TP PYSIDE

12 décembre 2023

Dorian Humeau
Mathis Caisson
Nicolas Caroff
Martin Goubet

Résumé

Pour ce TP vous pouvez vous appuyer sur les slides du cours afin de comprendre les différents fonctionnement de PySide et des outils utiles pour ce TP : `https://slides.com/red4game/code/fullscreen`

Table des matières

1 INSTALLATION

1.1 Sous Ubuntu

1.1.1 Installer l'environnement virtuel python

Pour éviter tout conflit par la suite avec les versions de Pyside vérifier la version de python soit bien supérieur ou égale à 3.7.

```
$ python3 -V
```

Pour installer Pyside6, il faut se placer dans le projet et créer un environnement virtuel python.

```
$ python3 -m venv env
```

Activer l'environnement.

```
$ source env/bin/activate
```

1.1.2 Installer PySide

Une fois dans l'environnement, installer Pyside.

```
$ pip install pyside6
```

Il est possible de sortir de l'environnement virtuel.

```
$ deactivate
```

Et de le relancer en le réactivant avec la commande.

```
$ source env/bin/activate
```

Ensuite créer 2 fichiers : `Ui_MainWindow.py` qui va contenir les éléments à afficher dans la fenêtre principale et `main.py` qui va définir les comportements des éléments dans la fenêtre.

1.1.3 Lancer QtDesigner

Dans le venv, lancer QtDesigner avec la commande suivante.

```
$ pyside6-designer
```

QtdDesigner va permettre de définir des fenêtres personnalisées et de générer le code qui sera copier dans `Ui_MainWindow.py`.

1.2 Sous Windows

Exécutez les mêmes commandes que sous linux en remplaçant la commande d'activation de l'environnement par celle-ci :

```
$ env\Scripts\activate.bat
```

Vous devez impérativement réaliser ces commandes dans un cmd et non pas un PowerShell. Enfin dans le cas où le module PySide6 ne serait pas trouvé, vous pouvez exécuter cette commande :

```
$ set PYTHONPATH=C:\path\to\your\venv\Lib\site-packages;%PYTHONPATH%
```

2 ETAPE 0 : FENÊTRE DE BASE

2.1 Bouton simple

Dans un premier temps nous allons réaliser une fenêtre avec un bouton. Lorsque l'on clique dessus il affiche quelque chose dans la console.

Copier le code suivant dans le `main.py`.

```
import sys

from PySide6.QtWidgets import QApplication, QMainWindow, QLabel
from Ui_MainWindow import Ui_MainWindow
from PySide6.QtCore import Slot, Qt

class MainWindow(QMainWindow, Ui_MainWindow):

    def __init__(self):
        super().__init__()
        self.setupUi(self)

App = QApplication(sys.argv)
window = MainWindow()
window.show()
App.exec_()
```

Ce code va permettre d'afficher la fenêtre principale. Il va maintenant falloir créer la fenêtre.

Ouvrir QtDesigner et créer un widget `Main Window`. Dans cette fenêtre placer un layout pour permettre aux éléments de bien se placer par la suite. Nous pouvons utiliser `Vertical Layout`. Pour faire en sorte que le layout prenne la taille de la fenêtre, faire un clic-droit sur le widget (qui contient le layout) et dans "Lay out" sélectionner "Lay out in a grid". Ajouter un `Push Button` dans le layout. Ensuite exporter le code python de cette fenêtre dans *Form->View Python code...* Si vous avec une erreur du type :

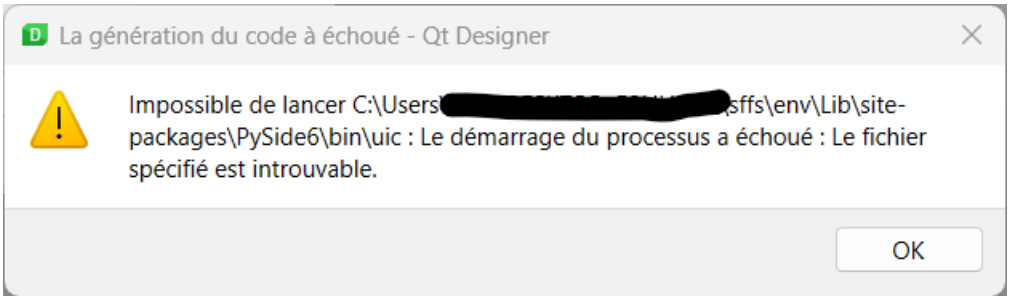

FIGURE 1 – Problème visualisation code Python

Allez dans le dossier PySide6 indiqué puis créez un dossier bin dans lequel vous copier/-coller l'exécutable `uic.exe` déjà présent dans le dossier */Pyside*. Vous pouvez maintenant récupérer le code Python.

Copier le code générer par Qt designer dans le fichier `Ui_MainWindow.py` préalablement créé. Dans le venv nous pouvons lancer l'application avec la commande suivante.

```
$ python3 main.py
```

FIGURE 2 – Fenêtre de base

L'application se lance mais le bouton ne fait rien, puisque qu'aucune fonction n'est appelé sur un clic.

