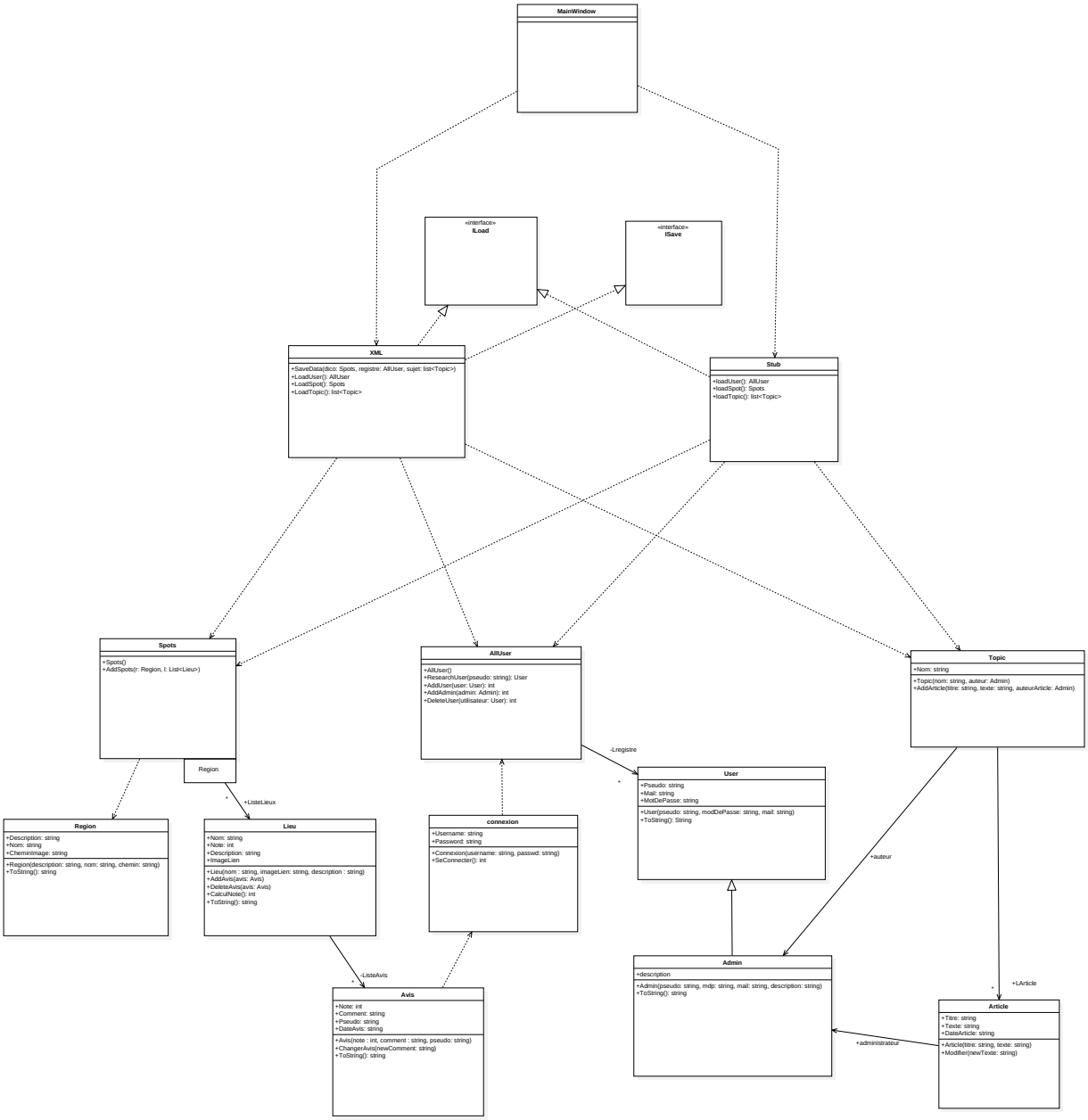
Model::Main


MainWindow
«interface»
ILoad
«interface»
ISave
XML
+SaveData(dico: Spots, registre: AllUser, sujet: list<Topic>)
+LoadUser(): AllUser
+LoadSpot(): Spots
+LoadTopic(): list<Topic>
Stub
+loadUser(): AllUser
+loadSpot(): Spots
+loadTopic(): list<Topic>
Spots
+Spots()
+AddSpots(r: Region, l: List<Lieu>)
AllUser
+AllUser()
+ResearchUser(pseudo: string): User
+AddUser(user: User): int
+AddAdmin(admin: Admin): int
+DeleteUser(utilisateur: User): int
Topic
+Nom: string
+Topic(nom: string, auteur: Admin)
+AddArticle(titre: string, texte: string, auteurArticle: Admin)
Region
-Lregistre
User
+Pseudo: string
+Mail: string
+MotDePasse: string
+User(pseudo: string, modDePasse: string, mail: string)
+ToString(): String
Region
+Description: string
+Nom: string
+CheminImage: string
+Region(description: string, nom: string, chemin: string)
+ToString(): string
+ListeLieux
Lieu
+Nom: string
+Note: int
+Description: string
+ImageLien
+Lieu(nom : string, imageLien: string, description : string)
+AddAvis(avis: Avis)
+DeleteAvis(avis: Avis)
+CalculNote(): int
+ToString(): string
connexion
+Username: string
+Password: string
+Connexion(username: string, passwd: string)
+SeConnecter(): int
+auteur
+ListeAvis
Avis
+Note: int
+Comment: string
+Pseudo: string
+DateAvis: string
+Avis(note : int, comment : string, pseudo: string)
+ChangerAvis(newComment: string)
+ToString(): string
Admin
+description
+Admin(pseudo: string, mdp: string, mail: string, description: string)
+ToString(): string
+LArticle
Article
+Titre: string
+Texte: string
+DataArticle: string
+Article(titre: string, texte: string)
+Modifier(newTexte: string)
+administrateur

