

SAE 3.02 Développer des applications communicantes

DOCUMENTATION CLIENT

SAMUEL ROLLI RT222

SOMMAIRE

I.	Introduction	3
II.	Manuel d'utilisation	3
1)	Établir une connexion client-serveur	4
2)	Se connecter ou s'inscrire	4
3)	Discuter	6
III.	Explication du code	7
1)	" <i>main window</i> "	7
2)	" <i>Accueil window</i> "	8
3)	" <i>sign-in window</i> "	9
4)	" <i>Login window</i> "	10
5)	" <i>Discussion window</i> "	11
IV.	Limitation & Réflexion.....	14
1)	Sécurité	14
2)	Limitation	15

Table des figures

Figure 1: tag "#MODIFY4PROD"	3
Figure 2: page d'accueil.....	4
Figure 3: erreur de connexion.....	4
Figure 4: connexion établie.....	4
Figure 5: page d'inscription.....	4
Figure 6: erreur de pseudo.....	4
Figure 7: erreur de mot de passe.....	4
Figure 8: page de connexion	5
Figure 9: mot de passe incorrecte.....	5
Figure 10: erreur de double connexion.....	5
Figure 11: erreur de ban	5
Figure 12: interface graphique.....	6
Figure 13: accès refusé au channel Blabla	6
Figure 14: code de la fenêtre principal	7
Figure 15: génération de la fenêtre d'accueil.....	8
Figure 16: méthode la fenêtre d'accueil	8
Figure 17: génération de la fenêtre d'inscription.....	9
Figure 18: méthode de la fenêtre d'inscription	9
Figure 19: génération de la fenêtre de connexions	10
Figure 20: méthodes de la fenêtre de connexion	10
Figure 21: génération de la fenêtre de discussion	11
Figure 22: méthode "Get channel"	11
Figure 23: méthode start thread & réception de messages	12
Figure 24: méthodes d'accès aux channels, vérification d'accès, "show error pop-up" & de validation de l'accès aux channels.....	13
Figure 25: méthode de récupération d'historique.....	14
Figure 26: méthode de récupération de saisie & d'envoi de message	14
Figure 27: retours du serveur sans bug.....	15
Figure 28: retours du serveur avec erreur de concaténation	15

I. Introduction

Ce document vise à vous présenter le client que j'ai créé pour la SAE 3.02 : Développer des applications communicantes.

Dans un premier temps, nous verrons comment utiliser le client, d'un point de vue utilisateur.

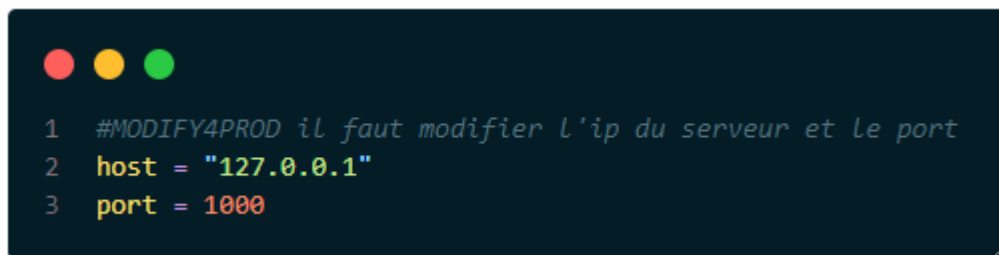
Puis, dans un second temps, je présenterai une explication du fonctionnement du code, et de ses fonctions.

Enfin, je vous proposerai une courte réflexion sur les limites de ce client et sur les options qui n'ont pas pu être mises en place.

II. Manuel d'utilisation

1) Préparer son client

Tout d'abord, il est très important de modifier les paramètres identifiés par le tag "*#MODIFY4PROD*" (modifier pour la version production), sur le client seul l'adresse IP et le port du serveur sont concernés.

A screenshot of a code editor with a dark background. At the top left, there are three colored circles: red, yellow, and green. Below them, there are three lines of code, each preceded by a number in a light blue font. The first line is a comment in light blue: *#MODIFY4PROD il faut modifier l'ip du serveur et le port*. The second line is *host = "127.0.0.1"* in yellow. The third line is *port = 1000* in yellow.

```
1  #MODIFY4PROD il faut modifier l'ip du serveur et le port
2  host = "127.0.0.1"
3  port = 1000
```

Figure 1: tag "*#MODIFY4PROD*"

Ensuite, vous remarquerez que l'exécution du client est impossible à ce point, en effet il manque des paquets pip à installer. Pour ce faire, vous pouvez utiliser la commande "*pip install -r requirement.txt*" avec "*requirement.txt*" le fichier texte présent dans le dossier contenant les fichiers client et serveur.

Installer les ces paquets à la main est possible, mais peut poser un problème de compatibilité, c'est pourquoi il est préférable d'installer uniquement les versions que j'ai utilisées lors du développement (celles présentes dans le fichier *requirement.txt*).

2) Établir une connexion client-serveur

Après avoir lancé l'application, vous allez être dirigés vers une page d'accueil vous proposant de vous connecter ou de vous inscrire. Là vous allez remarquer que l'application ne vous laissera pas vous inscrire ou vous connecter si vous n'avez pas connecté le serveur en amont.

La première étape sera donc de cliquer sur "connecter le serveur". Si le serveur est lancé, un message vert (cf. figure 4) vous l'indiquera, dans le cas contraire, un message rouge (cf. Figure 3) vous invitera à vérifier le bon fonctionnement de votre serveur.

Connected to server !!

