**Vous avez dit :**

je veux une fenetre daffiche avec plusieurs onglet

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici un exemple de code Python utilisant PyQt6 qui crée une fenêtre avec plusieurs onglets. Chaque onglet contient un contenu différent :

python

Copier le code

import sys

from PyQt6.QtWidgets import QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout, QLabel

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets")

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Ajout des onglets

self.add\_tab("Onglet 1", "Contenu de l'onglet 1")

self.add\_tab("Onglet 2", "Contenu de l'onglet 2")

self.add\_tab("Onglet 3", "Contenu de l'onglet 3")

self.setCentralWidget(self.tab\_widget)

def add\_tab(self, title, content):

# Crée un nouveau widget pour l'onglet

tab = QWidget()

layout = QVBoxLayout()

layout.addWidget(QLabel(content))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, title)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, app\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python app\_tabs.py

**Ce que fait le code :**

* Il crée une fenêtre avec trois onglets.
* Chaque onglet affiche un simple message.

N'hésite pas à personnaliser le contenu de chaque onglet ou à ajouter plus d'onglets selon tes besoins ! Si tu as d'autres questions, fais-le moi savoir.

4o mini

**Vous avez dit :**

je veux quon puisse rajouter et supprimer des onglet comme on veut quad on veut

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici un exemple modifié où tu peux ajouter et supprimer des onglets dynamiquement dans une fenêtre PyQt6. Un bouton permet d'ajouter un nouvel onglet, et chaque onglet a un bouton pour le supprimer :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Création d'un bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.add\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addWidget(self.add\_tab\_button)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Compteur pour les noms des onglets

self.tab\_count = 0

def add\_tab(self):

self.tab\_count += 1

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet {self.tab\_count}"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.remove\_tab(self.tab\_widget.indexOf(tab)))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, f"Onglet {self.tab\_count}")

def remove\_tab(self, index):

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, dynamic\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python dynamic\_tabs.py

**Ce que fait le code :**

* Un bouton "Ajouter un Onglet" permet d'ajouter un nouvel onglet.
* Chaque nouvel onglet contient un bouton "Supprimer cet Onglet" qui permet de le supprimer.

Tu peux maintenant ajouter et supprimer des onglets comme bon te semble ! Si tu as d'autres questions ou des modifications à faire, n'hésite pas à demander.

4o mini

**Vous avez dit :**

je vexu pouvoir choisir leur nom et je veux un bouton de confiramtion avant de la creer ou la supprimer

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici un exemple où tu peux choisir le nom de l'onglet avant de l'ajouter, ainsi qu'un bouton de confirmation avant d'ajouter ou de supprimer un onglet :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QLineEdit, QMessageBox

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Création d'un bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

main\_layout.addWidget(self.add\_tab\_button)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Compteur pour les noms des onglets

self.tab\_count = 0

def confirm\_add\_tab(self):

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous ajouter l'onglet '{tab\_name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

self.add\_tab(tab\_name)

def add\_tab(self, name):

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet '{name}'"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, confirm\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python confirm\_tabs.py

**Ce que fait le code :**

* Un champ de texte permet de saisir le nom de l'onglet.
* Avant d'ajouter un onglet, une boîte de dialogue de confirmation s'affiche.
* Chaque onglet a un bouton pour le supprimer, et une confirmation est requise avant de procéder à la suppression.

Si tu as d'autres questions ou des modifications à faire, fais-le moi savoir !

4o mini

**Vous avez dit :**

on peut activer ou descativert loption ajouter et sup^rimer un onglet

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici un exemple modifié qui permet d'activer ou de désactiver les options pour ajouter et supprimer des onglets. Nous allons ajouter deux boutons : un pour activer/désactiver l'ajout d'onglets et un autre pour activer/désactiver la suppression d'onglets.

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

main\_layout.addWidget(self.add\_tab\_button)

main\_layout.addWidget(self.toggle\_add\_button)

main\_layout.addWidget(self.toggle\_remove\_button)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget() # Récupère le bouton de suppression

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous ajouter l'onglet '{tab\_name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

self.add\_tab(tab\_name)

def add\_tab(self, name):

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet '{name}'"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, toggle\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python toggle\_tabs.py

**Ce que fait le code :**

* Les boutons "Désactiver Ajouter" et "Désactiver Supprimer" permettent de désactiver les options d'ajout et de suppression d'onglets respectivement.
* L'état de chaque bouton est mis à jour en fonction de son statut.

