Ajout Personnel

Nous avons décidé d'ajouter à notre projet une partie connexion. En effet, afin de donner un côté « communautaire » à notre application, nous avons choisi de développer une partie connexion et tout ce que cela engendre (différentes classes, …)

Nous avons donc ajouté à notre code des classes afin de rendre possible une connexion. Il y a la classe User qui représente un utilisateur lambda (connecté), la classe Admin qui représente les administrateurs de l'application, la classe AllUser qui est une liste de tous les utilisateurs (User et Admin confondu) et enfin la classe connexion permettant de gérer les connexions.

Etant donné la présence d'une connexion, lors du lancement de l'application, nous somme considérer comme une sorte de visiteur, sui ne peut que « regarder » l'application. L'utilisateur lambda, lui, peut ajouter un avis sur un lieu d'une région qu'il aura choisi au préalable. L'admin quant à lui, peu bien sur ajouter un avis, mais il peut aussi ajouter/supprimer un topic, un article, une région ou un lieu.

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.

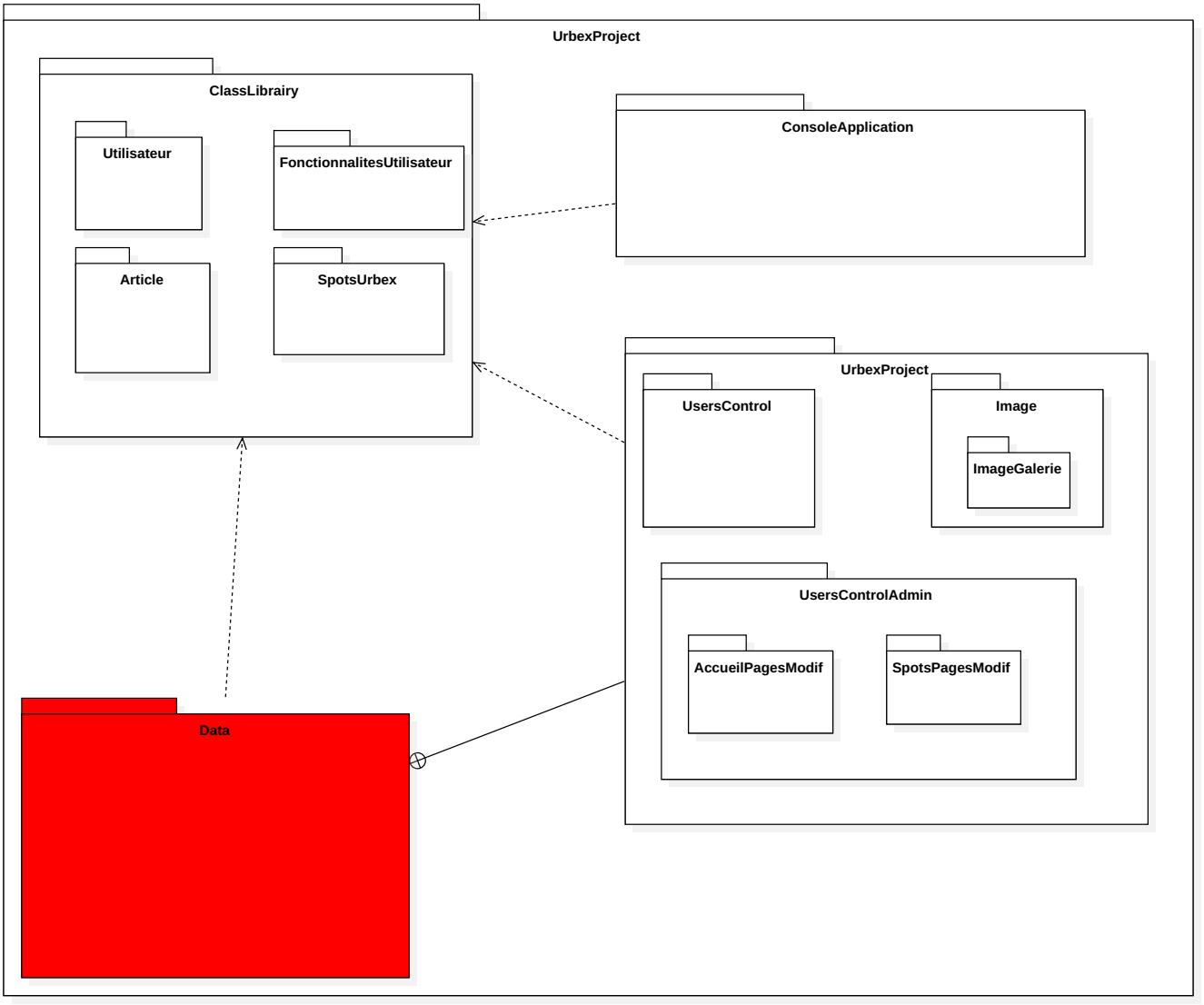
Image : ce package réuni toutes les photo/images dont l'application se sert. Il est séparé en deux : d'un côté des images quelconques, utilisé a diverse endroits dans l'application, et d'un autre package, ImageGalerie, contenant les images présentes dans la galerie. On a décidé de créer le package Image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

