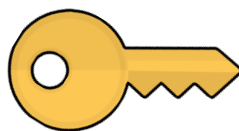

Rapport IHM

Gestionnaire de mots de passe - Benjamin LAMBERT




I - Préface

Mon projet consiste en un générateur et un gestionnaire de mot de passe. Il est composé de différentes parties, comme une page de connexion, une page de création de compte, et une page principale, elle-même composée de trois éléments : le gestionnaire (qui répertorie tous les mots de passe enregistrés), l'enregistreur de mot de passe (permettant d'enregistrer un mot de passe en renseignant le site d'origine, le nom d'utilisateur et le mot de passe) et le générateur de mot de passe.

The screenshot displays the 'Manager (Gandalf • 2022.02.10 • par Pékul)' application window. The interface is divided into two main sections: 'GESTIONNAIRE DE MOT DE PASSE' on the left and 'ENREGISTRER' on the right. The 'GESTIONNAIRE DE MOT DE PASSE' section contains a list of stored passwords, with the first entry showing 'Site d'origine: https://www.google.fr', 'Utilisateur: gandalf', and 'Mot de passe:'. Below this entry is a red 'Supprimer' button. The 'ENREGISTRER' section includes input fields for 'Site d'origine', 'Nom d'utilisateur', and 'Mot de passe', along with a 'Se connecter' button. Below these fields is a 'Force' indicator showing 'Très faible' and buttons for 'Annuler' and 'Enregistrer'. At the bottom right, there is a 'GENERER UN MOT DE PASSE' section with a 'Mot de passe' field showing 'vF%Y{xPoolW%ZSgA', a 'Longueur du mot de passe' slider set to 16, and buttons for 'Copier' and 'Générer le mot de passe'.

II - Les débuts : le générateur

Afin de prendre en main PyQt5, j'ai d'abord réalisé une interface simple servant à générer un mot de passe. Cette interface utilisateur comportait des QLabel, un QFormLayout, des QLineEdit, et un QSlider. Cette interface fut rapide à créer et le programme permettant de générer un mot de passe aléatoire (d'une taille allant de 4 à 24 caractères et comprenant toutes les lettres majuscules, minuscules, tous les chiffres, et toute la ponctuation) n'est pas complexe. Grâce à cette interface on pouvait donc générer un mot de passe, changer la longueur du mot de passe généré, ainsi que copier ce dernier.



The image shows a graphical user interface for a password generator. At the top, there is a title bar with the text "GENERER UN MOT DE PASSE". Below this, there is a label "Mot de passe" followed by a text input field containing the password "vF%Y{xPoolW%ZSgA". Below the password field, there is a label "Longueur du mot de passe" followed by a horizontal slider. The slider has a blue handle and is positioned at the value 16. To the right of the slider, the number "16" is displayed. At the bottom, there are two buttons: a white button with a downward arrow icon and the text "Copier", and a blue button with a gear icon and the text "Générer le mot de passe".

III - Création des différentes fenêtres

Par la suite, il m'a fallu créer une interface graphique afin de permettre à l'utilisateur de se connecter et/ou de créer un compte. J'ai alors fait le choix de créer deux fenêtres supplémentaires, une pour la connexion et une pour la création de compte. Afin de simplifier mes futures créations de fenêtres, j'ai décidé de créer une classe Window, héritant de QMainWindow. Cette classe m'a également permis de créer des attributs afin de stocker les différentes fenêtres de mon application, et permettre à mon code de pouvoir changer les différentes fenêtres pour s'adapter au contexte.

La fenêtre de connexion est composée de deux QLineEdit, un pour le nom d'utilisateur et un pour le mot de passe, ainsi que de deux boutons, un pour valider la connexion et un pour se créer un compte. La fenêtre de création de compte est identique mais incorpore un dernier QLineEdit pour confirmer le mot de passe.