Tu peux maintenant activer ou désactiver ces fonctionnalités selon tes besoins ! Si tu as d'autres questions ou des demandes supplémentaires, fais-le moi savoir !

4o mini

**Vous avez dit :**

je vexu que la grossuer de la fenetre soit 500x500

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Tu peux définir la taille de la fenêtre principale à 500x500 pixels en utilisant la méthode resize. Voici le code mis à jour :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(500, 500) # Définit la taille de la fenêtre à 500x500 pixels

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

main\_layout.addWidget(self.add\_tab\_button)

main\_layout.addWidget(self.toggle\_add\_button)

main\_layout.addWidget(self.toggle\_remove\_button)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget() # Récupère le bouton de suppression

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous ajouter l'onglet '{tab\_name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

self.add\_tab(tab\_name)

def add\_tab(self, name):

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet '{name}'"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, fixed\_size\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python fixed\_size\_tabs.py

**Ce que fait le code :**

* La taille de la fenêtre est définie sur 500x500 pixels.

Si tu as d'autres demandes ou questions, n'hésite pas à demander !

4o mini

**Vous avez dit :**

600x800

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici le code mis à jour pour que la taille de la fenêtre soit de 600x800 pixels :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800) # Définit la taille de la fenêtre à 600x800 pixels

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

main\_layout.addWidget(self.add\_tab\_button)

main\_layout.addWidget(self.toggle\_add\_button)

main\_layout.addWidget(self.toggle\_remove\_button)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget() # Récupère le bouton de suppression

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Vou

4o mini

**Vous avez dit :**

le scale grandit avec un rezise

**ChatGPT a dit :**

ChatGPT

Pour faire en sorte que la mise à l'échelle de l'interface s'adapte lors du redimensionnement de la fenêtre, tu peux utiliser un QVBoxLayout ou QGridLayout pour gérer la disposition des widgets. Cela permettra à la fenêtre de redimensionner les éléments de manière proportionnelle. Voici un exemple qui permet à la mise en page de s'adapter lorsque la taille de la fenêtre change :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800) # Définit la taille de la fenêtre à 600x800 pixels

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

main\_layout.addWidget(self.add\_tab\_button)

main\_layout.addWidget(self.toggle\_add\_button)

main\_layout.addWidget(self.toggle\_remove\_button)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget() # Récupère le bouton de suppression

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous ajouter l'onglet '{tab\_name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

self.add\_tab(tab\_name)

def add\_tab(self, name):

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet '{name}'"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Ce que fait le code :**

* La mise en page utilise un QVBoxLayout, ce qui permet aux widgets de s'adapter lorsque la fenêtre est redimensionnée.
* La taille initiale de la fenêtre est définie à 600x800 pixels.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, resizable\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python resizable\_tabs.py

Si tu as d'autres questions ou souhaites des modifications supplémentaires, fais-le moi savoir !

4o mini

**Vous avez dit :**

desoler je veux que quand on aggrandit la fenetre une mise a echelle est fait sur les composant de la fenetre

**ChatGPT a dit :**

ChatGPT

Pour obtenir un effet de mise à l'échelle sur les composants de la fenêtre lorsque celle-ci est agrandie, tu peux utiliser un QGridLayout et ajuster la taille des composants en utilisant des propriétés de mise à l'échelle. Voici un exemple où les éléments s'adaptent à la taille de la fenêtre :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox, QHBoxLayout

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800) # Définit la taille de la fenêtre à 600x800 pixels

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.add\_tab\_button)

button\_layout.addWidget(self.toggle\_add\_button)

button\_layout.addWidget(self.toggle\_remove\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget() # Récupère le bouton de suppression

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous ajouter l'onglet '{tab\_name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

self.add\_tab(tab\_name)

def add\_tab(self, name):

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet '{name}'"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Ce que fait le code :**

1. **Utilisation de Layouts** : En utilisant des QVBoxLayout et QHBoxLayout, les composants de la fenêtre s'ajustent lorsque la taille de la fenêtre change.
2. **Répartition des espaces** : Les boutons sont placés dans un QHBoxLayout, ce qui leur permet de s'étendre ou de se réduire en fonction de la taille de la fenêtre.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, scalable\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python scalable\_tabs.py

Cette mise en page permettra aux composants de s'ajuster et de se redimensionner en fonction de la taille de la fenêtre, créant ainsi un effet de mise à l'échelle. Si tu as d'autres questions ou des modifications à apporter, fais-le moi savoir !