Figure 4: connexion établie

server offline

Figure 3: erreur de connexion

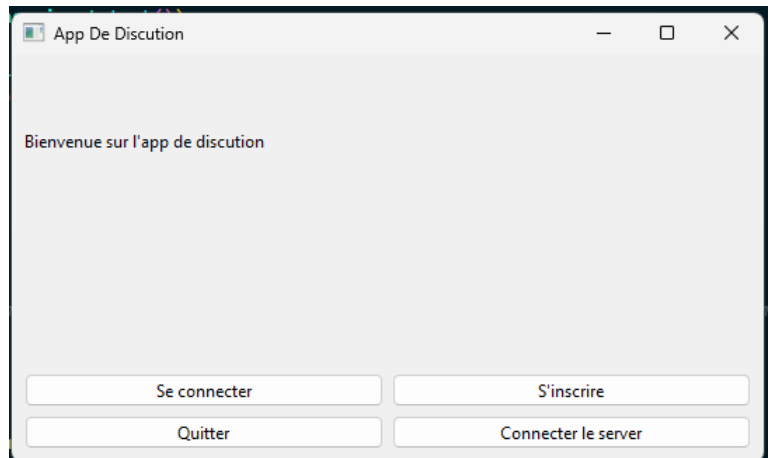


Figure 2: page d'accueil

3) Se connecter ou s'inscrire

a) S'inscrire

Une fois que le serveur est bel et bien connecté, nous pouvons passer à l'étape suivante, se connecter ou s'inscrire. Dans un premier temps, il est nécessaire de s'inscrire, car aucun compte n'est créé pour vous à votre première connexion.

Vous allez devoir maintenant remplir les champs "Username", "Password" et "Password confirmation" de la page d'inscription (cf. figure 5).

Vous remarquerez que, vous ne pouvez pas utiliser un "Username" s'il est déjà pris par quelqu'un (cf. figure 6), vous ne pourrez pas poursuivre votre connexion si les deux mots de passe ne sont pas absolument identiques (cf. figure 7).

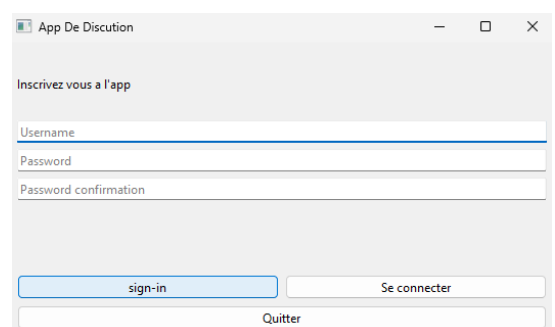


Figure 5: page d'inscription



Figure 6: erreur de mot de passe

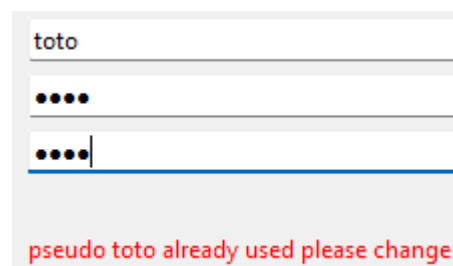


Figure 7: erreur de pseudo

Si tous les champs sont bien remplis et qu'aucune erreur n'apparaît, vous pourrez accéder à l'application et commencer à discuter, mais c'est un point que nous verrons plus tard.

b) Se connecter

A présent, nous allons voir comment se connecter. Sur la page d'accueil (cf. figure 2), il vous faudra cliquer sur "se connecter". Vous serez alors dirigé vers une page de connexion (cf. figure 8) et vous pourrez remplir votre "username" ainsi que votre "password". Vous remarquerez qu'il vous sera impossible de vous connecter avec un mot de passe incorrect, cela vous le sera indiqué par un message d'erreur en rouge (cf. figure 9) et qu'il vous sera aussi impossible de vous connecter si vous êtes déjà connecté (cf. figure 10). De plus, il est possible d'être banni du serveur (cf. Figure 11).

Wrong password for toto. Please retry.

Figure 9: mot de passe incorrecte

toto is already connected. Please logout from your other devices.

Figure 10: erreur de double connexion

toto is banned for 1.5 minutes

Figure 11: erreur de ban

(Note : le temps est affiché en fraction de minute 1,5 minute = 1 minute 30 secondes)

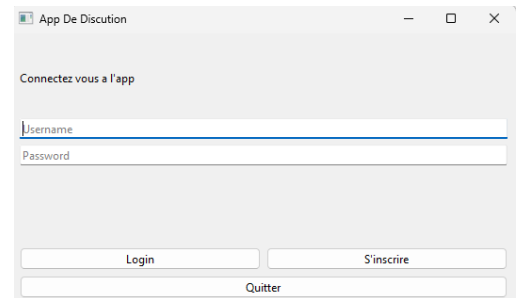


Figure 8: page de connexion

4) Discuter

a) Présentation des différents champs

L'application (cf. figure 12) se présente de la sorte : en haut, un bouton pour se déconnecter du compte et se connecter avec un autre compte, il renvoie sur la fenêtre de bienvenue (cf. figure 2). Le bouton "quitter" ferme la fenêtre et arrête le client.

Ensuite, la liste des différents channels et messages privés se présente comme une liste de boutons : il suffit simplement de cliquer sur un bouton pour accéder au fil de messages avec le correspondant. À gauche de cette liste se trouve la zone où seront affichés les messages. En dessous de celle-ci se trouve la zone de composition des messages, une simple pression de la touche "entrée", permet d'envoyer le message.