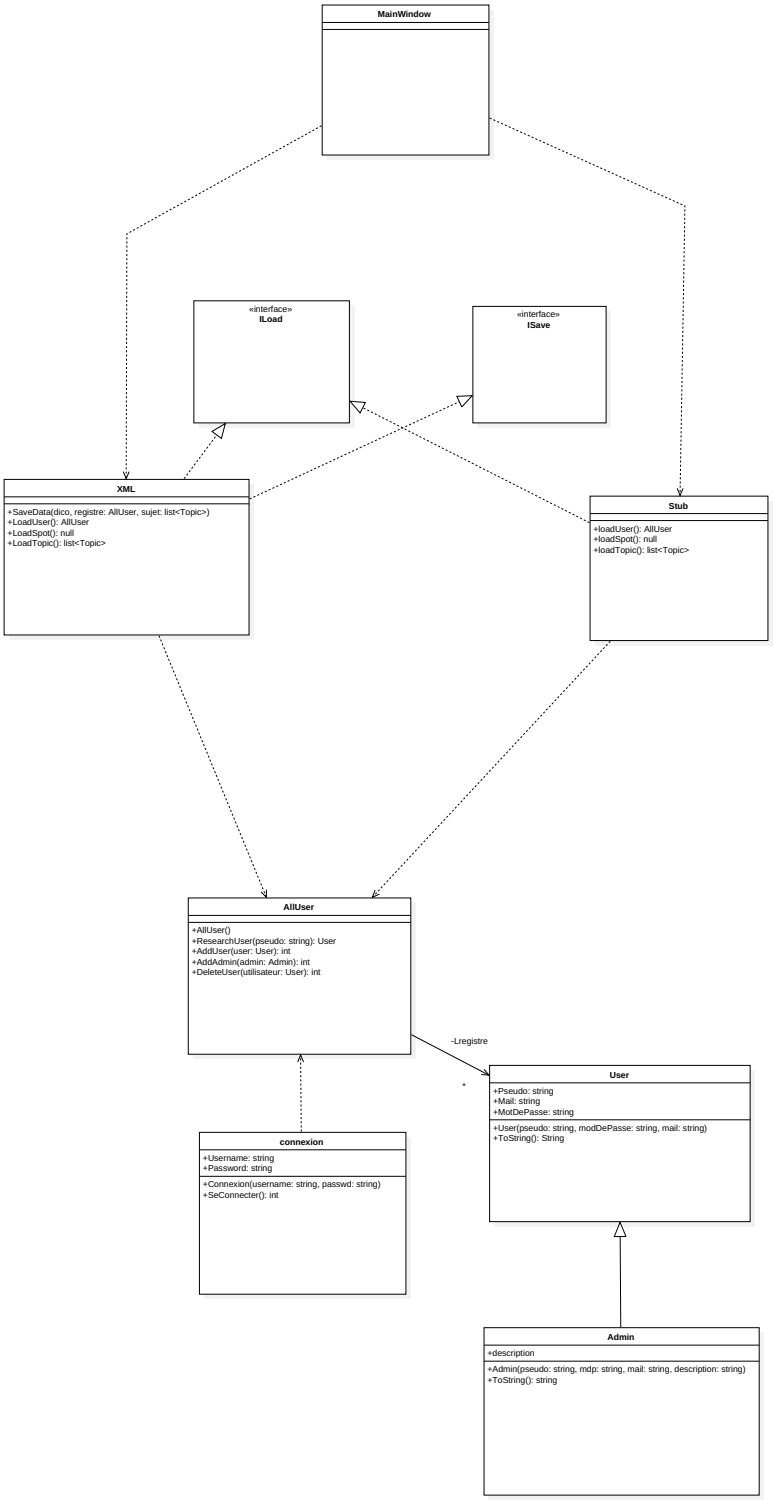
AccueilPagesModif : celui-ci contient les différentes pages qui vont s'afficher lorsqu'un administrateur fera des modifications sur l'onglet Accueil.