IV - Le début des difficultés

C'est de loin la fenêtre de gestion qui a été le plus complexe à créer. En effet, il a fallu créer une QScrollArea, où je devais rajouter plusieurs "widgets" (QFormLayout composé de plusieurs QLineEdit en read only, de plusieurs boutons, et d'un QRadioButton) représentant chacun un mot de passe enregistré, que l'on appellera par la suite 'item'. Cependant, si l'utilisateur décidait de supprimer ou d'ajouter un mot de passe, il fallait mettre à jour la liste des mots de passe afficher dans le QScrollArea. Pour cela, j'ai dû créer une méthode permettant de réinitialiser le QScrollArea. Au début, je pensais qu'il fallait simplement fermer chaque widget contenu dans mon layout. Cependant, je me suis vite rendu compte qu'en procédant de cette façon, mon layout restait identique, et dans le pire des cas j'avais une ou plusieurs erreurs. C'est alors que je me suis rendu compte que dans mon 'item', j'avais également des layouts (notamment des QHBoxLayout), qu'il fallait également supprimés. Ma méthode à donc dû se complexifier. En effet, pour chaque enfant de mon layout je devais soit fermer le widget si l'enfant était un widget, soit réinitialiser l'enfant si l'enfant était lui aussi un layout. C'est à ce moment là que j'ai compris qu'il y avait d'autres layouts dans mon layout initial, et qu'il fallait faire appel à de la récursivité.

```
55     def clear_layout(self, layout):
56         while layout.count():
57             child = layout.takeAt(0)
58
59             if child.widget() is not None:
60                 child.widget().close()
61                 layout.removeWidget(child.widget())
62             elif child.layout() is not None:
63                 self.clear_layout(child.layout())
```

V - Une interface fonctionnelle

Maintenant que toutes les interfaces de mon application ont été désignées, il m'a fallu implémenter l'intégralité des fonctionnalités.

J'ai ajouté chaque fonctionnalité dans le même ordre dans lequel l'utilisateur va les utiliser. La création de compte a donc été la première étape. J'ai utilisé 'sqlite3' pour la gestion des comptes utilisateurs. Un compte utilisateur regroupe le nom ainsi que le mot de passe de l'utilisateur. Le mot de passe de l'utilisateur est crypté grâce à la librairie hlib et la méthode sha256. Cela permet une encryption sans réel chemin de retour, mais cela ne pose aucun problème car, pour la connexion, j'ai juste à vérifier que les deux hash sont identiques, ce qui signifierait que les mots de passe non-cryptés étaient identiques.

J'ai ensuite créé la fenêtre principale, celle avec le générateur, l'enregistreur et le gestionnaire. Mon programme de génération est resté identique au premier que j'avais fait. Le programme d'enregistrement du mot de passe a lui aussi un fonctionnement très simple. Il prend les trois données d'entrées : le site d'origine, le nom d'utilisateur, et le mot de passe, puis enregistre dans un fichier JSON, sous le nom de l'utilisateur actuel le mot de passe.

```
1  {  
2    "admin": [  
3      {  
4        "origin": "https://www.google.fr",  
5        "username": "gandalf",  
6        "password": "r{n\u0080z\u0083Y3J"  
7      }  
8    ]  
9  }
```

Le mot de passe est alors crypté grâce à une méthode que j'ai moi même créer, se basant sur l'encodage CÉSAR. Cette méthode décale chaque caractère du mot de passe d'un nombre, choisis aléatoirement entre 10 et 26 et unique pour chaque caractère, transformant ainsi le caractère du mot de passe. 'bonjour' peut par exemple devenir ':/O,s!['.

La méthode de décryptage reprend le même principe, mais le fait dans l'autre sens.

La justesse de cette méthode d'encryptage est possible grâce à une graine de génération commune entre les deux 'aléatoires'.

```

7      def encrypt(self, text):
8          random = Random(self.key)
9          encrypted_text = ""
10
11         for char in text:
12             value = random.randint(10, 26)
13             encrypted_text += chr(ord(char) + value) if ord(char) + value < 256 else char
14
15         return encrypted_text
16
17     def decrypt(self, text):
18         random = Random(self.key)
19         decrypted_text = ""
20
21         for char in text:
22             value = random.randint(10, 26)
23             decrypted_text += chr(ord(char) + value) if ord(char) + 2 * value >= 256 else chr(ord(char) - value)
24
25         return decrypted_text

```

Pour la partie “Gestionnaire”, les ‘items’ étant créés il m’a simplement fallu créer les méthodes appelées par les différents boutons de chaque item, comme la fonction ‘delete_password’ qui supprimait un mot de passe de la base de données. De même, j’ai fait le choix de pouvoir afficher le message en clair, pour cela j’ai utilisé un QRadioButton, qui changeait le mode du QLineEdit pour alterner entre QLineEdit.Normal et QLineEdit.Password.