Créer une fonction `def buttonClicked(self):` qui affiche `"Button clicked"` dans la console lorsqu'elle est appelée. Pour effectuer une action lorsqu'un évènement est détecté il faut utiliser le wrapper `@Slot` avant la fonction

Ensuite il faut appelé cette fonction lorsque le bouton est appelé. Il faut donc définir dans l'initialisation que l'évènement `click` sur le bouton va déclencher la fonction précédemment définie. Pour se faire utiliser la structure :

```
self.<component>.<event>.connect(<function>)
```

Ici le composant correspond à `pushButton`, l'évènement à `clicked` et la fonction à `self.buttonClicked` Il est à noté qu'il est possible de changer le nom des composants dans `ObjectName` dans QtDesigner pour mieux s'y retrouver par la suite.

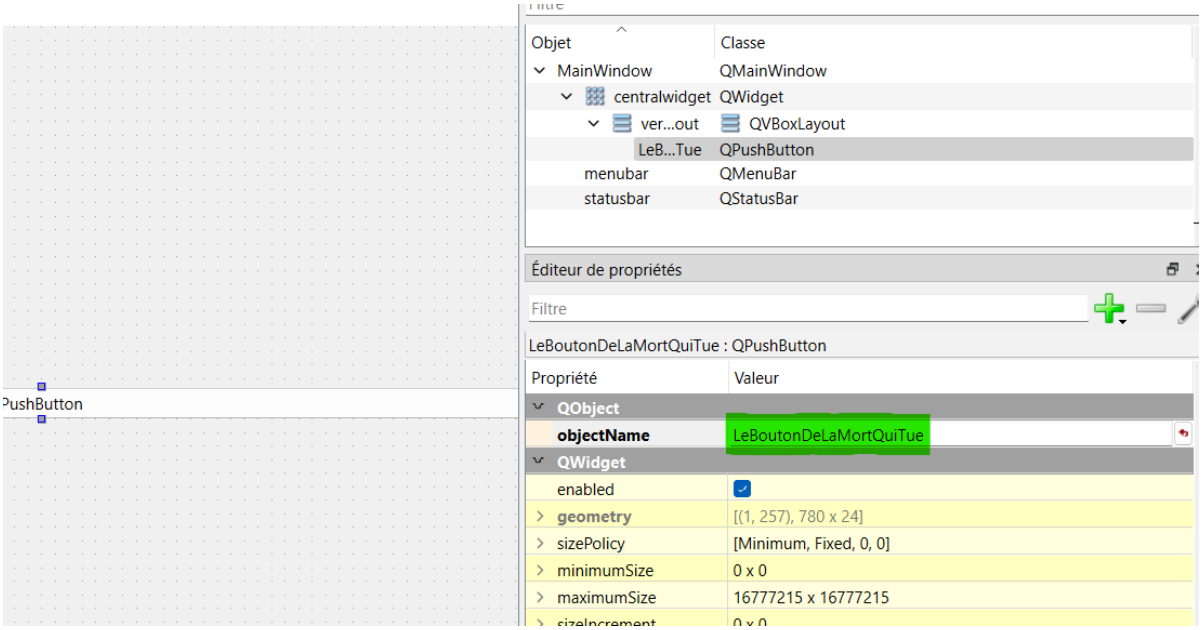

FIGURE 3 – Renommage d'un QtObject dans QtDesigner

2.2 Avec un *LineEdit*

Récupérons la fenêtre précédemment défini et ajoutons s'y une entrée texte `LineEdit`. Modifier le nom du `LineEdit` en `message`. Le but va d'être d'afficher le text contenue dans LineEdit lorsque qu'on appuie sur le bouton.

Importer le code généré par QtDesigner dans le fichier correspondant.

Modifier la fonction `Button clicked` pour que le message définie dans l'entrée de texte nommée "message" s'affiche dans la console. Pour récupérer le contenue de l'entrée de texte, utiliser la fonction `.text()`.

