Appli desktop de Chat avec CustomTkinter

CustomTkinter est un framework basé sur **Tkinter**, qui permet de créer des interfaces graphiques modernes et stylisées en utilisant des composants personnalisés (widgets). Il ajoute des fonctionnalités telles que les thèmes sombres et clairs, ainsi qu'un style plus moderne pour les applications de bureau.

sites de référence:

- customtkinter

Installer CustomTkinter

Installez `customtkinter` :

```
pip install customtkinter
```

Créer l'application avec CustomTkinter

Voici un exemple pour créer une application de chat utilisant **Flask-SocketIO** et **CustomTkinter**.

```
# desktop_chat_app_customtkinter.py

import sys
import threading
import asyncio
import socketio
import customtkinter

# Configuration de l'application CustomTkinter
class ChatClient(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        self.title("Flask-SocketIO Chat Client")
        self.geometry("400x300")

        # Configuration du layout
        self.chat_display = customtkinter.CTkTextbox(self, width=380,
height=200)
        self.chat_display.pack(padx=10, pady=10)

        self.input_field = customtkinter.CTkEntry(self, width=280)
        self.input_field.pack(side="left", padx=(10, 0), pady=10)

        self.send_button = customtkinter.CTkButton(self, text="Send",
command=self.send_message)
        self.send_button.pack(side="right", padx=(0, 10), pady=10)
```

```
        # Connexions des événements
        self.input_field.bind("<Return>", lambda event:
self.send_message())

        # Configuration du client SocketIO
        self.sio = socketio.Client()
        self.sio.on('message', self.receive_message)

        # Connexion au serveur
        self.connect_to_server()

    def connect_to_server(self):
        try:
            self.sio.connect('http://127.0.0.1:5000')
            self.chat_display.insert("end", "Connected to the
server.\n")
        except Exception as e:
            self.chat_display.insert("end", f"Connection error: {e}\n")

    def receive_message(self, msg):
        self.chat_display.insert("end", f"Server: {msg}\n")
        self.chat_display.yview("end")  # Scroll vers le bas

    def send_message(self):
        message = self.input_field.get()
        if message:
            self.sio.send(message)
            self.chat_display.insert("end", f"You: {message}\n")
            self.chat_display.yview("end")
            self.input_field.delete(0, "end")

    def close_event(self):
        # Ferme proprement la connexion SocketIO
        if self.sio.connected:
            self.sio.disconnect()
        self.destroy()

# Lancer l'application de bureau CustomTkinter
def main():
    app = ChatClient()
    app.protocol("WM_DELETE_WINDOW", app.close_event)
    app.mainloop()

if __name__ == "__main__":
    main()
```

PROF

Configuration du serveur Flask

Assurez-vous que le serveur Flask tourne comme décrit dans l'exemple précédent.

Exécuter l'application

Lancez l'application de bureau avec la commande :

```
python3 desktop_chat_app_customtkinter.py
```

Explications

1. **CustomTkinter** : Les widgets de base comme `CTk`, `CTkTextbox`, `CTkEntry`, et `CTkButton` sont utilisés pour créer une interface moderne.
2. **SocketIO** : Le client se connecte au serveur Flask via SocketIO pour envoyer et recevoir des messages.
3. **Gestion des événements** : Les événements `Return` (entrée) et les clics sur le bouton d'envoi sont gérés pour envoyer les messages au serveur.

AppManager - Transitions entre fénêtres

Les importations circulaires surviennent lorsque deux modules essaient de s'importer mutuellement. Pour résoudre ce problème dans une application avec deux classes (`ChatClient` et `LoginClient`) qui ont besoin d'accéder l'une à l'autre, il existe plusieurs méthodes. Voici quelques-unes des solutions les plus courantes :

Solution 1 : Importations différées (importations locales)

En Python, vous pouvez importer des modules ou des classes à l'intérieur des méthodes, ce qui diffère l'importation jusqu'à ce que la méthode soit appelée. Cela permet d'éviter les importations circulaires car les modules ne sont pas importés au moment de la définition de la classe.