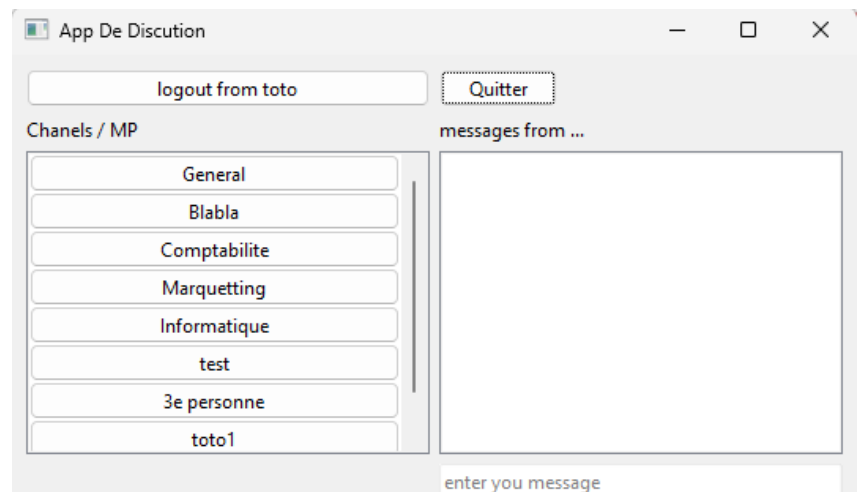


Figure 12: interface graphique

Attention, suite à un bug, il est important de ne cliquer qu'une fois sur le bouton d'un Channel pour y accéder, sinon le client envoie un flux de messages au serveur, ce qui a tendance à surcharger le serveur, surtout en cas de multi-clients. Si l'accès à un channel (General, Blabla, Comptabilité, Marqueting ou Informatique) vous est refusé, vous verrez apparaître le pop-up suivant (cf. figure 13) qui vous indiquera que vous devez demander l'accès, vous pouvez ignorer ce message en cliquant sur "OK" ou demander l'accès en cliquant sur "request access to ..." avec le nom du channel.

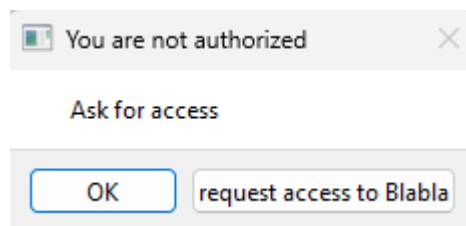


Figure 13: accès refusé au channel Blabla

III. Explication du code

L'architecture de mon application est la suivante, une fenêtre principale est créée, elle permet de donner les dimensions de la fenêtre, et d'appeler les autres fenêtres au moment voulu. Par la suite, j'ai créé une classe par fenêtre, une pour le log-in, sign-in et la discussion en elle-même.

1) "main window"

Comme dit au point précédent, la fenêtre principale, ne sert uniquement à déclarer les autres fenêtres comme étant des widgets. Cela me permet d'avoir un contrôle total sur mes fenêtres et d'éviter d'avoir à régénérer une fenêtre à chaque fois que je souhaite en changer. Nous pouvons remarquer les méthodes "switch" qui me permettent de définir quelle fenêtre est affichée.

La ligne 21 me permet de définir l'accueil comme étant la fenêtre par défaut.

Seule la méthode "switch_discussion" diffère des autres, avec l'appel de la méthode "get_channel()" qui me permet, une fois que le client est bien identifié, de faire une demande au serveur pour recevoir la liste des channels.

```

1  class mainWindow(QMainWindow):
2      global connected
3
4      def __init__(self):
5          super().__init__()
6          self.central_widget = QStackedWidget()
7          self.setWindowTitle("App De Discution")
8          self.setCentralWidget(self.central_widget)
9          self.login = login_window(self)
10         self.signin = signin_window(self)
11         self.acceuil = accueil_window(self)
12         self.discution = discutionwindow(self)
13
14         self.stacked_widget = self.central_widget
15         self.stacked_widget.setLayout(self.stacked_widget.layout())
16
17         self.central_widget.addWidget(self.acceuil)
18         self.central_widget.addWidget(self.login)
19         self.central_widget.addWidget(self.signin)
20         self.central_widget.addWidget(self.discution)
21         self.central_widget.setCurrentWidget(self.acceuil)
22         self.server_socket = server_socket
23
24
25
26
27     def switch_acceuil(self):
28         self.central_widget.setCurrentWidget(self.central_widget.widget(0))
29
30     def switch_login(self):
31         self.central_widget.setCurrentWidget(self.central_widget.widget(1))
32
33     def switch_signin(self):
34         self.central_widget.setCurrentWidget(self.central_widget.widget(2))
35
36     def switch_discussion(self):
37
38         self.central_widget.setCurrentWidget(self.central_widget.widget(3))
39         self.discution.get_chanel()

```

Figure 14: code de la fenêtre principal

2) "Accueil window"

La fenêtre d'accueil n'est composée uniquement de 4 boutons (cf. figure 2), et comme vu précédemment, est la fenêtre par défaut.

Ses boutons renvoient vers des fonctions (cf. figure 16)

```

1  def connect(self):
2      self.error_message.clear()
3      try:
4          server_socket.connect((host, port))
5      except ConnectionRefusedError:
6          self.show_error_message("server offline")
7      else:
8          global connected
9
10         connected = True
11         self.connected_message.setText("Connected to server !!")
12
13     def show_error_message(self, message):
14         self.error_message.setText(message)
15
16     def login(self):
17         if connected:
18             self.parentWidget().parentWidget().switch_login()
19         else:
20             self.show_error_message("please connect the server first")
21
22     def signin(self):
23         if connected:
24             self.parentWidget().parentWidget().switch_signin()
25         else:
26             self.show_error_message("please connect the server first")
27
28
29     def quit(self):
30         global stop_app
31         stop_app= True
32         QApplication.exit(0)

```

Figure 16: méthode la fenêtre d'accueil

le cas, fait appel à la méthode de "main window" pour changer la fenêtre affichée.

