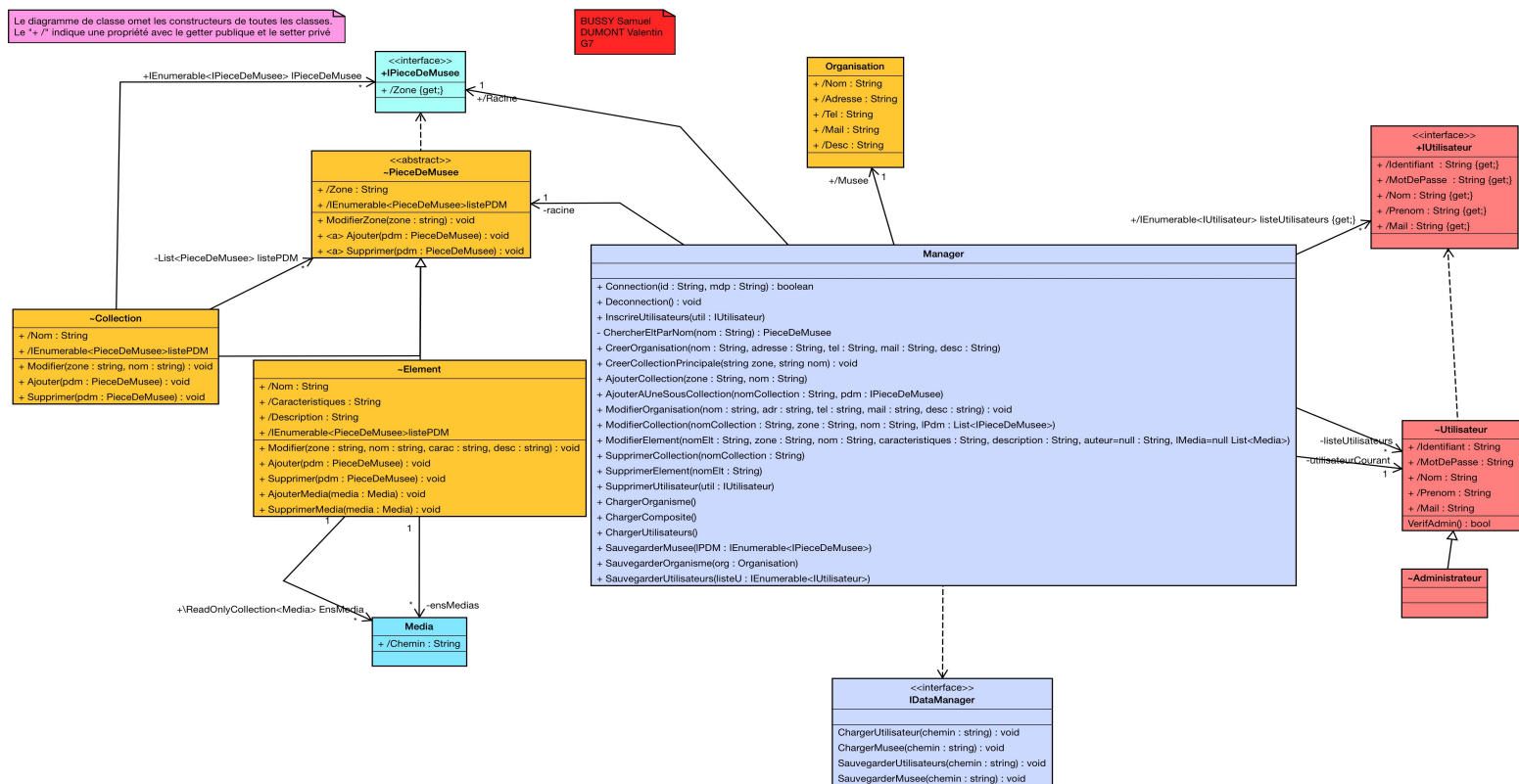
Description :

Ce diagramme de paquetage précise la partie persistance. Encadrés en rouge nous retrouvons 2 paquets qui vont constituer la persistance en XML de notre projet. Encadré en vert il s'agit du Stub qui nous permet de rentrer en brut des données dans notre application.

Le 1er paquetage, « Persistence », contient la classe `Persistence_XML.cs` qui contient l'ensemble des méthodes de sauvegarde et de chargement (qui sont précisées dans le diagramme de classe mettant en avant la partie persistance dans ce document) des données du musée. Nous utilisons le système de sérialisation/dé-sérialisation pour sauvegarder chaque objet dont le musée a besoin pour fonctionner en XML dans différents fichiers (« `piecedemusee.xml` » qui contient l'ensemble des collections et des éléments d'un musée, « `organisme.xml` » qui contient les informations d'un musée et « `utilisateur.xml` » qui contient l'ensemble des utilisateurs/administrateurs d'un musée).