4o mini

**Vous avez dit :**

faire un bouton quitter qui me permet de quitter la fenetre

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici comment ajouter un bouton "Quitter" qui ferme la fenêtre lorsque vous cliquez dessus. J'ai ajouté le bouton à la mise en page et connecté son signal au slot qui ferme la fenêtre.

Voici le code mis à jour :

python

Copier le code

import sys

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox, QHBoxLayout

)

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800) # Définit la taille de la fenêtre à 600x800 pixels

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close) # Ferme la fenêtre

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.add\_tab\_button)

button\_layout.addWidget(self.toggle\_add\_button)

button\_layout.addWidget(self.toggle\_remove\_button)

button\_layout.addWidget(self.quit\_button) # Ajoute le bouton Quitter

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget() # Récupère le bouton de suppression

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous ajouter l'onglet '{tab\_name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

self.add\_tab(tab\_name)

def add\_tab(self, name):

tab = QWidget()

layout = QVBoxLayout()

# Contenu de l'onglet

layout.addWidget(QLabel(f"Contenu de l'onglet '{name}'"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

* **Ajout d'un bouton "Quitter"** : Un nouveau bouton a été ajouté pour quitter l'application. Il appelle la méthode close() lorsqu'il est cliqué.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, quit\_button\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python quit\_button\_tabs.py

Tu peux maintenant utiliser le bouton "Quitter" pour fermer l'application. Si tu as d'autres questions ou modifications à apporter, fais-le moi savoir !

4o mini

**Vous avez dit :**

je veux une fonctionalite des onglet preset: qui est logger (admin/employe/boss) et lheure quil a ete connecter en dernier

**ChatGPT a dit :**

ChatGPT

Pour ajouter une fonctionnalité qui permet d'afficher les onglets de connexion des utilisateurs (avec des rôles tels qu'admin, employé, et boss) ainsi que l'heure de leur dernière connexion, nous allons modifier le code pour inclure ces fonctionnalités.

Voici un exemple complet avec cette fonctionnalité :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox, QHBoxLayout

)

class User:

def \_\_init\_\_(self, role):

self.role = role

self.last\_login = datetime.now()

def update\_last\_login(self):

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.add\_tab\_button)

button\_layout.addWidget(self.toggle\_add\_button)

button\_layout.addWidget(self.toggle\_remove\_button)

button\_layout.addWidget(self.quit\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

# Utilisateurs prédéfinis

self.users = {

"admin": User("Admin"),

"employe": User("Employé"),

"boss": User("Boss"),

}

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget()

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

# Choix de l'utilisateur

user\_choice, ok = QMessageBox.getItem(self, "Choisir un utilisateur",

"Sélectionnez l'utilisateur :",

["admin", "employe", "boss"], 0, False)

if ok and user\_choice in self.users:

self.users[user\_choice].update\_last\_login() # Met à jour l'heure de connexion

self.add\_tab(tab\_name, user\_choice)

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

user = self.users[user\_role]

layout.addWidget(QLabel(f"Onglet créé par : {user.role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {user.last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Classe User** : Ajout d'une classe User pour représenter un utilisateur avec un rôle et une heure de dernière connexion.
2. **Utilisateurs prédéfinis** : Création de trois utilisateurs prédéfinis : admin, employe, et boss.
3. **Sélection d'utilisateur** : Lors de l'ajout d'un onglet, une boîte de dialogue permet de choisir l'utilisateur. L'heure de dernière connexion est mise à jour et affichée dans l'onglet.
4. **Affichage des informations** : Chaque onglet montre le rôle de l'utilisateur et l'heure de sa dernière connexion.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, user\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python user\_tabs.py

Cette implémentation permet de gérer les onglets de connexion avec des rôles et de visualiser l'heure de dernière connexion de chaque utilisateur. Si tu as besoin de modifications supplémentaires, fais-le moi savoir !