Guillaume Cothenet
Yannis Mahiou

Description de l'UML

Spots : Spots est une classe contenant un dictionnaire qui a comme clé une Region et comme valeur une liste de Lieu. Elle possède deux méthodes : son constructeur et « AddSpots » qui permet d'ajouter une clé ainsi que sa valeur dans le dictionnaire. Elle utilise les méthodes de la classe XML ou stub afin de charger ou sauvegarder ses données. Nous avons décidé de regrouper nos spots dans un dictionnaire afin de faciliter leur « rangement » (chaque région a une liste de lieu) et on peut donc choisir facilement des lieux à proximité de l'endroit où l'on se trouve puisqu'ils sont « rangé » par région.

Region : Region est une classe contenant un nom, une description ainsi qu'un lien vers une image. Elle possède deux méthodes : son constructeur et une redéfinition de la méthode « ToString ». Nous avons choisi cette classe comme clé de notre dictionnaire puisqu'il nous paraissait logique de regroupé les lieux en fonction de leur endroit.

Lieu : Lieu est une classe contenant un nom, une note, une description, un lien vers une image et une liste d'avis. Elle possède 5 méthodes : son constructeur, AddAvis (qui permet d'ajouter un avis à sa liste), DeleteAvis (qui permet de supprimer un avis de sa liste), CalculNote (qui permet de faire une moyenne des notes ajoutées par les utilisateurs) et une redéfinition de « ToString ». C'est en quelque sorte le cœur de notre application puisque c'est ce que l'on présente dans notre détail.

Avis : Avis est une classe contenant une note, un commentaire, un pseudo et une date. Elle possède 3 méthodes : son constructeur, ChangerAvis (qui permet de changer un avis que l'on a mis) et une redéfinition de « ToString ». Cette classe est venue après la réalisation de la connexion, il nous paraissait logique d'ajouter un avantage aux utilisateur qui avait un compte.

User : User est une classe contenant un pseudo, un mot de passe et un mail. Elle possède 2 méthode : son constructeur et une redéfinition de « ToString ». Nous l'avons créé pour instaurer une sorte de hiérarchie entre les utilisateurs et pouvoir ajouter des fonctionnalités disponibles ou non en fonction de notre « grade ».

AllUser : AllUser est une classe contenant une liste d'User. Elle possède 5 méthodes : son constructeur, ResearchUser (qui prend en paramètre un string et qui return un User), AddUser (qui permet d'ajouter un utilisateur à sa liste), AddAdmin (qui permet d'ajouter un Admin à sa liste) et DeleteUser (qui permet de supprimer un User). Elle utilise les méthodes de la classe XML ou stub afin de charger ou sauvegarder ses données. Cette classe a été créée afin de stocker tous les utilisateurs de l'application et pouvoir parcourir cette liste à notre guise, pour d'autre méthodes.

Connexion : Connexion est une classe contenant un username et un password. Elle possède 2 méthodes : son constructeur et SeConnecter (qui permet de se connecter à l'application). Cette classe nous permet de savoir si un utilisateur est connecté ou non.

Admin : Admin est une classe fille d'User, contenant une description. Elle possède 2 méthodes : son constructeur et une redéfinition de « ToString ». On a décidé de la créer afin

de distinguer le simple utilisateur et l'admin. Elle nous permet aussi de rajouter certaines fonctionnalités accessibles uniquement aux Admin.

Article : Article est une classe contenant un titre, un texte, une date et un Admin. Elle possède 2 méthodes : son constructeur et Modifier (qui permet de modifier un article). On a créé cette classe afin de regrouper tous les textes que l'on voulait afficher dans notre application.

Topic : Topic est une classe contenant un nom, un Admin et une liste d'article. Elle possède 2 méthodes : son constructeur et AddArticle (qui permet d'ajouter un article à sa liste). Elle utilise les méthodes de la classe XML ou stub afin de charger ou sauvegarder ses données. Cette classe nous permet d'afficher une liste d'article dans une listebox (par exemple) avec un simple binding sur le Topic en question.

XML : XML est une classe qui implémente les interfaces ILoad et ISave. Elle possède 4 méthodes : SaveData (qui permet de serializer toutes les données, LoadUser (qui permet de charger la liste d'utilisateur), LoadSpot (qui permet de charger le dictionnaire de Spots) et LoadTopic (qui permet de charger une liste de Topic. Nous avons créé cette classe afin de sauvegarder dans des fichier les données de l'application.