Enfin, la méthode "quit", sert uniquement à arrêter l'application.

```

1  class accueil_window(QMainWindow):
2      def __init__(self, parent=None):
3          super(accueil_window, self).__init__(parent)
4          self.setGeometry(200,200,400,300)
5          self.central_widget = QWidget(self)
6          self.setCentralWidget(self.central_widget)
7          self.layout = QGridLayout(self.central_widget)
8
9          self.Text_accueil = QLabel("Bienvenue sur l'app de discussion", self)
10         self.error_message = QLabel("", self)
11         self.error_message.setStyleSheet("color: red;")
12         self.connected_message = QLabel("", self)
13         self.connected_message.setStyleSheet("color: green;")
14         self.log_in_button = QPushButton("Se connecter",self)
15         self.log_in_button.clicked.connect(self.login)
16         self.sign_in_button = QPushButton("S'inscrire",self)
17         self.sign_in_button.clicked.connect(self.signin)
18         self.close_button = QPushButton("Quitter",self)
19         self.close_button.clicked.connect(self.quit)
20         self.connect_button = QPushButton("Connecter le server",self)
21         self.connect_button.clicked.connect(self.connect)
22         self.layout.addWidget(self.Text_accueil,0,0)
23         self.layout.addWidget(self.error_message,1,0)
24         self.layout.addWidget(self.connected_message,1,0)
25         self.layout.addWidget(self.log_in_button, 2,0)
26         self.layout.addWidget(self.sign_in_button,2,1)
27         self.layout.addWidget(self.close_button,3,0)
28         self.layout.addWidget(self.connect_button,3,1)

```

Figure 15: génération de la fenêtre d'accueil

La méthode "connect", sert à vérifier que le serveur est bien allumé et fonctionnel. Si le serveur est bien connecté, elle affichera le message vert (cf. figure 3) dans le cas contraire, elle fera appel à la méthode suivante, "show error message". Cette méthode sert uniquement à afficher un message, quelle prend en argument, en rouge.

Ensuite la méthode "login" et "signin" qui vérifie que le serveur est connecté, et, si c'est

3) "sign-in window"

Comme expliqué précédemment, elle est composée de 3 champs de texte et de 3 boutons.

Les trois champs de textes sont :

- "user input" pour mettre le nom d'utilisateur
- "password input" pour définir son code
- "password confirmation input", pour, vous l'aurez compris, confirmer son mot de passe

Les boutons, quant à eux, servent à valider l'inscription (même si la ligne 20 de la figure 17 nous permet de presser "entrée" dans le champ de confirmation de mot de passe pour automatiquement cliquer sur le bouton "sign-in"), à changer vers la fenêtre de connexion, si l'on s'est trompé de bouton sur la page d'accueil, et le dernier à quitter l'application.

```

1 def signin(self):
2     user = self.user_input.text()
3     password = self.password_input.text()
4     password_confirmation = self.password_confirmation_input.text()
5     if password and password_confirmation and user:
6         if password != password_confirmation:
7             self.show_error_message(message="error password")
8         else:
9             message = ":signin" + user + ":" + password
10            try:
11                server_socket.send(message.encode())
12                print("message envoyé")
13            except ConnectionAbortedError or ConnectionResetError:
14                self.show_error_message("Server has disconnected")
15            else:
16                try:
17                    print("attente du message")
18                    message = self.parent().parent().server_socket.recv(1024).decode()
19                    print(f"message reçu (message)")
20                except ConnectionAbortedError:
21                    self.show_error_message("No server connected")
22                except ConnectionResetError:
23                    self.show_error_message("Server has stopped")
24                else:
25                    if message == "signed-in":
26                        global username
27                        username = user
28                        self.parentWidget().parentWidget().switch_discussion()
29                    elif message == "already":
30                        self.show_error_message(f"pseudo {user} already used please change")
31
32            elif not user:
33                self.show_error_message(message="please enter a valid username")
34            elif not password or password_confirmation:
35                self.show_error_message(message="please enter a password")
36
37
38
39 def show_error_message(self, message):
40     self.error_message.setText(message)
41
42 def correction(self):
43     self.error_message.clear()
44
45 def quit(self):
46     global stop_app
47     stop_app = True
48     QApplication.exit(0)

```

Figure 18: méthode de la fenêtre d'inscription

```

1 class signin_window(QMainWindow):
2     def __init__(self, parent=None):
3         super(signin_window, self).__init__(parent)
4         self.setGeometry(200,200,400,300)
5         central_widget = QWidget(self)
6         self.setCentralWidget(central_widget)
7         self.layout = QGridLayout(central_widget)
8
9         self.Text_accueil = QLabel("Inscrivez vous a l'app", self)
10        self.user_input = QLineEdit(self)
11        self.user_input.setPlaceholderText("Username")
12        self.password_input = QLineEdit(self)
13        self.password_input.setPlaceholderText("Password")
14        self.password_input.setEchoMode(QLineEdit.EchoMode.Password)
15        self.password_input.textChanged.connect(self.correction)
16        self.password_confirmation_input = QLineEdit(self)
17        self.password_confirmation_input.setPlaceholderText("Password confirmation")
18        self.password_confirmation_input.setEchoMode(QLineEdit.EchoMode.Password)
19        self.password_confirmation_input.textChanged.connect(self.correction)
20        self.password_confirmation_input.returnPressed.connect(self.signin)
21        self.error_message = QLabel("", self)
22        self.error_message.setStyleSheet("color: red;")
23        self.signin_button = QPushButton("sign-in", self)
24        self.signin_button.clicked.connect(self.signin)
25        self.login_button = QPushButton("Se connecter", self)
26        self.login_button.clicked.connect(parent.switch_login)
27        self.close_button = QPushButton("Quitter", self)
28        self.close_button.clicked.connect(self.quit)
29
30
31        self.layout.addWidget(self.Text_accueil,0,0,1,2)
32        self.layout.addWidget(self.user_input,1,0,1,2)
33        self.layout.addWidget(self.password_input,2,0,1,2)
34        self.layout.addWidget(self.password_confirmation_input,3,0,1,2)
35        self.layout.addWidget(self.error_message,4,0,1,2)
36        self.layout.addWidget(self.signin_button,5,0)
37        self.layout.addWidget(self.login_button,5,1)
38        self.layout.addWidget(self.close_button,6,0,1,2)