4o mini

**Vous avez dit :**

Peut tu faire un classe séparé onglet qui sont des onglets deja existant comme ca a chaque fois que le programme est rouler, les onglets existants sont affichés

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Nous allons créer une classe séparée pour représenter des onglets prédéfinis. Ces onglets seront affichés chaque fois que le programme est exécuté. Voici comment faire cela :

1. **Classe Onglet** : Une nouvelle classe pour représenter chaque onglet avec un rôle utilisateur et l'heure de dernière connexion.
2. **Liste d'onglets prédéfinis** : Au démarrage de l'application, les onglets existants seront créés et affichés.

Voici le code mis à jour :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox, QHBoxLayout

)

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

def update\_last\_login(self):

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.add\_tab\_button)

button\_layout.addWidget(self.toggle\_add\_button)

button\_layout.addWidget(self.toggle\_remove\_button)

button\_layout.addWidget(self.quit\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("Onglet 1", "Admin"),

Onglet("Onglet 2", "Employé"),

Onglet("Onglet 3", "Boss"),

]

# Affiche les onglets existants

self.afficher\_onglets\_existants()

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget()

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

# Choix de l'utilisateur

user\_choice, ok = QMessageBox.getItem(self, "Choisir un utilisateur",

"Sélectionnez l'utilisateur :",

["Admin", "Employé", "Boss"], 0, False)

if ok and user\_choice in ["Admin", "Employé", "Boss"]:

# Crée un nouvel onglet

onglet = Onglet(tab\_name, user\_choice)

onglet.update\_last\_login() # Met à jour l'heure de connexion

self.onglets\_existants.append(onglet) # Ajoute l'onglet à la liste existante

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Classe Onglet** : Représente chaque onglet avec un nom, un rôle et une heure de dernière connexion.
2. **Liste d'onglets prédéfinis** : self.onglets\_existants contient des onglets déjà créés, qui sont affichés lors du démarrage de l'application.
3. **Affichage des onglets existants** : La méthode afficher\_onglets\_existants parcourt la liste des onglets existants et les affiche dans le QTabWidget.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, existing\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python existing\_tabs.py

Avec cette mise à jour, l'application affichera les onglets prédéfinis à chaque démarrage. Si tu souhaites d'autres modifications ou ajouts, fais-le moi savoir !

4o mini

**Vous avez dit :**

les nom de mes onglet pre set: admin bos employe

**ChatGPT a dit :**

ChatGPT

Pour mettre à jour les noms des onglets prédéfinis en "admin", "boss" et "employe", voici le code modifié :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox, QHBoxLayout

)

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

def update\_last\_login(self):

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Bouton pour activer/désactiver la suppression d'onglets

self.toggle\_remove\_button = QPushButton("Désactiver Supprimer")

self.toggle\_remove\_button.setCheckable(True)

self.toggle\_remove\_button.toggled.connect(self.toggle\_remove\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.add\_tab\_button)

button\_layout.addWidget(self.toggle\_add\_button)

button\_layout.addWidget(self.toggle\_remove\_button)

button\_layout.addWidget(self.quit\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

self.remove\_enabled = True

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Affiche les onglets existants

self.afficher\_onglets\_existants()

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def toggle\_remove\_tab(self, checked):

self.remove\_enabled = checked

for i in range(self.tab\_widget.count()):

tab\_widget = self.tab\_widget.widget(i)

remove\_button = tab\_widget.layout().itemAt(1).widget()

remove\_button.setEnabled(checked)

self.toggle\_remove\_button.setText("Activer Supprimer" if not checked else "Désactiver Supprimer")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

# Choix de l'utilisateur

user\_choice, ok = QMessageBox.getItem(self, "Choisir un utilisateur",

"Sélectionnez l'utilisateur :",

["Admin", "Employé", "Boss"], 0, False)

if ok and user\_choice in ["Admin", "Employé", "Boss"]:

# Crée un nouvel onglet

onglet = Onglet(tab\_name, user\_choice)

onglet.update\_last\_login() # Met à jour l'heure de connexion

self.onglets\_existants.append(onglet) # Ajoute l'onglet à la liste existante

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

# Bouton pour supprimer cet onglet

remove\_button = QPushButton("Supprimer cet Onglet")

remove\_button.clicked.connect(lambda: self.confirm\_remove\_tab(name))

layout.addWidget(remove\_button)

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

def confirm\_remove\_tab(self, name):

if not self.remove\_enabled:

QMessageBox.warning(self, "Erreur", "La suppression d'onglets est désactivée.")

return

reply = QMessageBox.question(self, 'Confirmation', f"Voulez-vous supprimer l'onglet '{name}' ?",

QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No)

if reply == QMessageBox.StandardButton.Yes:

index = self.tab\_widget.indexOf(self.tab\_widget.currentWidget())

if index >= 0:

self.tab\_widget.removeTab(index)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Noms des onglets prédéfinis** : Les noms des onglets sont désormais "admin", "boss" et "employe".