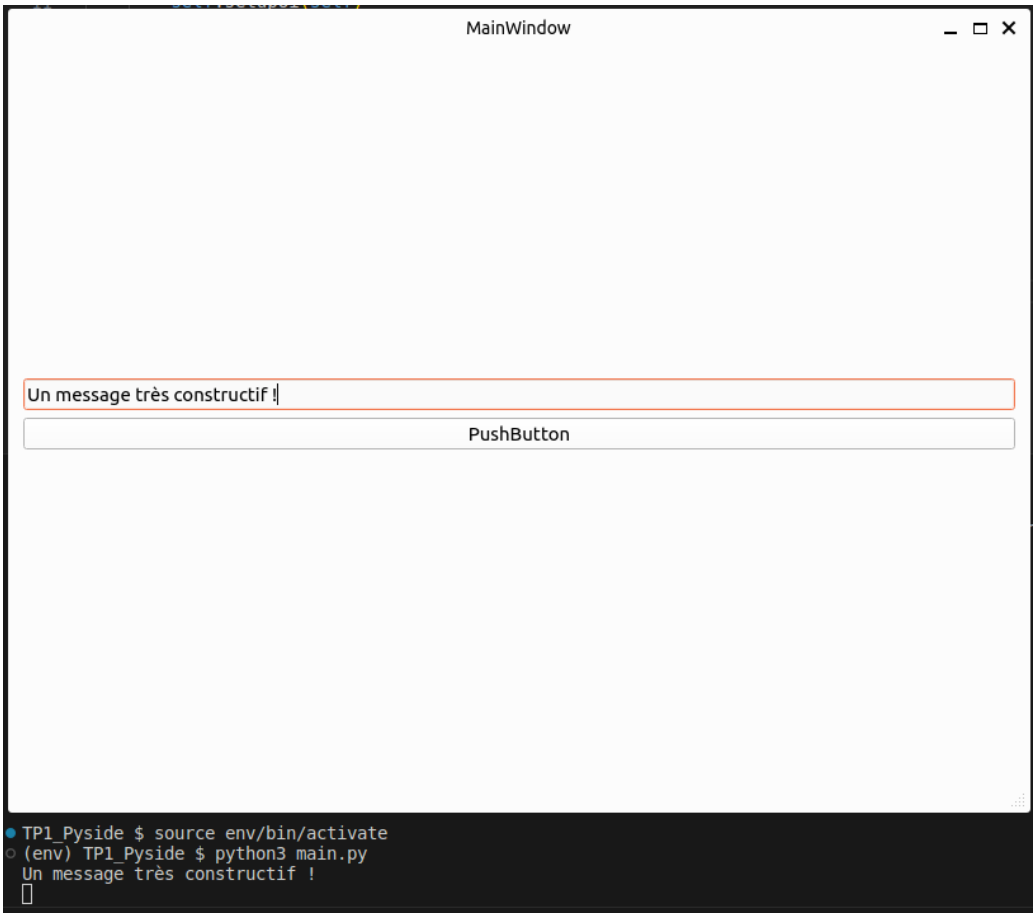

FIGURE 4 – Fenêtre de base avec un LineEdit

Faire en sorte que le texte de l'input box disparaisse une fois que le message est envoyé. Regarder la documentation Doc/Pyside6/QLineEdit.html pour trouver la fonction à utiliser. Chercher dans la section "Functions" pour connaître les actions réalisable sur l'object.

3 ETAPE 1 : DEUX FENÊTRES

3.1 Design de la première fenêtre

Nous allons mettre tous les widgets nécessaires à la création du client IRC dans la première fenêtre (nous allons réutiliser la fenêtre de l'étape 0 en enlevant les widgets que l'on a mis). Pour se connecter au serveur de chat IRC, nous avons besoin de 3 paramètres : l'adresse du serveur, le port, et un pseudo. Il faudra également un bouton de connexion. Pour demander ces 3 informations, nous allons utiliser un layout formulaire (`FormLayout`). Pour l'adresse du serveur et le port, vous pouvez mettre des valeurs par défaut pour ne pas avoir besoin de les rentrer à chaque fois. **Pour le TP, nous utiliserons le serveur dorian.tk sur le port 5000**. [1]

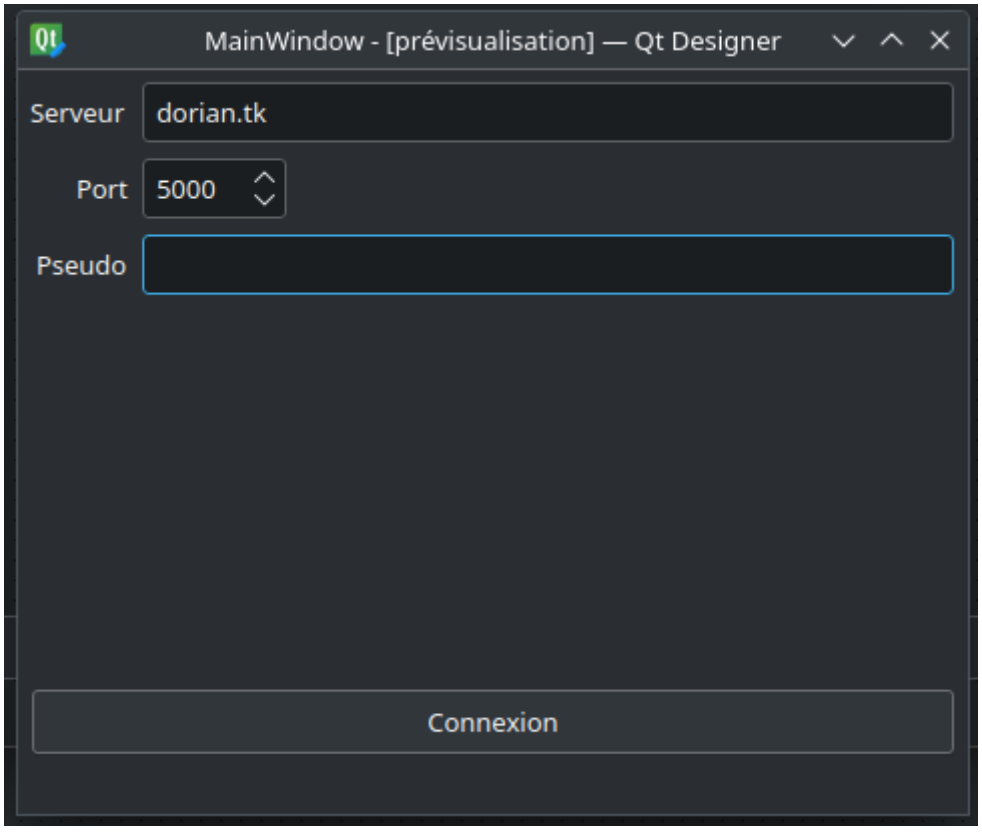

FIGURE 5 – Un exemple de la première fenêtre

Remarque : pour le port vous pouvez utiliser un `SpinBox`, avec des valeurs allant de 1 à 65536. Mais il est aussi possible d'utiliser un simple `LineEdit`.