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

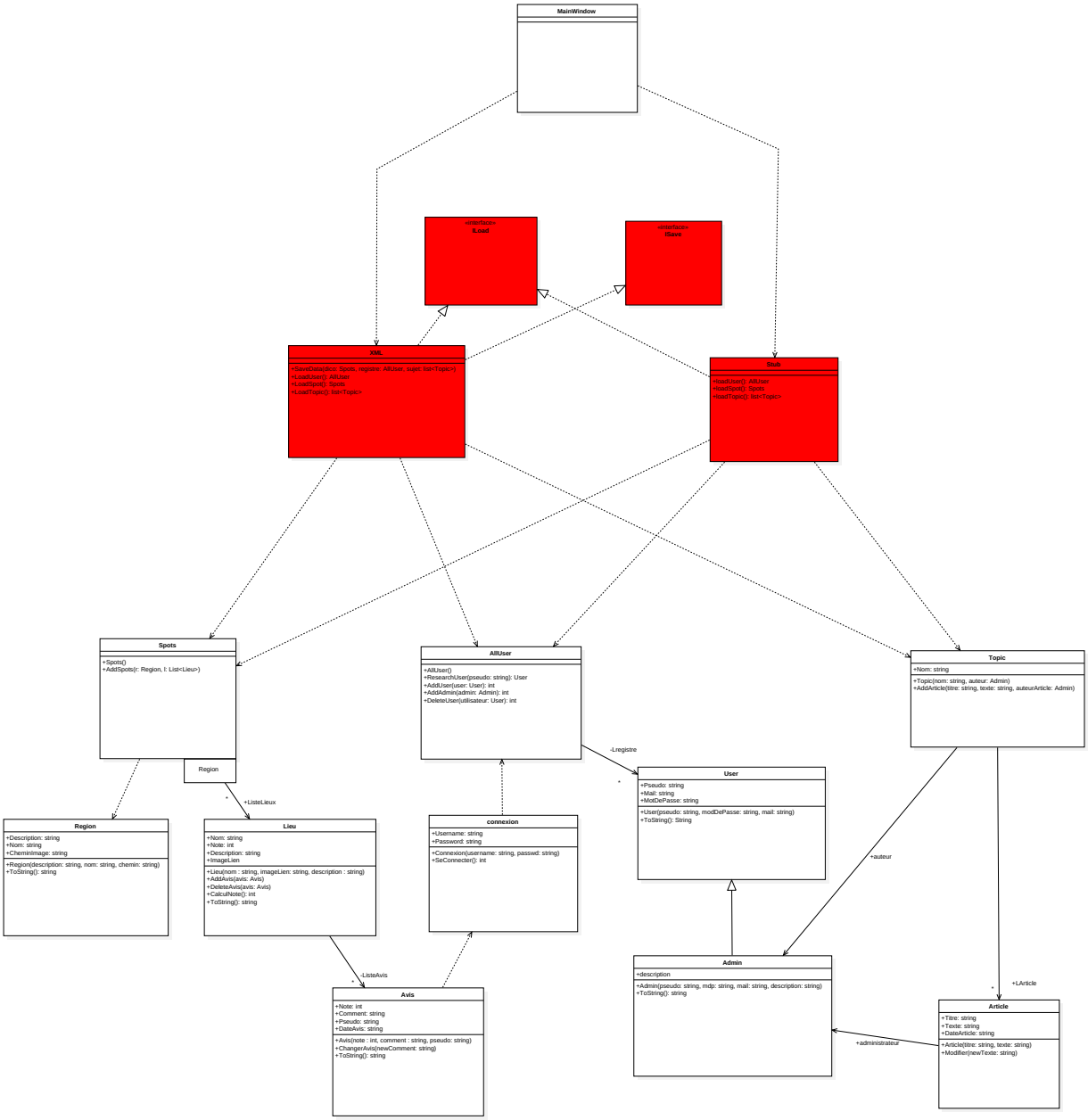
Model::ClassDiagram1



Model::Main


MainWindow
«interface»
ILoad
«interface»
ISave
XML
+SaveData(dico, registre: AllUser, sujet: list<Topic>)
+LoadUser(): AllUser
+LoadSpot(): null
+LoadTopic(): list<Topic>
Stub
+loadUser(): AllUser
+loadSpot(): null
+loadTopic(): list<Topic>
AllUser
+AllUser()
+ResearchUser(pseudo: string): User
+AddUser(user: User): int
+AddAdmin(admin: Admin): int
+DeleteUser(utilisateur: User): int
-Lregistre
*
User
+Pseudo: string
+Mail: string
+MotDePasse: string
+User(pseudo: string, modDePasse: string, mail: string)
+ToString(): String
connexion
+Username: string
+Password: string
+Connexion(username: string, passwd: string)
+SeConnecter(): int
Admin
+description
+Admin(pseudo: string, mdp: string, mail: string, description: string)
+ToString(): string

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
+ListeLieux
Region
+Description: string
+Nom: string
+CheminImage: string
+Region(description: string, nom: string, chemin: string)
+ToString(): string
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
+DateArticle: string
+Article(titre: string, texte: string)
+Modifier(newTexte: string)
+administrateur

Vidéo

Voici le lien de la vidéo de notre projet :

Lien = VidéoProjet