Ainsi, dans le 2ème paquetage qui constitue le métier de notre application, nous retrouvons des `[DataContract]` et des `[DataMember]` (détaillés dans le diagramme de classe mettant en avant la partie persistance) sur l'ensemble des classes que nous avons besoin de sérialiser et donc de sauvegarder pour le bon fonctionnement de l'application.

- Diagramme de classe :



Description :

Le diagramme de classe est composé de différentes parties que nous pouvons distinguer à l'aide des différentes couleurs des classes. Nous allons donc voir quelles sont ces différentes parties et quelles sont les classes qui les composent.

En premier, nous avons la partie Utilisateur, qui contient les classes « Utilisateur » et « Administrateur » et une interface « IUtilisateur », qui va permettre de gérer les autorisations accordées à un utilisateur du programme selon s'il est administrateur ou simple utilisateur. Un utilisateur basique correspond à la classe « Utilisateur » et possède des propriétés standards : Identifiant, Mot de passe, Nom, Prénom et Mail. Si l'utilisateur correspond à la classe « Administrateur » alors il aura accès aux différentes méthodes disponibles dans la classe « Manager ».

Cette dernière est une composante de la 2ème partie de notre diagramme : la partie de gestion entre l'application et nos classes. Nous utilisons le patron dit « Façade » dont sa mise en place permet d'éviter à l'utilisateur du programme d'utiliser directement les méthodes de nos classes et donc ajoute un niveau de sécurité. En effet, il sera le principal point d'accès au métier pour les autres paquets (Vue, Persistance...). La classe « Manager » contient l'ensemble des méthodes d'ajout, de suppression, de modification des éléments et collections ainsi que le système de connexion et déconnexion des personnes utilisant le programme. Cette classe Manager nous permet d'avoir une passerelle entre notre code et notre application afin de protéger les modifications qui pourraient directement être effectuées sur nos classes. Nous limitons ainsi le risque d'erreur produite par un client dans notre code.

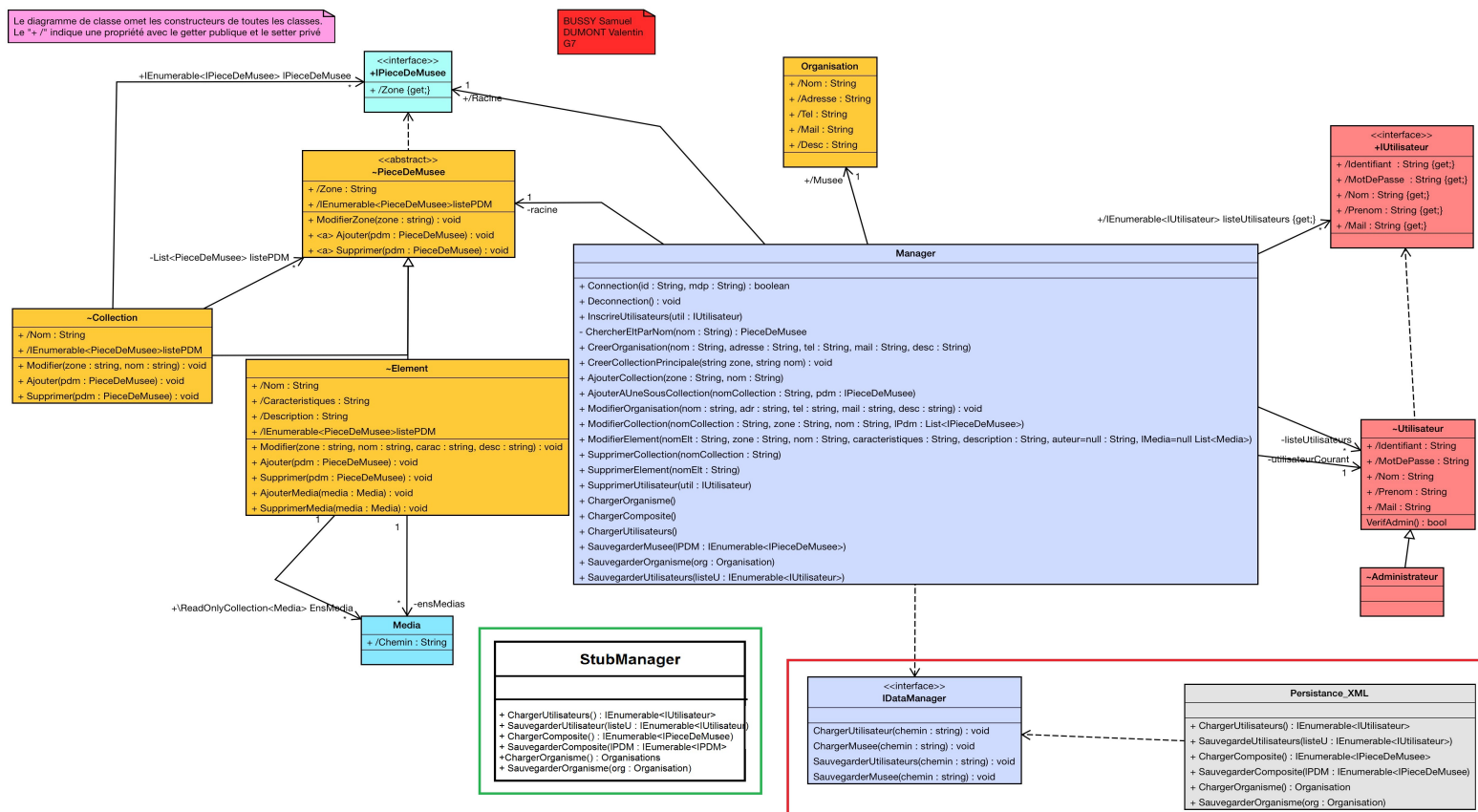
La classe « IDataManager » nous servira pour la persistance et aux tests.