1. Normalement IRC utilise le port 6667 par défaut, mais puisqu'il est bloqué par la DSI j'ai utilisé le port 5000 qui ne l'est pas, pour que le TP marche sur l'insaWifi

Après avoir mis le nouveau code de la première fenêtre dans "ui_mainWindow.py" et retiré les `connect` dans le "main.py", vous pouvez lancer le main, et la première fenêtre devrait s'afficher.

3.2 Création de la 2ème fenêtre

Nous allons désormais créer une deuxième fenêtre qui récupérera des données de la première, cela servira pour la suite du TP. Afin de pouvoir la différencier dans le code, il est important de changer son nom. Cela se fait en changeant la propriété `objectName` dans l'objet de classe `QMainWindow`. Nous allons appeler notre fenêtre **ChatWindow** (notez que ce nom n'est utilisé qu'en interne. Le nom affiché est celui de la propriété `windowTitle`. Vous pouvez le changer comme vous le voulez)

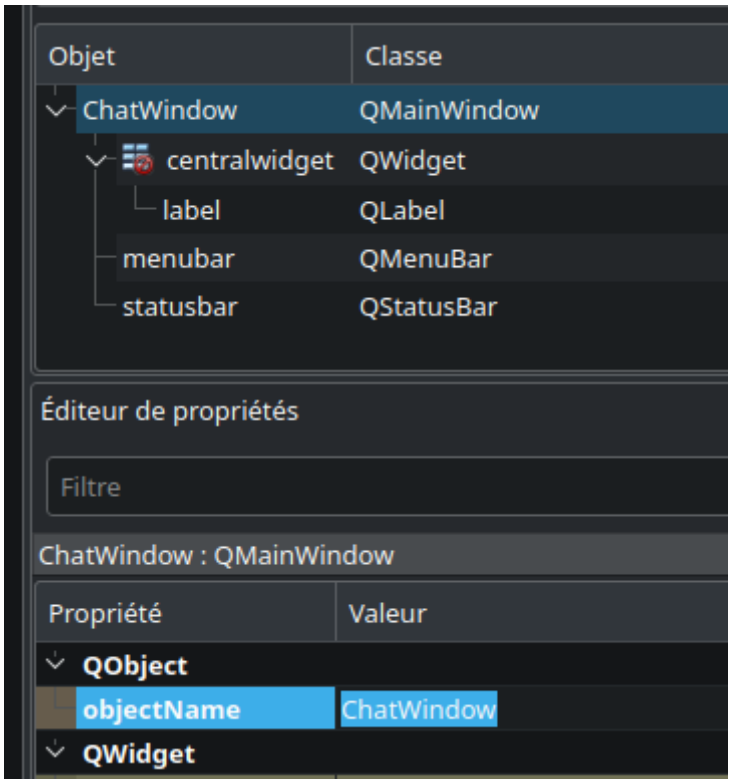

FIGURE 6 – Nom d'une fenêtre dans QT Designer

Dans cette fenêtre, nous allons simplement mettre un `Label` pour l'instant. Une fois la fenêtre créée, copiez son code et mettez-le dans un **nouveau fichier py** que l'on nommera "ui_chatWindow.py".

3.3 Liaison des 2 fenêtres

Nous allons faire en sorte qu'une fois que l'on clique sur le bouton de connexion, la première fenêtre disparaisse, et la deuxième apparaisse **avec les informations rentrées dans la première**.

Tout d'abord, il faut créer dans le main une classe pour la deuxième fenêtre. Cette classe fonctionne de la même manière que la classe `MainWindow`, mais dérivera de la classe `Ui_ChatWindow` au lieu de `Ui_MainWindow` :

```
from Ui_ChatWindow import Ui_ChatWindow

#(...)

class ChatWindow(QMainWindow, Ui_ChatWindow):

    def __init__(self, var1, var2, ....):
        super().__init__()
        self.setupUi(self)
        self.var1 = var1
        self.var2 = var2
        #(etc...)
```

En remplaçant `var1` et `var2` par les noms de variables voulus. Il faudra récupérer les 3 informations de la première fenêtre, donc il faudra 3 variables bien nommées pour stocker ces valeurs.

Ensuite, pour vérifier si l'on a bien réussi à récupérer ces valeurs, rajoutez une ligne dans le constructeur qui viendra modifier le label de la 2ème fenêtre pour y afficher la valeur d'une des 3 variables.

Attention : on serait tenté de faire `label.text="..."` mais c'est `label.setText("...")` qu'il faut utiliser

Enfin, dans la classe `MainWindow`, nous pouvons rajouter un Slot qui construira la 2ème fenêtre avec les bons paramètres, masquera la première, et affichera la deuxième :

```
@Slot()
def showChat(self):
    self.chatWindow = ChatWindow(...)
    self.hide()
    self.chatWindow.show()
```

N'oubliez pas de connecter le bouton de connexion au Slot !

Une fois fait, vous devriez pouvoir lancer le main, cliquer sur Connexion et avoir la deuxième fenêtre qui s'affiche avec les informations de la première.

4 ETAPE 2 : WIDGET DE CHAT

Afin d'avoir un composant réutilisable, nous allons créer notre propre Widget que l'on intégrera dans la 2ème fenêtre.

4.1 Création du widget

Dans QT Designer, créez un nouvel élément, qui cette fois-ci sera un `Widget` et non un `Main Window`. Pensez à modifier le nom du widget, nous allons l'appeler `ChatWidget`.

Pour ce widget, nous allons avoir besoin d'un `TextEdit` en lecture seule pour contenir les messages, d'un `LineEdit` pour écrire le message à envoyer, et d'un bouton pour envoyer le message.