```

Figure 17: génération de la fenêtre d'inscription

La méthode "signin", va récupérer le contenu des différents champs de texte, et commencer par vérifier si tous les champs de texte sont remplis, puis si les mots de passes correspondent, si ce n'est pas le cas, elle fait appel à la méthode "show error message". Ensuite, elle envoie au serveur, un message protocolaire (reconnaissable car il commence par ":" et est suivi d'un mot explicite, ici ":signin") pour demander confirmation au serveur du nom d'utilisateur et du mot de passe. Si le serveur, retourne un message positif, elle fait appel à la méthode de "main window" pour passer à la fenêtre de discussion. Si ce n'est pas le cas, par exemple si le nom d'utilisateur est déjà pris, elle fait appel à la méthode "show error message" pour demander à l'utilisateur de changer de nom d'utilisateur.

Ensuite la méthode "show error message" et

"quit" sont identiques à la fenêtre précédente.

La méthode "correction" permet de faire disparaître le message d'erreur, pour éviter d'induire l'utilisateur en erreur.

4) "Login window"

Au même titre que la fenêtre d'inscription vue au point 3, la fenêtre de connexion est assez simple : d'abord deux champs de texte, un pour le nom d'utilisateur et un pour le mot de passe. Ensuite trois boutons, un pour quitter, un pour valider les champs (même si la ligne 17 de la figure 19 nous indique que taper sur la touche entrée, déclenche automatiquement la pression du dit bouton) et en fin, un bouton pour aller sur la fenêtre de connexion si l'on s'est trompé sur la fenêtre d'accueil.

La méthode "login", comme la "signin" avant elle, va commencer par vérifier que tous les champs sont remplis, puis va demander au serveur une vérification sur le login et le mot de passe avec un message protocolaire : ":login". De là, quatre possibilités, soit le serveur renvoie "logged-in" et le client est au courant qu'il est donc bien connecté, soit "error" et donc le client peut demander à l'utilisateur de retaper son mot de passe ou son nom d'utilisateur, soit "already connected" où le client va pouvoir indiquer à l'utilisateur qu'il doit déconnecter ses autres appareils afin de connecter celui-ci. Et enfin "ban" avec le temps restant ce qui lève un message rouge.

Les méthodes "quit", "show error message" et "corrections" sont exactement les mêmes que pour la fenêtre d'inscription, vue au point précédent.

Figure 20: méthodes de la fenêtre de connexion

```

1 class login_window(QMainWindow):
2     def __init__(self, parent=None):
3         super(login_window, self).__init__(parent)
4         self.setGeometry(200,200,400,300)
5         central_widget = QWidget(self)
6         self.setCentralWidget(central_widget)
7         self.layout = QGridLayout(central_widget)
8
9         self.Text_acceuil = QLabel("Connectez vous a l'app", self)
10        self.user_input = QLineEdit(self)
11        self.user_input.setPlaceholderText("Username")
12        self.user_input.textChanged.connect(self.correction)
13        self.password_input = QLineEdit(self)
14        self.password_input.setPlaceholderText("Password")
15        self.password_input.setEchoMode(QLineEdit.EchoMode.Password)
16        self.password_input.textChanged.connect(self.correction)
17        self.password_input.returnPressed.connect(self.login)
18        self.error_message = QLabel("", self)
19        self.error_message.setStyleSheet("color: red;")
20        self.login_button = QPushButton("Login", self)
21        self.login_button.clicked.connect(self.login)
22        self.signin_button = QPushButton("S'inscrire",self)
23        self.signin_button.clicked.connect(parent.switch_signin)
24        self.close_button = QPushButton("Quitter",self)
25        self.close_button.clicked.connect(self.quit)
26
27        self.layout.addWidget(self.Text_acceuil,0,0,1,2)
28        self.layout.addWidget(self.user_input,1,0,1,2)
29        self.layout.addWidget(self.password_input,2,0,1,2)
30        self.layout.addWidget(self.error_message,3,0,1,2)
31        self.layout.addWidget(self.login_button,4,0)
32        self.layout.addWidget(self.signin_button,4,1)
33        self.layout.addWidget(self.close_button,5,0,1,2)

```

Figure 19: génération de la fenêtre de connexions

```

1 def quit(self):
2     global stop_app
3     stop_app = True
4     server_socket.send(":bye".encode())
5     QApplication.exit(0)
6
7 def login(self):
8     user = self.user_input.text()
9     password = self.password_input.text()
10
11     if user and password:
12         message = ":login" + user + ":" + password
13         try:
14             self.parent().parent().server_socket.send(message.encode())
15             print("message envoyé")
16         except ConnectionAbortedError or ConnectionResetError:
17             self.show_error_message("Server has disconnected")
18         else:
19             try:
20                 print("attente du message")
21                 message = self.parent().parent().server_socket.recv(1024).decode()
22                 print(f"message reçu {message}")
23             except ConnectionAbortedError:
24                 self.show_error_message("No server connected")
25             except ConnectionResetError:
26                 self.show_error_message("Server has stopped")
27             else:
28                 if message == 'logged-in':
29                     global username
30                     username = user
31                     self.parentWidget().parentWidget().switch_discussion()
32                 elif message == "error":
33                     self.show_error_message(f"Wrong password for {user}. Please retry.")
34                 elif message == 'already connected':
35                     self.show_error_message(f"{user} is already connected. Please logout from your other devices.")
36                 elif message[:3] == 'ban':
37                     time_left = message[3:]
38                     self.show_error_message(f"{user} is banned for {time_left} minutes")
39
40
41
42 def show_error_message(self, message):
43     self.error_message.setText(message)
44
45 def correction(self):
46     self.error_message.clear()

```

5) "Discussion window"

La fenêtre de discussion est la fenêtre la plus importante, car c'est elle qui contient l'application. C'est donc normal qu'elle demande le plus de codes, approximativement 240 lignes.