Fichier `login_client.py`

```
# login_client.py
import customtkinter as ctk

class LoginClient(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Connexion")

        # Interface simplifiée pour l'exemple
        self.label_username = ctk.CTkLabel(self, text="Nom
d'utilisateur")
        self.label_username.pack()

        self.entry_username = ctk.CTkEntry(self)
        self.entry_username.pack()

        self.login_button = ctk.CTkButton(self, text="Connexion",
command=self.open_chat_client)
        self.login_button.pack()
```

```
    def open_chat_client(self):
        from chat_client import ChatClient  # Importation locale pour
éviter le problème d'importation circulaire
        self.destroy()  # Fermer la fenêtre de connexion
        chat_client = ChatClient()  # Créer et ouvrir la fenêtre de chat
        chat_client.mainloop()
```

Fichier `chat_client.py`

```
# chat_client.py
import customtkinter as ctk

class ChatClient(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Chat")

        # Interface simplifiée pour l'exemple
        self.label_chat = ctk.CTkLabel(self, text="Bienvenue dans le
chat")
        self.label_chat.pack()

        self.logout_button = ctk.CTkButton(self, text="Déconnexion",
command=self.return_to_login)
        self.logout_button.pack()

    def return_to_login(self):
        from login_client import LoginClient  # Importation locale pour
éviter le problème d'importation circulaire
        self.destroy()  # Fermer la fenêtre de chat
        login_client = LoginClient()  # Créer et ouvrir la fenêtre de
connexion
        login_client.mainloop()
```

PROF

Solution 2 : Passer une instance de classe dans le constructeur

Une autre option consiste à passer une instance de `LoginClient` à `ChatClient` (et vice versa) en utilisant une sorte de gestionnaire central pour la navigation. Ce modèle fonctionne bien pour les applications complexes en permettant une gestion centralisée de la logique de transition.

Exemple avec un gestionnaire d'application

Créez un fichier principal, `app_manager.py`, qui gère la navigation entre les classes et maintient les importations séparées.

`app_manager.py`

```python
# app_manager.py
from login_client import LoginClient
from chat_client import ChatClient

class AppManager:
    def __init__(self):
        self.login_client = LoginClient(self)
        self.chat_client = ChatClient(self)

    def show_login(self):
        self.chat_client.withdraw()  # Masque la fenêtre de chat
        self.login_client.deiconify()  # Affiche la fenêtre de connexion

    def show_chat(self):
        self.login_client.withdraw()  # Masque la fenêtre de connexion
        self.chat_client.deiconify()  # Affiche la fenêtre de chat

    def run(self):
        self.show_login()
        self.login_client.mainloop()
```

Ensuite, adaptez les classes `LoginClient` et `ChatClient` pour accepter un paramètre `manager` dans leur constructeur, leur permettant d'accéder au gestionnaire pour gérer la navigation :

login_client.py

```python
# login_client.py
import customtkinter as ctk

class LoginClient(ctk.CTk):
    def __init__(self, manager):
        super().__init__()
        self.manager = manager
        self.title("Connexion")

        # Interface simplifiée pour l'exemple
        self.label_username = ctk.CTkLabel(self, text="Nom
d'utilisateur")
        self.label_username.pack()

        self.entry_username = ctk.CTkEntry(self)
        self.entry_username.pack()

        self.login_button = ctk.CTkButton(self, text="Connexion",
command=self.manager.show_chat)
        self.login_button.pack()
```

PROF

chat_client.py

```
# chat_client.py
import customtkinter as ctk

class ChatClient(ctk.CTk):
    def __init__(self, manager):
        super().__init__()
        self.manager = manager
        self.title("Chat")

        # Interface simplifiée pour l'exemple
        self.label_chat = ctk.CTkLabel(self, text="Bienvenue dans le
chat")
        self.label_chat.pack()

        self.logout_button = ctk.CTkButton(self, text="Déconnexion",
command=self.manager.show_login)
        self.logout_button.pack()
```

Lancer l'application

Enfin, pour lancer l'application, créez un fichier `main.py` :

main.py

```
# main.py
from app_manager import AppManager

if __name__ == "__main__":
    app_manager = AppManager()
    app_manager.run()
```

PROF

Cette solution avec un `AppManager` est propre et extensible pour des applications plus complexes, car elle centralise la gestion des transitions entre les fenêtres et permet d'éviter les importations circulaires.