Une fois le widget créé, exportez le code python dans "ui_chatWidget.py"

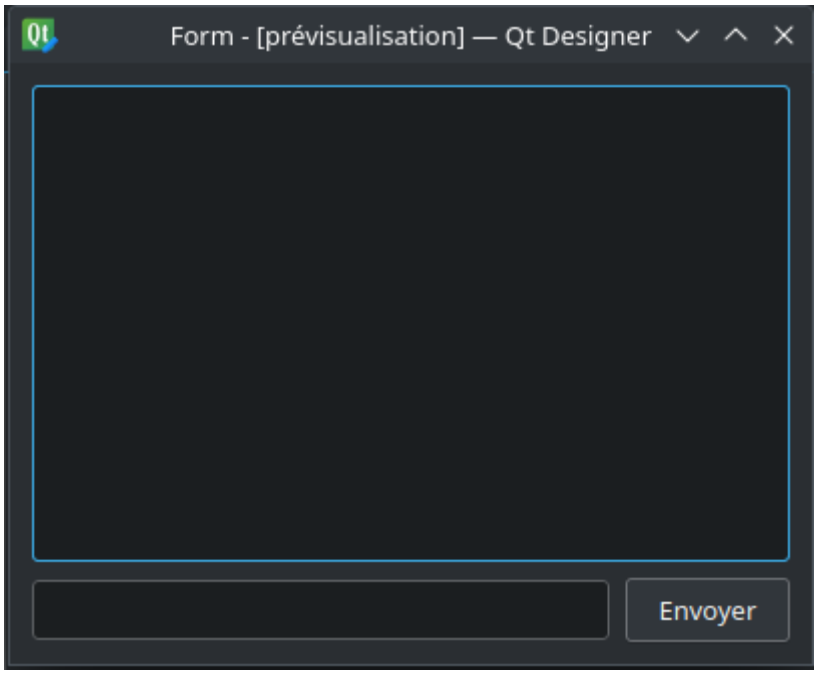

FIGURE 7 – Exemple de widget de chat

4.2 Intégration du widget

Tout comme pour la deuxième fenêtre, il faut créer une nouvelle classe `ChatWidget` dans le main, qui dérivera de `Ui_ChatWidget` (pensez à importez le fichier), et cette fois-ci de `QtWidgets.QWidget` au lieu de `QtWidgets.MainWindow`.

Pour les Widgets, on aura aussi besoin de récupérer le parent (qui sera l'objet qui contient notre widget), il faudra donc l'indiquer dans le constructeur :

```
class ChatWidget(QtWidgets.QWidget, Ui_ChatWidget):
    def __init__(self, parent=None):
        super().__init__(parent=parent)
        self.setupUi(self)
```

Il faut maintenant mettre ce widget dans la 2ème fenêtre. Retirez le `Label` et mettez un élément de type `Widget` qui remplit toute la fenêtre. Ce widget n'est qu'une coquille vide pour l'instant, il faudra le remplacer par le notre. Il est possible de le faire avec du code (nous aurons l'occasion de le faire plus tard), ou de simplement donner à QT Designer le nom de la classe à utiliser pour remplir ce widget.

Pour ce faire, faites un clic-droit sur le widget dans l'inspecteur d'objet, et cliquez sur "Promouvoir en..."

Dans la fenêtre qui s'ouvre, indiquez le nom de la classe à utiliser (ChatWidget) et

le nom du fichier qui contient cette classe (main) [2]. Cliquez ensuite sur "Ajouter" puis "Promouvoir"