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, preset\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python preset\_tabs.py

Avec ces modifications, les onglets prédéfinis afficheront les noms que tu as spécifiés. Si tu souhaites d'autres changements ou ajouts, fais-le moi savoir !

4o mini

**Vous avez dit :**

enelve supprimer onglet

**ChatGPT a dit :**

ChatGPT

D'accord ! Pour enlever la fonctionnalité de suppression des onglets, nous allons retirer le bouton de suppression et la logique associée. Voici le code mis à jour :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QLineEdit, QMessageBox, QHBoxLayout

)

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

def update\_last\_login(self):

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Champ de texte pour le nom de l'onglet

self.tab\_name\_input = QLineEdit()

self.tab\_name\_input.setPlaceholderText("Nom de l'onglet")

# Bouton pour ajouter un onglet

self.add\_tab\_button = QPushButton("Ajouter un Onglet")

self.add\_tab\_button.clicked.connect(self.confirm\_add\_tab)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour activer/désactiver l'ajout d'onglets

self.toggle\_add\_button = QPushButton("Désactiver Ajouter")

self.toggle\_add\_button.setCheckable(True)

self.toggle\_add\_button.toggled.connect(self.toggle\_add\_tab)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_name\_input)

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.add\_tab\_button)

button\_layout.addWidget(self.toggle\_add\_button)

button\_layout.addWidget(self.quit\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Initialisation des états

self.add\_enabled = True

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Affiche les onglets existants

self.afficher\_onglets\_existants()

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def toggle\_add\_tab(self, checked):

self.add\_enabled = checked

self.add\_tab\_button.setEnabled(checked)

self.toggle\_add\_button.setText("Activer Ajouter" if not checked else "Désactiver Ajouter")

def confirm\_add\_tab(self):

if not self.add\_enabled:

QMessageBox.warning(self, "Erreur", "L'ajout d'onglets est désactivé.")

return

tab\_name = self.tab\_name\_input.text()

if not tab\_name:

QMessageBox.warning(self, "Erreur", "Veuillez entrer un nom d'onglet.")

return

# Choix de l'utilisateur

user\_choice, ok = QMessageBox.getItem(self, "Choisir un utilisateur",

"Sélectionnez l'utilisateur :",

["Admin", "Employé", "Boss"], 0, False)

if ok and user\_choice in ["Admin", "Employé", "Boss"]:

# Crée un nouvel onglet

onglet = Onglet(tab\_name, user\_choice)

onglet.update\_last\_login() # Met à jour l'heure de connexion

self.onglets\_existants.append(onglet) # Ajoute l'onglet à la liste existante

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

self.tab\_name\_input.clear() # Vide le champ de texte

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Suppression du bouton de suppression** : Le code pour le bouton de suppression d'onglet a été enlevé.
2. **Suppression de la logique de suppression** : Toutes les références à la suppression d'onglets ont été retirées, simplifiant ainsi le code.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, no\_remove\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python no\_remove\_tabs.py

Avec ces modifications, tu ne pourras plus supprimer d'onglets. Si tu as besoin d'autres changements, fais-le moi savoir !

4o mini

**Vous avez dit :**

enleve aussi ajouter onglet

**ChatGPT a dit :**

ChatGPT

D'accord ! Voici le code mis à jour, où la fonctionnalité d'ajout d'onglets a été complètement retirée. L'application affichera uniquement les onglets prédéfinis :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QMessageBox, QHBoxLayout

)

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour le bouton de quitter

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.quit\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Affiche les onglets existants

self.afficher\_onglets\_existants()

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Suppression de la fonctionnalité d'ajout d'onglets** : Les éléments pour ajouter des onglets ont été complètement retirés.
2. **Interface simplifiée** : L'interface ne propose plus de champ de texte pour entrer un nom d'onglet ni de bouton d'ajout.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, only\_preset\_tabs.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python only\_preset\_tabs.py

Avec ces modifications, l'application affichera uniquement les onglets prédéfinis sans possibilité d'ajouter ou de supprimer des onglets. Si tu as besoin d'autres ajustements, fais-le moi savoir !