Tout d'abord, nous pouvons voir que la fenêtre de discussion principale possède très peu de déclaration d'éléments, par exemple la liste des channels ou la liste des messages est déclarée comme une liste vide, ce n'est que par la suite, et nous le verrons après que ces listes sont remplies.

```

1  def get_channels(self):
2      global username
3
4      if connected:
5          self.logout_button.setText(f"logout from {username}")
6          while not self.channel_list_got:
7              print("requesting channels")
8              message = f":channel_list:{username}"
9              try:
10                 server_socket.send(message.encode())
11                 print("message envoyé")
12             except ConnectionAbortedError or ConnectionResetError:
13                 self.show_error_message(message = "Server has disconnected")
14             else:
15                 try:
16                     print("attente du message")
17                     message = server_socket.recv(1024).decode()
18                     print(f"message reçu {message}")
19                     except ConnectionAbortedError or ConnectionResetError:
20                         print('ya une erreur, il faut que je fasse ça propre') #TODO ça
21                 else:
22                     self.channel_list_got = True
23                     message = message.split(":")
24                     self.tous_channels = message
25                     print(message)
26                     for i in message:
27                         if i != "":
28                             button_item = QPushButton(f"{i}", self)
29                             button_item.clicked.connect(lambda _, i=i :self.access_channel(i))
30
31                             list_widget_item = QListWidgetItem()
32                             list_widget_item.setSizeHint(button_item.sizeHint())
33
34                             self.channel_list.addItem(list_widget_item)
35                             self.channel_list.setItemWidget(list_widget_item, button_item)
36                     print('requesting starting receiving thread')
37                     self.start_thread()
38
39

```

Figure 22: méthode "Get channel"

```

1  class discussionwindow(QMainWindow):
2      def __init__(self, parent=None):
3          super(discutionwindow, self).__init__(parent)
4          self.channel_list_got = False
5          self.on_channel = ""
6          self.start_history = False
7          self.start_check = False
8          self.tous_channels = []
9          self.setGeometry(200,200,400,300)
10         central_widget = QWidget(self)
11         self.setCentralWidget(central_widget)
12         self.layout = QGridLayout(central_widget)
13
14         self.close_button = QPushButton("Quitter",self)
15         self.close_button.clicked.connect(self.quit)
16         self.logout_button = QPushButton("logout",self)
17         self.logout_button.clicked.connect(parent.switch_acceuil)
18         self.channel_list_titre = QLabel("Channels / MP", self)
19         self.channel_list = QListWidget(self)
20         self.message_list_titre = QLabel("messages from ...",self)
21         self.message_list = QListWidget(self)
22
23         self.message_input = QLineEdit(self)
24         self.message_input.setPlaceholderText("enter you message")
25         self.message_input.returnPressed.connect(self.message_enter)
26
27
28
29
30
31         self.layout.addWidget(self.logout_button,0,0,1,1)
32         self.layout.addWidget(self.close_button,0,1,1,1)
33         self.layout.addWidget(self.channel_list_titre,1,0,1,1)
34         self.layout.addWidget(self.message_list_titre,1,1,1,4)
35         self.layout.addWidget(self.channel_list,2,0,4,1)
36         self.layout.addWidget(self.message_list,2,1,4,4)
37         self.layout.addWidget(self.message_input,6,1,1,4)
38

```

Figure 21: génération de la fenêtre de discussion

Vient ensuite, la méthode la plus importante, puisqu'elle est appelée juste après que la fenêtre soit générée puis affichée (cf ligne 39 de la figure 14). Elle va demander au serveur la liste des utilisateurs ainsi que des channels, et créer un bouton par channel et fils de message privé, et les mettre dans la liste vide vue sur la

figure 21. Enfin, une fois qu'elle a fini de récupérer tous les éléments à afficher, elle va faire appel à la méthode "start thread" que nous allons voir. Chaque bouton renvoie vers la méthode "access channel" qui prend en argument le nom du channel ou du correspondant de message privé.

La méthode `"start thread"` sert à définir puis démarrer le thread de réception de messages. Elle fait donc directement appel à la méthode `"receive messages"`. Cette méthode n'a qu'un seul but : recevoir et traiter les messages, elle va par exemple être capable de détecter si un message est un protocole en testant à la ligne 17 (cf. figure 23) si le message commence par `":"`, puis en fonction du protocole faire les actions nécessaires. Ensuite s'il s'agit uniquement d'un message envoyé par un autre utilisateur, car nous le verrons dans la documentation dédiée au serveur plus en détails, le serveur renvoie l'intégralité des messages envoyé par les utilisateurs en broadcast sur les autres clients. C'est à eux de vérifier s'ils sont concernés par le message reçu, nous pouvons voir cela à la ligne 32 de la figure 23 avec le test qui regarde si le correspondant est bien celui que le client affiche (message privé ou channel) et s'il en est bien la destination, si ce n'est pas le cas, le message ne sera pas stocké et juste ignoré. De plus, nous le verrons plus tard, mais tous les messages envoyés ne sont jamais réellement stockés sur les clients eux même, ils sont stockés sur le serveur SQL, et les clients font une demande au serveur à chaque fois qu'ils veulent afficher un channel ou un fil de discussion privé pour avoir l'historique des messages.

```

1  def receive_messages(self):
2      print('started')
3      while not stop_app:
4          global connected
5          while connected:
6              if not self.start_history and not self.start_check:
7                  try:
8                      print('thread WAITING FOR MESSAGE')
9                      message = server_socket.recv(1024).decode()
10                     except ConnectionResetError:
11                         self.message_error("Server forcibly disconnected")
12                         connected = False
13                     except ConnectionAbortedError:
14                         self.message_error("Server forcibly disconnected")
15                         connected = False
16                 else:
17                     if message[:1] == ":":
18                         message = message[1:]
19                         if message[:13] == "start_history":
20                             self.start_history= True
21                             print(f"(tread receive) {message}")
22                         if message[:11] == "start_check":
23                             self.start_check= True
24                             print(f"(tread receive) {message}")
25                         if message[:12] == ":end_history":
26
27                             self.start_history = False
28                             self.start_check = False
29                     else:
30                         print(f"(tread receive) {message}")
31                         message = message.split(":"to:")
32                         if (self.on_channel == message[0] and username == message[1]) or
33                             (self.on_channel == message[1] and message[1] in self.tous_channels):
34                             result = message[0] + ' : ' + message[2]
35                             self.message_list.addItem(result)
36                             scrollbar = self.message_list.verticalScrollBar()
37                             scrollbar.setValue(scrollbar.maximum())
38
39
40  def start_thread(self):
41      receive_thread = threading.Thread(target=self.receive_messages)
42      receive_thread.start()
43