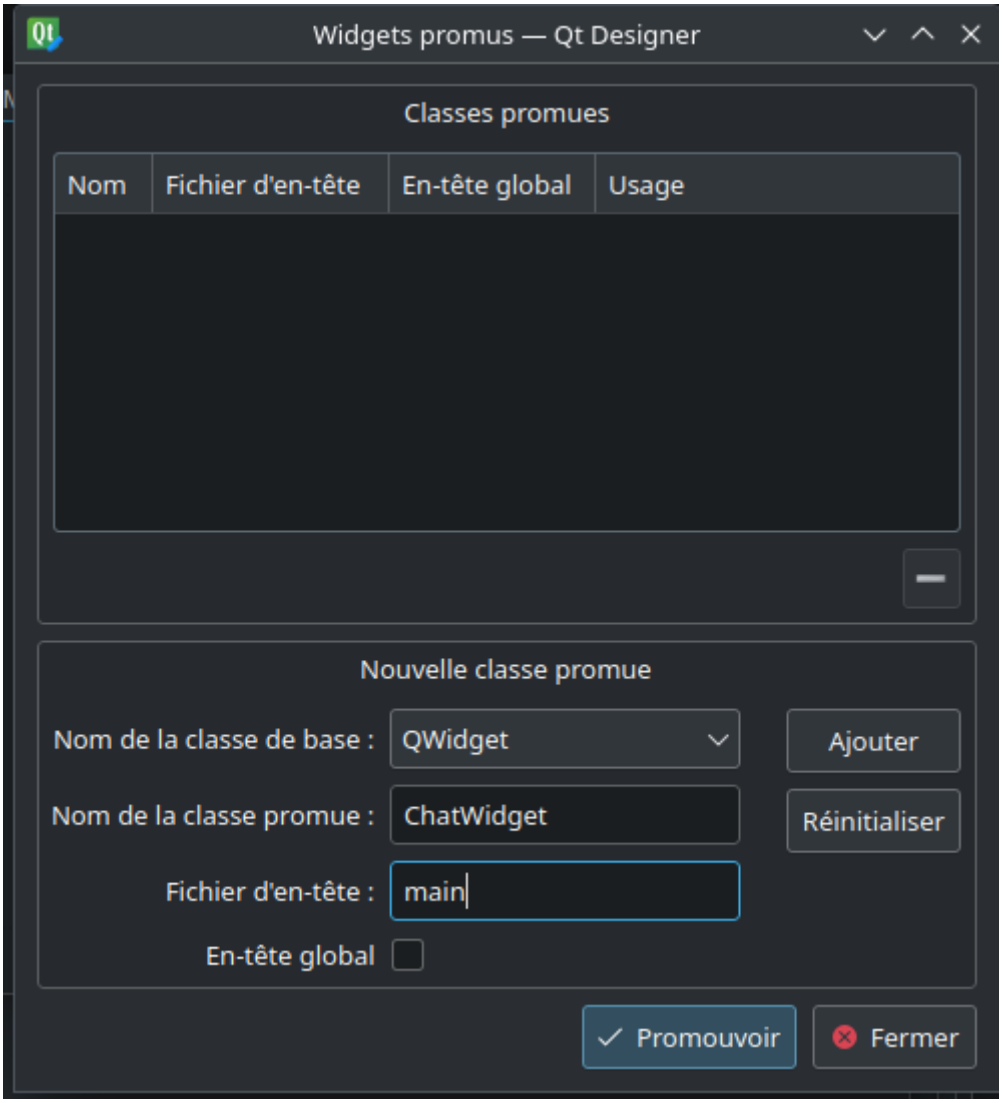

FIGURE 8 – Promotion du widget

Exportez le code python dans "ui_chatWindow.py" et lancez le main. Une fois que vous cliquez sur "Connexion", la deuxième fenêtre avec notre widget de chat doit apparaître. (il faudra retirer le code qui modifie le label puisqu'il y est plus)

2. Par défaut QT Designer met un exemple avec un .h à la fin vu qu'il est prévu pour du C++. Mais quand on converti en Python, le .h est retiré s'il y est, donc "main" et "main.h" font la même chose pour notre cas

5 ETAPE 3 : INTÉGRATION IRC BASIQUE

5.1 Intégration de la librairie

Il est l'heure d'intégrer la communication IRC dans le projet ! Pour ce faire, ajoutez le fichier "QIRC.py" qui est fourni avec le sujet dans le dossier du TP.

Pour l'utiliser, il faut instancier une classe `QIRCClient` avec les bons paramètres (expliqués juste après). Cependant, cette classe contient une boucle qui attend les messages (tout comme QT contient une boucle qui attend les évènements claviers et souris). Il faut donc "exécuter" cette classe dans un thread séparé. Pour faciliter cette tâche, la classe `QIRCClient` a justement été créée avec cet usage en tête, avec des signaux QT pour pouvoir l'intégrer très facilement dans notre projet.

Pour intégrer le thread, il faut ajouter le code suivant dans la classe `ChatWindow` :

```
from QIRC import QIRCClient
from PySide6.QtCore import QThread

#...

class ChatWindow(...):
    thread = QThread()
    IRCworker = QIRCClient(nickname=...=,server=...,port=...)

    def __init__(...):
        #...
        self.thread.started.connect(self.IRCworker.run)
        self.IRCworker.moveToThread(self.thread)

        self.IRCworker.basicEvent.connect(...)
        # (connexions des slots)

        self.thread.start()
```

Dans ce code, il faudra connecter `self.IRCworker.basicEvent` à un Slot qui prend en paramètre un string. Ce string contiendra le message reçu ou d'autres évènements déjà formatés. Créez ce slot de manière à ce qu'il ajoute le string reçu dans le `TextEdit` et lancez le main. S'il vous manque la librairie irc, installez-la avec `pip`.

Lorsque la fenêtre de chat s'ouvre, vous devriez avoir des messages qui s'affichent rien qu'au moment de vous connecter. Si ce n'est pas le cas, n'hésitez pas à demander de l'aide.

IMPORTANT : De base, IRC n'est pas chiffré, et le serveur utilisé n'en intègre pas. N'envoyez rien de personnel !