4o mini

**Vous avez dit :**

avec le bouton quitter fait un bouton retour qui nous retourne sur le dernier onglet visite

**ChatGPT a dit :**

ChatGPT

Pour ajouter un bouton "Retour" qui permet de revenir au dernier onglet visité, nous allons stocker l'index du dernier onglet actif. Voici le code mis à jour :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout

)

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour retourner au dernier onglet

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

main\_layout.addLayout(button\_layout)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Stockage de l'index du dernier onglet** : La variable self.last\_index stocke l'index de l'onglet actif précédent.
2. **Signal currentChanged** : Ce signal est connecté à la méthode update\_last\_index pour mettre à jour l'index chaque fois que l'utilisateur change d'onglet.
3. **Bouton "Retour"** : Ce bouton permet de revenir au dernier onglet visité en utilisant la méthode go\_back.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_back\_button.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_back\_button.py

Avec ces modifications, tu pourras utiliser le bouton "Retour" pour revenir au dernier onglet visité. Si tu as besoin d'autres ajustements ou ajouts, fais-le moi savoir !

4o mini

**Vous avez dit :**

un bouton sign out qui nous deconnecter une secton qui affiche lheure (qui sauto refresh) une section qui affiche le nom de la personne connecter et un section qui affiche son niveau fde visibilite (boos/admin/employe)

**ChatGPT a dit :**

ChatGPT

Pour ajouter un bouton "Sign Out" (Déconnexion), une section pour afficher l'heure avec auto-refresh, ainsi qu'une section pour afficher le nom et le niveau de visibilité de l'utilisateur connecté, voici le code modifié :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QTimer

)

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour retourner au dernier onglet

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

# Bouton pour se déconnecter

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Admin", "admin") # Remplace par les informations réelles

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Bouton "Déconnexion"** : Ajouté un bouton pour déconnecter l'utilisateur.
2. **Mise à jour de l'heure** : Une section pour afficher l'heure actuelle qui se met à jour toutes les secondes.
3. **Informations utilisateur** : Une section pour afficher le nom de l'utilisateur connecté et son rôle (visibilité).
4. **Fonctionnalité de déconnexion** : Lorsque l'utilisateur se déconnecte, les informations de connexion sont réinitialisées et un message de confirmation est affiché.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_sign\_out.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_sign\_out.py

Avec ces modifications, l'application affichera l'heure actuelle, le nom de l'utilisateur connecté et son rôle, tout en permettant la déconnexion. Si tu souhaites d'autres modifications ou ajouts, fais-le moi savoir !

4o mini

**Vous avez dit :**

PS C:\Users\2163453> & C:/Python312/python.exe "x:/PYTHON JM/squelette.py" Traceback (most recent call last): File "x:\PYTHON JM\squelette.py", line 3, in <module> from PyQt6.QtWidgets import ( ImportError: cannot import name 'QTimer' from 'PyQt6.QtWidgets' (C:\Python312\Lib\site-packages\PyQt6\QtWidgets.pyd) PS C:\Users\2163453>

**ChatGPT a dit :**

ChatGPT

Il semble que le module QTimer soit manquant dans l'importation. En fait, QTimer doit être importé depuis PyQt6.QtCore. Voici la correction à apporter au code :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox

)

from PyQt6.QtCore import QTimer # Correction ici

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Bouton pour quitter

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

# Bouton pour retourner au dernier onglet

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

# Bouton pour se déconnecter

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Admin", "admin") # Remplace par les informations réelles

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Importation de QTimer** : Ajout de l'importation correcte de QTimer à partir de PyQt6.QtCore.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_sign\_out.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_sign\_out.py

Cela devrait résoudre le problème. Si tu rencontres d'autres erreurs ou as besoin de modifications supplémentaires, fais-le moi savoir !