Nous avons aussi une partie importante, constituée des classes essentielles de notre programme : « Organisation » (qui représente un musée avec son nom, son adresse, son téléphone, une description), une interface « IPieceDeMusee » pour l'encapsulation des éléments contenus dans les classes héritées de PieceDeMusee, une classe « PieceDeMusee », qui est abstraite, (qui va nous permettre de gérer l'ensemble de nos collections et éléments par zone), une classe « Collection » qui redéfinit les méthodes de « PieceDeMusee » (qui sont les méthodes utilisées par le manager pour gérer le musée) et une classe « Élément ». On remarque que les 3 classes « PieceDeMusee », « Collection » et « Élément » sont disposées selon le patron de conception structurel « Composite », que l'on peut représenter comme une structure d'arbre. Ce patron nous permet de trouver une solution afin de gérer nos listes d'élément et de collection. En effet, nous utilisons un objet composite (notre PieceDeMusee) qui est constitué de 2 objets similaires (Collection et Élément), le but est de manipuler ce groupe d'objets comme s'il ne s'agissait que d'un seul objet. L'ensemble des objets contenus dans ce patron de conception ont ainsi des méthodes communes. Le composant (qui est l'abstraction de tous les composants) est donc PieceDeMusee, la feuille (qui représente un composant n'ayant pas de sous-éléments) est Élément et le composite (qui représente un composant pouvant avoir des sous-éléments) est Collection. Notre classe « Collection » possède aussi une liste IEnumerable de « PieceDeMusee ».

Notre dernière partie est la partie média d'un élément. En effet, un élément peut avoir des médias (audio et/ou photo) qui sont représentés par la classe « Media » qui a comme unique propriété le chemin d'accès au média.

L'ensemble des interfaces (IDataManager, IPieceDeMusee et IUtilisateur) présentes dans le Core nous permet d'ajouter une couche de protection envers les classes qui implémentent ces interfaces. Ainsi dans le Manager, quand nous avons besoin d'une méthode d'une classe principale, nous appelons la méthode de l'interface qui ira elle-même chercher la méthode dont le Manager aura besoin. Le Manager ne peut donc ainsi pas directement modifier une des classes importantes de notre code. Il s'agit du patron de conception Façade.

- Diagramme de classe mettant en avant le persistance :



Dans ce diagramme de classe (qui est le même que le précédent mais avec une classe Persistence_XML en plus issue du paquetage Persistence), nous pouvons voir comment la persistance est-elle réalisée sur nos classes.

Étant donné que nous avons choisi une persistance en XML, il nous a fallu ajouter des [DataContract] et [DataMember] dans les classes que nous voulons sérialiser : PieceDeMusee, Collection, Elément, Utilisateur, Administrateur, Organisation. Chaque classe contient des propriétés auxquelles nous avons appliqué les [DataMember] dont a besoin XML pour fonctionner.

Afin de renforcer la protection du Manager, la classe IDataManager intervient comme une passerelle entre la classe Persistence_XML et le Manager qui détient les méthodes d'appel des méthodes contenues dans Persistence_XML. Ainsi l'interface va elle-même appeler ces méthodes et donc le Manager est protégé.

Il en est de même entre IDataManager et StubManager qui va nous permettre de rentrer en brut des données dans notre application.