VI - Les fonctionnalités bonus

Une fois les fonctionnalités de base implémentées, j'ai cherché d'autres fonctionnalités bonus que je pourrais faire. Je me suis donc dit que le fait de pouvoir copier une des informations d'authentification (mot de passe ou nom d'utilisateur) pourrait être pratique. J'ai donc utilisé le module 'pyperclip' afin de facilement pouvoir ajouter un texte au presse papier. Cela allait de paire avec un bouton "Copier" à côté de l'information correspondante.

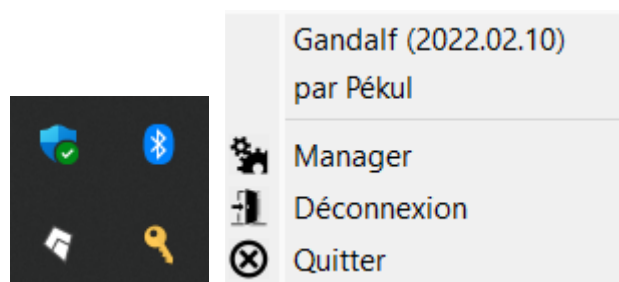
J'ai également décidé d'ajouter un QMenuBar, afin de simplifier la navigation et de permettre à l'utilisateur de réinitialiser son gestionnaire, mais aussi de se déconnecter sans avoir besoin de redémarrer l'application.



Je me suis ensuite dit qu'il serait utile de toujours avoir accès à l'application, et ce, rapidement. Je me suis donc renseigné pour pouvoir ajouter une icône dans la barre des tâches afin d'avoir un accès rapide et simple à tout moment. Mes recherches m'ont alors mené vers les QSystemTray. Les QSystemTray sont les petites icônes que l'on peut voir dans nos barres des tâches. Un clic gauche (ou droit) sur ces icônes permettent de faire des actions précises sans avoir besoin d'avoir l'application ouverte.

Il s'est avéré que le QSystemTray était principalement, et dans mon cas uniquement, composé d'un QMenuBar. Une très bonne découverte puisque j'utilisais déjà un QMenuBar, j'ai donc réutilisé le même menu que pour ma fenêtre principale.

Il ne me restait plus qu'à afficher le QSystemTray quand l'utilisateur fermait la fenêtre. J'ai donc réécrit la fonction close, ainsi que la fonction hide de ma fenêtre afin de permettre à mon code de ne pas fermer la fenêtre mais simplement la cacher et d'afficher le QSystemTray dans la barre des tâches. De même, lors de l'appel de la méthode show, il fallait cacher le QSystemTray. Je l'ai donc également réécrit pour me permettre de faire cela.



J'ai également ajouté un installateur automatique des modules python nécessaires au bon fonctionnement de mon programme.

VII - L'optimisation et la refactorisation

Dans l'état actuel, mon application pouvait être qualifiée de "séquentielle" : chaque page de mon application était une `MainWindow`, et j'affichais chacune de ces `MainWindow` les unes après les autres. Cela avait pour effet de faire disparaître puis réapparaître, quelques millisecondes après, l'application. Cependant, cela pouvait faire croire que différentes applications se lançaient, ce qui pourrait perdre l'utilisateur. J'ai donc décidé de "réunifier" toutes les pages de mon application dans une seule, et même, fenêtre.

Pour cela il a fallu que mes `MainWindow` héritent, non plus de `QMainWindow`, mais de `QWidget`. Une autre classe, appelée `Gandalf` (nom de l'application), héritait elle de `QMainWindow`. Ainsi, je pouvais mettre dans le 'centralWidget' de mon objet 'Gandalf' la `MainWindow` que je voulais.