4o mini

**Vous avez dit :**

je veux que les bouton et information a la meme hauteur

**ChatGPT a dit :**

ChatGPT

Pour que les boutons et les informations (comme l'heure et les détails de l'utilisateur) soient alignés à la même hauteur, nous pouvons utiliser un layout avec des propriétés d'alignement. Voici le code mis à jour pour s'assurer que tout est bien aligné :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox

)

from PyQt6.QtCore import QTimer

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter) # Centrer le texte

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Admin", "admin") # Remplace par les informations réelles

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter) # Centrer le texte

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Alignement centré** : Utilisation de setAlignment(Qt.AlignmentFlag.AlignCenter) pour centrer les labels d'heure et d'informations utilisateur, ce qui aide à garder l'interface propre.
2. **Mise en page uniforme** : La mise en page des boutons et des informations est maintenant alignée.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_sign\_out.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_sign\_out.py

Cela devrait donner un aspect plus uniforme et bien aligné à l'interface. Si tu souhaites d'autres modifications, fais-le moi savoir !

4o mini

**Vous avez dit :**

rajoute moi un bouton connectio n(qui demande un identifiant et un mot de passe)

**ChatGPT a dit :**

ChatGPT

Pour ajouter un bouton de connexion qui demande un identifiant et un mot de passe, nous allons créer une boîte de dialogue pour entrer ces informations. Voici le code mis à jour :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit

)

from PyQt6.QtCore import QTimer

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin": # Exemple d'identifiants

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss": # Exemple d'identifiants

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe": # Exemple d'identifiants

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Bouton de connexion** : Ajout d'un bouton "Connexion" qui ouvre une boîte de dialogue pour entrer l'identifiant et le mot de passe.
2. **Boîte de dialogue de connexion** : La méthode show\_login\_dialog crée une nouvelle boîte de dialogue avec des champs pour l'identifiant et le mot de passe.
3. **Vérification des identifiants** : La méthode login vérifie les identifiants et met à jour les informations utilisateur si elles sont correctes. Les identifiants sont définis comme exemple (admin, boss, employé).

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_login.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_login.py

Avec ces modifications, l'application dispose d'une fonctionnalité de connexion. Si tu souhaites d'autres modifications ou ajouts, fais-le moi savoir !

4o mini

**Vous avez dit :**

Traceback (most recent call last): File "x:\PYTHON JM\squelette.py", line 162, in <module> window = MainWindow() ^^^^^^^^^^^^ File "x:\PYTHON JM\squelette.py", line 54, in \_\_init\_\_ self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter) ^^ NameError: name 'Qt' is not defined PS C:\Users\2163453>

**ChatGPT a dit :**

ChatGPT

Il semble que l'importation de Qt soit manquante. Nous devons importer Qt à partir de PyQt6.QtCore. Voici le code corrigé :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt # Importation de Qt ajoutée

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin": # Exemple d'identifiants

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss": # Exemple d'identifiants

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe": # Exemple d'identifiants

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Importation de Qt** : Ajout de from PyQt6.QtCore import Qt pour permettre l'utilisation de Qt.AlignmentFlag.AlignCenter.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_login.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_login.py

Cela devrait résoudre le problème. Si tu as d'autres questions ou besoins, fais-le moi savoir !

4o mini

**Vous avez dit :**

je veux que chanque onglet a un niveau de visibilite et que pas tout les role peut le voir ex; boss voit tout mais employe vois quemploye

**ChatGPT a dit :**

ChatGPT

Pour implémenter un système de niveaux de visibilité sur les onglets, nous allons ajouter une logique qui détermine quels onglets chaque rôle peut voir. Voici comment procéder :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Onglets prédéfinis

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

Onglet("employe\_boss", "Employé (visibilité Boss)", "boss"),

]

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Admin":

return True # L'admin peut voir tous les onglets

elif self.current\_user\_role == "Boss":

return role in ["boss", "employe\_boss"] # Boss peut voir ses onglets et ceux des employés

elif self.current\_user\_role == "Employé":

return role == "employe" # Employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.tab\_widget.clear() # Efface les onglets existants avant de les afficher à nouveau

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Niveaux de visibilité des onglets** : Chaque onglet a un niveau de visibilité défini par son rôle. Par exemple, seuls les administrateurs peuvent voir tous les onglets, les bosses peuvent voir leurs propres onglets et ceux des employés, et les employés ne peuvent voir que leurs onglets.
2. **Logique de vérification d'accès** : Ajout de la méthode user\_role\_can\_access qui détermine si l'utilisateur peut voir un onglet en fonction de son rôle.
3. **Mise à jour dynamique des onglets** : Lorsqu'un utilisateur se connecte, la liste des onglets est mise à jour pour ne montrer que ceux auxquels il a accès.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Avec ces modifications, chaque rôle ne voit que les onglets auxquels il a accès. Si tu as d'autres demandes ou des ajustements à faire, fais-le moi savoir !