```

Figure 23: méthode `start thread` & réception de messages

J'ai décidé de mettre les méthodes `"access channel"`, `"check access"`, `"show error pop-up"`, `"ok no requesting"` et `"sub to channel"` dans le même screen shot car, selon moi, elles sont indissociables les unes des autres car travaillent intimement ensemble.

En effet la méthode `"access channel"` est appelée lorsque l'utilisateur clique sur le bouton correspondant à un channel. Si le test d'accès au dit channel est validé (nous allons voir ça plus en détail par la suite) la liste des messages affichés sera vidée, le texte `"message from ..."` sera changé et sera fait appel à la méthode d'historique des messages (cf. figure 25). Dans le cas où le test ne sera pas validé, la méthode d'affichage d'erreur sera utilisée pour informer l'utilisateur qu'il n'a pas accès au dit channel.

La méthode `"check access"` va envoyer un message protocolaire au serveur `":check_access..."` avec le pseudo de l'utilisateur et attendre la réponse. Si celle-ci est `":subed"` (qui veut dire abonné) elle en déduit que le client a accès

à ce channel. Au contraire, si la réponse est `":notsubed"` (pas abonné), l'accès est donc refusé. Par la suite si l'accès est accepté, nous avons vu que la méthode `"historique message"` est appelée. Dans le cas contraire c'est la méthode `"show error pop-up"` qui est appelée, celle-ci va prendre en argument le nom du channel pour lequel l'accès a été refusé et générer un pop-up avec deux boutons un `"ok"` et un `"request access to ..."` qui sont liés aux méthodes suivantes.

Les méthodes `"ok_no_requesting"` et `"sub_to_channel"` sont encore en cours de développement. Pour l'instant elles ne font qu'un print car je n'ai pas eu le temps de déboguer. Dans l'idéal, elles enverraient des messages protocolaires pour informer le serveur de la décision de l'utilisateur concernant le channel dont l'accès a été refusé.



```

1 def access_channel(self, channel):
2     if self.check_access(channel):
3         print('access granted')
4         self.message_list.clear()
5         self.message_list_titre.setText(f"messages from {channel}")
6         self.historique_message(channel = channel)
7         self.on_channel = channel
8     else:
9         self.show_error_popup(channel)
10
11 def check_access(self, channel):
12     check_finished = False
13     while not check_finished:
14         message = f":check_access{channel}"
15         server_socket.send(message.encode())
16
17         message = server_socket.recv(1024).decode()
18         if message == ":subed":
19             check_finished = True
20             return True
21         elif message == ":notsubed":
22             check_finished = True
23             return False
24
25 def show_error_popup(self, channel):
26     popup = QMessageBox(self)
27     popup.setWindowTitle('You are not authorized')
28     popup.setText('Ask for access')
29     sub_button = QPushButton(f"request access to {channel}", self)
30     ok_button = QPushButton("OK", self)
31     popup.addButton(sub_button, QMessageBox.ButtonRole.ActionRole)
32     popup.addButton(ok_button, QMessageBox.ButtonRole.AcceptRole)
33     #HACK here there is a problem, WHY
34     sub_button.clicked.connect(lambda _, i=channel: self.sub_to_channel(i))
35     ok_button.clicked.connect(self.ok_no_requesting)
36     popup.exec()
37
38 def ok_no_requesting(self):
39     # server_socket.send(':confirmation'.encode())
40     # time.sleep(0.1)
41     # server_socket.send(":ignore".encode())
42     print("ok")
43
44 def sub_to_channel(self, channel):
45     # server_socket.send(':confirmation'.encode())
46     # time.sleep(0.1)
47     # message = username + ":subbing:" + channel
48     # server_socket.send(message.encode())
49     print(f"Subbing to channel {channel}")

```

Figure 24: méthodes d'accès aux channels, vérification d'accès, `"show error pop-up"` & de validation de l'accès aux channels