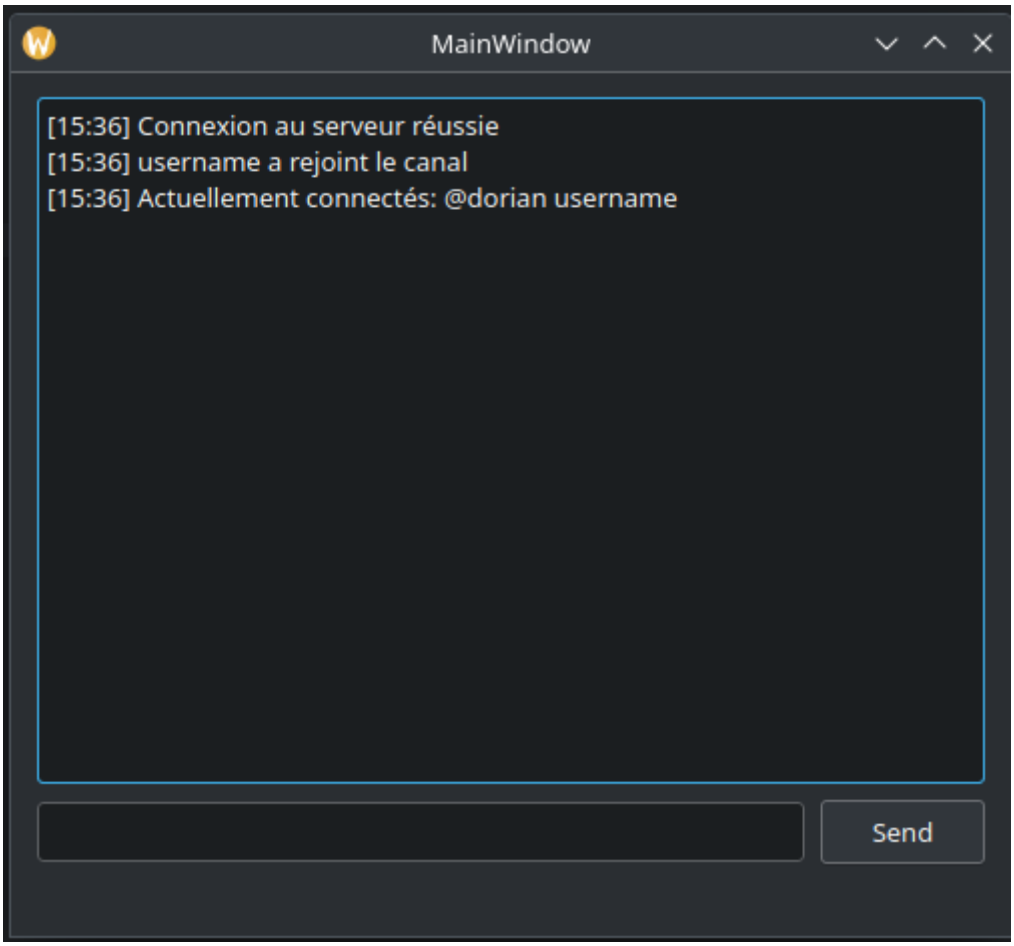

FIGURE 9 – L'interface de chat fonctionnelle

5.2 Envoi de messages

Grâce à la classe fournie, vous pouvez facilement envoyer un message sur le salon de chat. Créez un slot qui sera connecté au bouton d'envoi et au `LineEdit`, et appelez simplement la fonction suivante :

```
self.IRCworker.sendMessage(...)
```

NOTE : Quand vous envoyez un message, cela ne déclenche pas de signaux, donc le message ne sera pas affiché pour vous même si les autres le verront ! Dans votre slot, il faudra donc aussi faire en sorte que le message envoyé soit rajouté au `TextEdit`

6 ETAPE 4 : UN CLIENT PLUS COMPLET

6.1 Interface plus complète

Maintenant que nous avons un client minimal fonctionnel, nous pouvons ajouter des fonctionnalités. Nous allons gérer le fait de pouvoir envoyer des messages privés Au lieu de mettre juste le widget de chat, on met des tabs (un pour le salon principal, et les autres ce sera pour les MP). C'est là qu'on profitera du fait qu'on a fait un widget réutilisable. Pour créer un MP, on mettra à droite la liste des gens connectés (le script permet de le faire facilement) et on s'en servira pour créer un nouveau tab à la demande pour écrire des MP à quelqu'un.

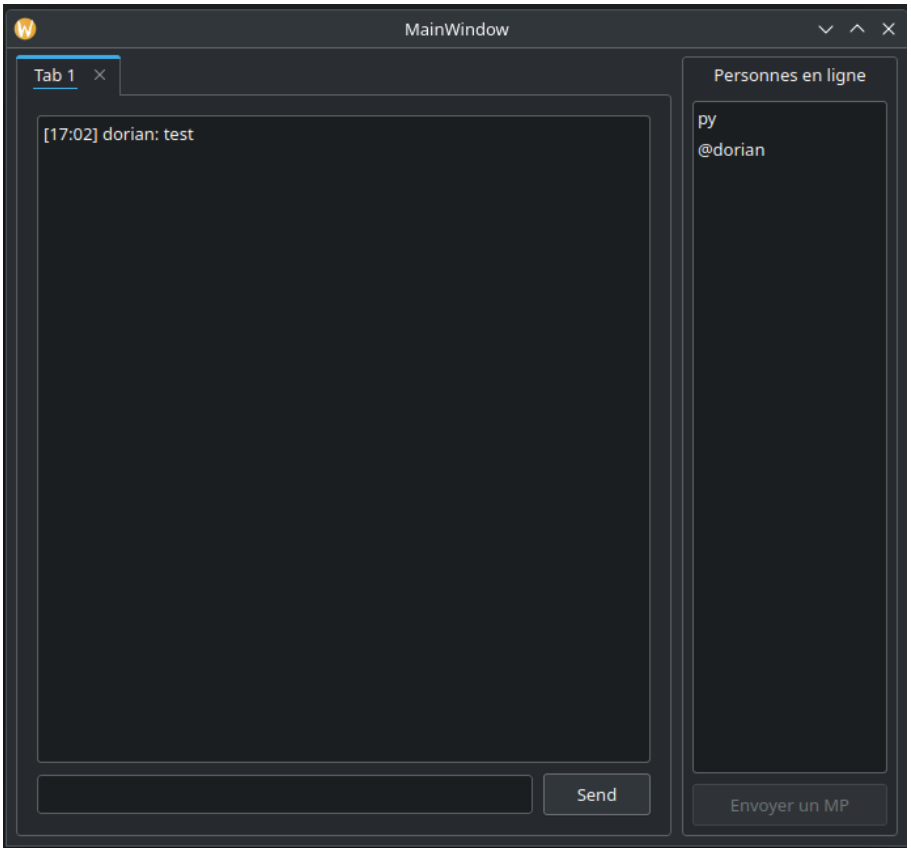

FIGURE 10 – L'interface complète

Pour créer cette interface, on utilisera un `HorizontalLayout`, avec à gauche un `TabWidget` et à droite un `VerticalLayout` qui contiendra tous les éléments.