Stub : Stub est une classe qui implémente l'interface ILoad. Elle possède 3 méthodes : LoadUser (qui permet de charger la liste d'utilisateur), LoadSpot (qui permet de charger le dictionnaire de Spots) et LoadTopic (qui permet de charger une liste de Topic. Cette classe à été créé afin de « remplir » l'application de données lors des différents tests effectués.

ILoad : ILoad est une interface qui a comme méthode LoadUser, LoadSpot et LoadTopic. Elle permet aux classes XML et Stub de pouvoir deserializer les données utilisées pour le projet.

ISave : ISave est une interface qui a comme méthode SaveData(). Elle permet à la classe XML de pouvoir serializer dans des fichiers .xml les données du projet.

Yannis Mahiou
Guillaume Cothenet

Description PackageDiagram :

Notre projet "UrbexProject" est composé de différents packages. On y retrouve la bibliothèque de classe, l'application console, les « data » ainsi que l'application UrbexProject.

La bibliothèque de classe (regroupe les principales les classes du projet):

Utilisateurs : Le package Utilisateurs regroupe toutes les classes qui concernent les utilisateurs du projet (User.cs, Admin.cs et AllUser.cs). Nous avons décidé de regrouper les classe Users et Admin car ce sont deux types d'utilisateurs possibles dans notre application. De plus, ce package contient la classe "AllUser", qui a pour but de regrouper tous les utilisateurs ;. Relier ces classes dans un même package permet de montrer que celles-ci font toutes partie de l'aspect utilisateur de notre application

SpotsUrbex : Le package SpotsUrbex contient 3 classes : "Spots", "Région", et "Lieu". Ce sont les classes qui sont fortement reliées puisque « Spots » contient un dictionnaire de « Région/Lieu ». Elles forment la fonctionnalité principale de notre projet : permettre à l'utilisateurs de trouver des lieux au travers de différentes régions afin de pratiquer l'Urbex. Par ce fort lien qui uni ces classes, nous avons décidé de les regrouper en un package.

FonctionnalitesUtilisateurs : Ce package contient la classe "Connexion", ainsi que la classe "Avis". Ce sont des classes qui permettent au utilisateurs du projet de bénéficier de plusieurs fonctionnalités, comme de pouvoir se connecter, ou de donner son avis sur un lieu d'Urbex. Nous les regroupons dans ce package car ces classes ont la même finalité : accorder aux utilisateurs de l'application des fonctionnalités.

Articles : Ce package contient la classe Article et la classe Topic. Ces deux classes sont liées : un topic est une liste d'articles. Les regrouper permet donc de regrouper cette fonctionnalité du projet, et de la séparer des autres packages.

Application Console (Application console du projet) :

Ce package contient la classe Program. Cette classe du projet permet de tester les différentes fonctionnalités que l'on peut faire en utilisant les classes de la bibliothèque de classes, au travers de la Console, sans les lier directement au model.

Data :

Ce package concerne tous ce qui touche au chargement et à la sauvegarde des données de notre application. Elle contient un classe « Stub » dans laquelle des données brutes sont mises, ainsi que la classe « XML » qui, elle, va charger et sauvegarder les données en passant par des fichiers. Il y a aussi deux interfaces qui sont implémentées dans les deux classes précédemment citées.

UrbexProject :

Ce package contient toute la partie model de l'application. Nous avons choisi de ne pas ranger certaines « windows » dans des packages car elles sont communes à tous les utilisateurs. Mais nous avons cependant créé d'autres packages afin de ranger les autres pages.

UsersControl : ce package contient une partie du model susceptible de changer lors d'une connexion d'un utilisateur. En effet, si un utilisateur non connecté ou un simple utilisateur connecté utilise l'application, une partie du model chargé sera ici. On a décidé de créer ce package afin de séparer l'interface d'un utilisateur lambda et celle d'un administrateur.

Image : ce package réuni toutes les photo/images dont l'application se sert. Il est séparé en deux : d'un côté des images quelconques, utilisé a diverse endroits dans l'application, et d'un autre package, ImageGalerie, contenant les images présentes dans la galerie. On a décidé de créer le package Image afin de regrouper en un seul endroit les images pour éviter un « fouillis » potentiel.

UsersControlAdmin : ce package contient une partie du model susceptible de changer lors d'une connexion d'un utilisateur. En effet, si un utilisateur Admin utilise l'application, une partie du model chargé sera ici. On a décidé de créer ce package afin de séparer l'interface d'un utilisateur lambda et celle d'un administrateur. Ce package contient aussi deux autres packages :

AccueilPagesModif : celui-ci contient les différentes pages qui vont s'afficher lorsqu'un administrateur fera des modifications sur l'onglet Accueil.

SpotsPagesModif : celui-ci contient les différentes pages qui vont s'afficher lorsqu'un administrateur fera des modifications sur l'onglet Spots.

Model::ClassDiagram1