Je me suis rendu compte très peu de temps après avoir fait cette modification, que lorsque je passais de la fenêtre de connexion à la fenêtre de création de compte, ou tout autre passage entre deux fenêtres, que le chemin retour m'était impossible. En effet, lorsque je revenais en arrière, mon interface était vide et la console m'indiquait que ma `MainWindow` avait été "détruite".

Il a donc fallu trouver une alternative. C'est alors que j'ai vu, notamment grâce au cours, qu'il existait des `QStackedWidget`. Ainsi, ma classe `Gandalf`, qui héritait jusqu'à présent de `QMainWindow`, héritait dorénavant de `QStackedWidget`. Avec les modifications nécessaires, mon application était de nouveau fonctionnelle, et cette fois-ci, tout fonctionnait comme espéré.

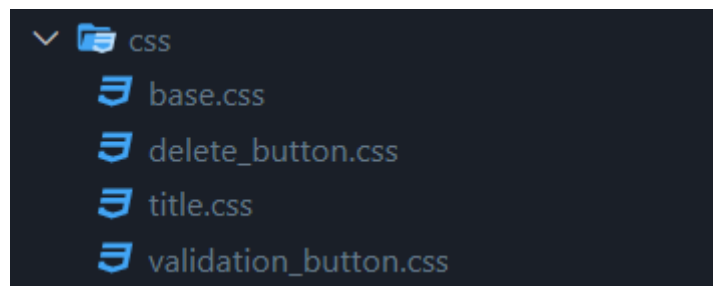
De plus, pour améliorer la sécurité et le cryptage des mots de passe utilisateur, j'ai fait en sorte que la clé d'encryption soit générée automatiquement grâce au mot de passe de l'utilisateur. Elle est donc propre à chaque utilisateur, et non commune à toute l'application, ce qui rend l'application plus sûre pour le stockage de mot de passe, le mot de passe "maître" (celui utilisé par l'utilisateur pour se connecter à l'application) est quasi introuvable.

VIII - L'esthétisme de mon application

Arrivé ici, la très grande majorité de l'interface était finie, il fallait donc la rendre agréable à regarder. Je me suis alors mis à créer le fichier CSS allant avec l'application. J'ai d'abord créé un unique fichier, dans lequel j'ai mis tout le style sheet appliqué à chacune de mes fenêtres. Ce fichier comportait des références aux QLineEdit, aux QLabel, aux QPushButton, et aux QRadioButton.

J'ai ensuite voulu mettre en place des boutons de différentes couleurs, pour mieux guider l'utilisateur. Par exemple, je voulais que les boutons importants soient en bleu, et les autres en blancs, et celui pour supprimer un mot de passe en rouge. Je me suis alors renseigné pour savoir s'il existait un système de classification, comme en HTML. Je n'ai malheureusement trouvé aucun système équivalent dans PyQt5, d'autant plus que le module est basé sur du CSS 2 et non du CSS 3. Il manquait donc des fondamentaux comme la propriété 'transition' ou encore 'cursor'.

Ainsi, ne trouvant aucune alternative viable au système de classification, j'ai décidé de créer d'autres fichiers CSS : un pour les boutons importants (bleu), un pour les boutons de suppression (rouge), les boutons blancs étant de base, le fichier créé initialement suffisait. Chacun des fichiers alors créés était mis comme style sheet des boutons en question. Le bouton gardait alors les propriétés du premier fichier (appliqué à la fenêtre), et remplaçait uniquement certaines propriétés.



De même, je voulais faire en sorte que les titres des différentes sections de mon application soit remarquable, j'ai donc utilisé la même technique.

Par la suite, j'ai également décidé d'ajouter des images à certains de mes boutons, et à mes QMenu, afin de faciliter la compréhension.

IX - Liens utiles

Pendant toute la durée du projet, j'ai utilisé Github afin de sauvegarder ma progression.
Vous pouvez consulter le [repository ici](#).

De plus, toutes les images et icône de l'application sont extraite du site : [Game-icons.net](#)