Dans le cadre du cours 8INF138, le dernier travail pratique consistait à créer une application implémentant différentes fonctionnalités de sécurité telle que l’authentification et la gestion des accès de différents utilisateurs. C’est à l’aide du langage C# ainsi que Windows Forms que j’ai pu créer une version locale d’un tel système sécuritaire et fonctionnel. L’IDE utilisé était Visual Studio 2022. Regardons d’abord l’architecture du système.

Le système est divisé en deux parties distincts, les classes aidantes et les classes qui héritent de « Form » (pour nous donner du visuel). Commençons par les classes héritant de « Form » ; il y en existe une pour chaque type d’accès (« Admin.cs », « Buisness.cs », « Residential.cs » et « NoAccess,cs »), qui affiche de l’information relatif à chaque type d’accès. Il existe cependant aussi d’autres classes « Form », telle que :

* « Login.cs » : utilisé lorsqu’on veut se connecter.
* « AccessModifier.cs » : utilisé lorsque l’administrateur veux modifier les accès des autre utilisateurs.
* « ChangePassword.cs » utilisé lorsqu’un utilisateur souhaite modifier son mot de passe.

Si on va du côté des classes que j’appelle aidante, nous avons :

* « Logger.cs » : utilisé pour journaliser des informations dans le fichier ‘logs.log’.
* « Authentifier.cs » : utilisé pour authentifier, modifier ou créer un nouvel utilisateur dans le fichier ‘passwords.dat’.
* « AccessManager.cs » : utilisé pour gérer les accès de chaque utilisateur et stocké dans le fichier ‘access.dat’.
* « Settings.cs » : utilisé pour gérer les paramètres définis par l’administrateur telle que par exemple la longueur minimale du mot de passe. Ces réglages sont stocké dans le fichier ‘settings.cfg’.
* « User.cs » : utilisé pour stocker de l’information sur un utilisateur donné telle que ses accès ainsi que son nom.
* « Cryptography.cs » : utilisé pour gérer tout ce qui concerne le chiffrage, le déchiffrage ainsi que le hachage de données.

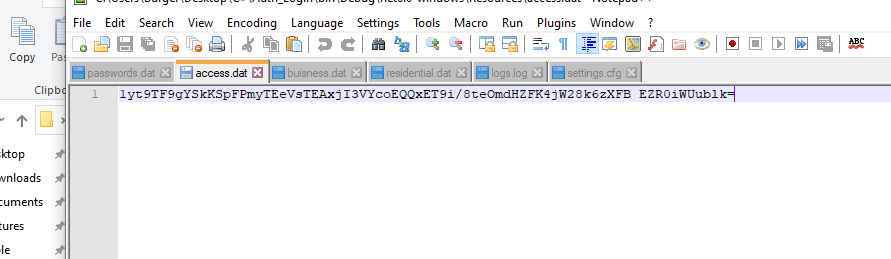
Bref, cela explique bien l’architecture à haut niveau du projet.

Pour ce qui est de la gestion des accès, commençons d’abord par observer comment les accès sont stocké. Les accès sont stockés d’une façon décrite par cet exemple pour un utilisateur: ‘Admin:0,1,2;’ On prend d’abord le nom de l’utilisateur et on met des chiffres pour définir quelle(s) accès l’utilisateur possède. Dans cette application, les accès possibles sont les suivants :

Une image contenant texte

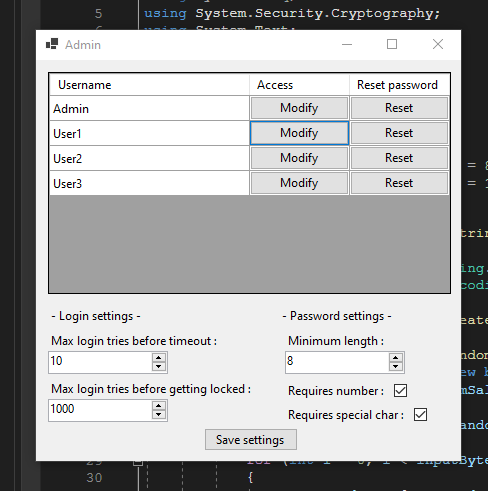
Description générée automatiquement

Lors de la sauvegarde de l’accès des utilisateurs, on colle chaque définition d’accès un derrière l’autre et on encrypte cette ligne (à l’aide de « Cryptography.cs » qui utilise le chiffrage AES) pour obtenir un résultat comme celui-ci.

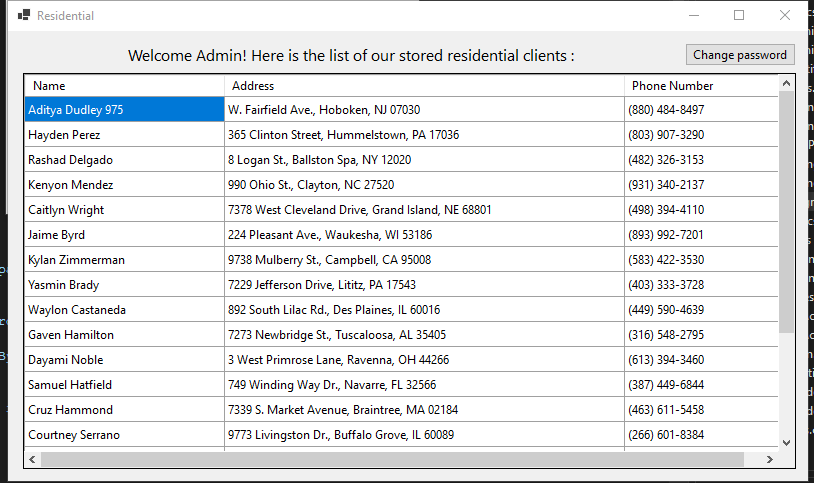


À l’inverse, quand l’application est lancé, une liste des accès de chaque utilisateur est créer à partir du déchiffrage de ce fichier. Cette liste est donc utilisé lorsqu’un utilisateur ce connecte pour savoir quelle « Form »(s) ouvrir (quelle « Form »(s) l’utilisateur qui se connecte aura accès). Voici donc chaque « Form » possible dépendant des accès de l’utilisateur :

* Admin :



* Residential :



* Buisness :

Une image contenant texte

Description générée automatiquement

* No access (quand un nouvel utilisateur est créer) :Une image contenant texte

  Description générée automatiquement

L’administrateur peux aussi modifier les accès de chaque utilisateur dans le « Form » Admin en cliquant sur le bouton « Modify » d’un utilisateur :

Une image contenant table

Description générée automatiquement

Les points forts sont la possibilité d’ajuster les accès à partir de l’administrateur, ainsi que le chiffrage des accès dans le fichier. Le plus grand point faible est le fait que l’application ne soit pas en ligne, car les accès ainsi que le mot de passe pour chiffrer les accès sont stockés localement à côté de l’exécutable de l’application, ce qui ruine la sécurité de cette partie.

Bref, cela résume bien comment les accès sont gérés dans l’application.

Pour ce qui est de l’authentification, les classes « Authentifier.cs » et « Cryptography.cs » viennent jouer un rôle important. En effet, quand on clique sur le bouton login de « Login.cs », les entrées sont prit et envoyé dans la fonction « LoginUser » de « Authentifier.cs ». Cette fonction marche en 3 étapes :

1. On obtient la ligne dans