Avec cette approche, la liste des utilisateurs connectés prend la moitié de la fenêtre, ce qui n'est pas forcément souhaitable. Cela peut être réglé en modifiant la propriété `LayoutStretch` du `HorizontalLayout`. Par défaut elle vaut "0,0" mais on peut la mettre par exemple à "3,1", ce qui signifie que l'élément de gauche prend 3 fois plus de place que l'élément de droite. Vous pouvez bien sûr ajuster ce ratio.

6.2 Liste des personnes connectées

La liste des personnes connectées à droite peut être remplie grâce au signal `IRCworker.userList`, qui donne en argument une liste de tous les pseudos des personnes en ligne. Ce signal est déclenché quand le serveur nous donne la list au moment de se connecter. Il est aussi possible de redemander la liste en appelant `IRCworker.requestUserList()`

Lorsqu'une personne se connecte ou se déconnecte, il faut également mettre à jour cette liste. On peut utiliser les signaux `IRCworker.connected` et `IRCworker.disconnected`. Ces deux signaux donnent en argument un `dict` qui contient entre autres l'heure de l'évènement (`time`) et le nom de la personne (`nickname`). Pour rappel, en Python on peut récupérer une valeur du `dict` en faisant `mon_dict["clé"]`

Lorsqu'on clique sur "Envoyer un MP", il faut créer un nouvel onglet et mettre un nouveau `ChatWidget` dedans. Cela peut se faire avec le code suivant :

```
target = ... # Nom de la personne avec qui discuter
newTab = QWidget()
gridLayout = QGridLayout(newTab)
self.tabWidget.addTab(newTab, target)
chat = ChatWidget(newTab)
self.channels[target] = chat
gridLayout.addWidget(chat,0,0,1,1) # Permet de faire en sorte que le
    ↪ widget prenne toute la taille de l'onglet
```

Il faudra créer le `dict` *channels* qui servira de moyen pour accéder aux widgets de chaque page (en faisant par exemple `self.channels["pseudo"].sendButton`). Au moment d'envoyer le message, au lieu de simplement appeler `IRCworker.sendMessage(message)`, vous pouvez préciser en deuxième paramètre le destinataire.

Pour mettre les bons messages aux bons endroits, il faudra utiliser des signaux plus évolués que `IRCworker.basicEvent` (vous pouvez le garder et juste print ces évènements

dans la console pour aider à débugger). En plus des signaux décrits avant, vous avez le signal `IRCworker.messageReceived` qui donne en argument un `dict` avec les valeurs suivantes :

- `time` : L'heure à laquelle le message a été reçu (en string, format "HH :MM")
- `message` : Le message en lui-même
- `nickname` : Le pseudo de la personne qui a envoyé le message
- `channel` : "#main" si c'est le chat principal, sinon le pseudo de la personne qui a envoyé le message [3]

Pour pouvoir facilement envoyer les messages aux bons endroits, vous pouvez ajouter le salon principal dans le `dict` *channels* (`self.channels["#main"]=self.tab.chatWidget` par exemple, à adapter aux noms que vous utilisez)) puis lorsque vous recevez un message, de simplement appeler `self.channels[eventData["channel"]].chatTextEdit.append(...)`, en adaptant aux noms que vous utilisez.

Enfin, si vous le souhaitez, vous pouvez rajouter des petits comportements supplémentaires :

- Faire en sorte que les onglets contiennent le nom du salon
- Faire en sorte que le bouton "Envoyer un MP" ne soit cliquable que si vous avez sélectionné quelqu'un dans la liste (éventuellement quelqu'un d'autre que vous)
- Rendre les onglets fermables (propriété `tabsCloseable`). Lorsque l'on clique sur le bouton de fermeture, cela déclenche le signal `tabCloseRequested` qu'il faudra traiter, en supprimant l'onglet en question
- Si quelqu'un vous envoie un MP alors que l'onglet n'est pas ouvert, créer l'onglet avec le message reçu

3. Si le message est dans un autre canal public, ce sera aussi un nom commençant par # mais on ne s'en sert pas dans ce TP.

7 ETAPE 5 : BONUS

7.1 Qt material

La librairie Qt material permet de rajouter du style facilement à votre application.

Installez et appliquer une feuille de style à l'aide de la documentation suivante : `https://pypi.org/project/qt-material/`

Observez le changement de style et choisissez le theme qui vous plaît le plus :) .

7.2 PyInstaller

Utilisez les instructions du cours (`https://slides.com/red4game/code/fullscreen`) section déploiement ainsi que la documentation du paquet (`https://pyinstaller.org/en/stable/`)

Installez PyInstaller puis essayer de créez le fichier d'installation du TP qui se trouvera dans le dossier /dist/. Essayer d'installer comme dans le cours puis avec le flag –**onefile** et observez la différence entre les deux méthodes.