Cette méthode, comme on l'a vu est appelée à chaque fois que le client fait une demande pour changer de channel. Nous reparlerons plus en détails de cette méthode dans la documentation du serveur, car l'essentiel de ce qui se passe à ce moment est du côté du serveur. Côté client, il fait une demande au serveur à l'aide d'un message protocolaire `":get_history..."` avec le nom du channel ou du fil de message privé. Puis il attend de recevoir les messages du serveur, et, pour chaque message (de la forme `"nom de l'auteur du message :said: message"`) il va les traiter un à un et les ajouter à la liste de messages vide créée à l'affichage de la page (cf. figure 21). Un fois que le serveur a fini de rendre tout l'historique, il envoie un message protocolaire `":end_history"` ce qui informe le client que l'historique est fini et que le thread de réception principale peut reprendre l'écoute sur le socket.

```

1 def historique_message(self, channel):
2     message = f":get_history{channel}"
3     try:
4         server_socket.send(message.encode())
5     except ConnectionAbortedError or ConnectionResetError:
6         self.show_error_message("Server has disconnected")
7     else:
8         history_finish = False
9         while not history_finish:
10            try:
11                message = server_socket.recv(1024).decode()
12                print(f"historique {message}")
13            except ConnectionAbortedError or ConnectionResetError:
14                print('ya une erreur, il faut que je fasse ça propre')
15            else:
16
17                if message[:12] == ":end_history":
18                    history_finish = True
19                    self.start_history = False
20                    self.start_check = False
21
22                elif message == ":start_history" or message == ":start_check":
23                    self.start_history = True
24                    self.start_check = True
25
26                elif message[:1] == ":":
27                    print(f'message {message} ignored')
28                else:
29                    message = message.split(':said:')
30                    result = message[0] + ' : ' + message[1]
31                    self.message_list.addItem(result)
32                    scrollbar = self.message_list.verticalScrollBar()
33                    scrollbar.setValue(scrollbar.maximum())
34                    scrollbar = self.message_list.verticalScrollBar()
35                    scrollbar.setValue(scrollbar.maximum())

```

Figure 25: méthode de récupération d'historique

```

1 def message_enter(self):
2     global username
3     if self.message_input.text():
4         self.message_list.addItem(f"{username} : {self.message_input.text()}")
5         self.send_message(message = self.message_input.text())
6         self.message_input.clear()
7         scrollbar = self.message_list.verticalScrollBar()
8         scrollbar.setValue(scrollbar.maximum())
9
10 def send_message(self, message):
11     message = message + ":to:" + self.on_channel
12     server_socket.send(message.encode())

```

Figure 26: méthode de récupération de saisie & d'envoi de message

En suite la méthode `"message_enter"` est appelée lorsque l'utilisateur appuie sur "entrée" dans la zone de saisie de message. Elle va d'abord vérifier que du texte a été entré dans la zone de saisie, puis en récupérer le contenu, elle va ensuite rajouter à la liste des messages, le contenu de la saisie ainsi que le pseudo de l'utilisateur, en suite elle fait appel a la méthode `"send_message"` qui prend en argument le contenu de la saisie. Enfin, elle vide la zone de

saisie et met la barre de défilement a la valeur maximale pour permettre d'afficher le dernier message en date (note: Dans PyQt6 la barre de défilement a un bug, la fonction `"scrollbar.maximum()"` ne met pas la valeur maximal, en effet il reste encore un peu de marge, que l'utilisateur doit défiler a la main)

IV. Limitation & Réflexion

1) Sécurité

Un des plus gros problèmes de sécurité est le fait que le mot de passe de l'utilisateur est envoyé en clair avec son pseudo, je n'ai pas eu le temps de mettre en place une clé de hash ou autre.

De plus, le fait que les messages soient envoyés en broadcast à tous les utilisateurs connectés à ce moment pose un problème : si un utilisateur décidait de modifier son client, il pourrait avoir accès à tous les messages de tout le monde (messages privés & channels auxquels il ne devrait pas avoir accès).

Enfin, je projette de mettre en place une sécurité qui empêcherait tous les utilisateurs de commencer leur message par `":` pour qu'ils ne puissent pas simuler des messages protocolaires auprès du serveur.

Pour finir, comme dit précédemment, je n'ai pas réussi à faire la sécurisation des channels, les channels demandant une vérification d'un administrateur sont simplement inaccessibles.

2) Limitation

Par manque de temps, plusieurs options n'ont pas pu être mise en place :

- La possibilité de Kick / Ban / time out
- La possibilité de demander l'accès à un channel sur contrôle d'accès

Ces options manquantes sont dues au fait que PyQt6 n'a pas un retour d'erreur très performant : lorsqu'une erreur survient il est difficile de comprendre l'origine de cette erreur ce qui est très chronophage, ce qui sur un projet de ce type est très compromettant.

Vous trouverais tout au long du code des début de code, mis en commentaire, qui n'ont pas été aboutis, notamment pour la possibilité de demander l'accès a certain channels. Ainsi que des tag "TODO" qui me permettes de retrouver ce qu'il fallait que je finisse à la dernière session de travail.

Ensuite, le client a une tendance à freeze et à exécuter ou concaténer (cf. figure 28) une quantité aléatoire d'actions en une fraction de seconde, ce qui à tendance à faire crasher le serveur.

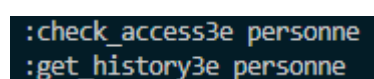


Figure 27: retours du serveur sans bug

Ici, nous pouvons voir que le client a fait une demande pour accéder au fil de discussion avec l'utilisateur "3^e personne", deux messages protocolaires ont été envoyés depuis le client : ":check_access3e personne" pour être sûr que l'utilisateur avait accès à ce fil de discussion, et ":get_history3e personne" pour que le serveur serve

l'historique au client correctement.

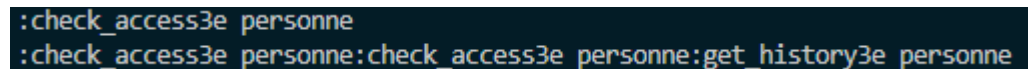


Figure 28: retours du serveur avec erreur de concaténation

Sur ce 2^e screen shot, nous pouvons voir qu'une demande de vérification d'accès a été faite à l'aide d'un message protocolaire ":check_acces" mais qu'il a été suivi, non pas d'un message protocolaire ":get_history" mais d'une concaténation de deux messages protocolaires ":check_acces" et d'un message protocolaire ":get_history".

Ce problème est particulièrement perturbant car il a pour conséquence de ne pas servir correctement l'historique de certaines conversations et d'empêcher le serveur de fonctionner optimalement. Si cette erreur se produit, il est préférable de relancer le serveur et le client.