4o mini

**Vous avez dit :**

File "x:\PYTHON JM\squelette.py", line 178, in <module> window = MainWindow() ^^^^^^^^^^^^ File "x:\PYTHON JM\squelette.py", line 60, in \_\_init\_\_ self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "x:\PYTHON JM\squelette.py", line 130, in update\_user\_info self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "x:\PYTHON JM\squelette.py", line 88, in afficher\_onglets\_existants for onglet in self.onglets\_existants: ^^^^^^^^^^^^^^^^^^^^^^ AttributeError: 'MainWindow' object has no attribute 'onglets\_existants' PS C:\Users\2163453>

**ChatGPT a dit :**

ChatGPT

L'erreur indique que l'attribut onglets\_existants n'est pas accessible au moment où il est appelé. Cela peut être dû à un ordre d'initialisation incorrect. Assurons-nous que tous les attributs nécessaires sont définis avant leur utilisation. Voici le code corrigé :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = "Invité" # Valeur par défaut

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

Onglet("employe\_boss", "Employé (visibilité Boss)", "boss"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Admin":

return True # L'admin peut voir tous les onglets

elif self.current\_user\_role == "Boss":

return role in ["boss", "employe\_boss"] # Boss peut voir ses onglets et ceux des employés

elif self.current\_user\_role == "Employé":

return role == "employe" # Employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **Attributs d'état** : L'attribut current\_user\_role est initialisé dans le constructeur de MainWindow pour s'assurer qu'il est défini avant d'être utilisé.
2. **Effacement des onglets** : La méthode afficher\_onglets\_existants efface d'abord tous les onglets avant de les réafficher, ce qui garantit que seuls les onglets accessibles sont montrés.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Cela devrait résoudre le problème que tu as rencontré. Si tu as d'autres questions ou demandes, n'hésite pas à demander !

4o mini

**Vous avez dit :**

Traceback (most recent call last): File "x:\PYTHON JM\squelette.py", line 178, in <module> window = MainWindow() ^^^^^^^^^^^^ File "x:\PYTHON JM\squelette.py", line 28, in \_\_init\_\_ Onglet("employe\_boss", "Employé (visibilité Boss)", "boss"), ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ TypeError: Onglet.\_\_init\_\_() takes 3 positional arguments but 4 were given PS C:\Users\2163453>

**ChatGPT a dit :**

ChatGPT

L'erreur indique que la classe Onglet est définie avec deux paramètres, mais nous lui passons trois arguments. Cela signifie que la définition de la classe Onglet doit être modifiée pour accepter le niveau de visibilité.

Voici le code corrigé :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = "Invité" # Valeur par défaut

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

Onglet("employe\_boss", "Employé (visibilité Boss)"), # Correction ici

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Affiche les onglets existants

self.afficher\_onglets\_existants()

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role, onglet.last\_login)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Admin":

return True # L'admin peut voir tous les onglets

elif self.current\_user\_role == "Boss":

return role in ["boss", "employe\_boss"] # Boss peut voir ses onglets et ceux des employés

elif self.current\_user\_role == "Employé":

return role == "employe" # Employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role, last\_login):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {last\_login.strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements :**

1. **Correction de la classe Onglet** : J'ai retiré l'argument visibilité dans la définition de l'onglet pour ne conserver que name et role.
2. **Éléments de l'onglet** : Les onglets préétablis dans la liste onglets\_existants sont maintenant définis correctement.

**Instructions pour exécuter le code :**

1. Assure-toi d'avoir installé PyQt6 :

bash

Copier le code

pip install PyQt6

1. Copie le code ci-dessus dans un fichier Python (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Cela devrait résoudre l'erreur. Si tu as d'autres questions ou demandes, fais-le moi savoir !

4o mini

**Vous avez dit :**

mias la je me connecte en tant que boss mais je vois aucun onglert

**ChatGPT a dit :**

ChatGPT

Pour résoudre ce problème, assurons-nous que les onglets sont bien affichés en fonction du rôle de l'utilisateur. Voici quelques ajustements à apporter au code afin que chaque rôle puisse voir les onglets appropriés :

1. **Vérification des rôles** : Nous devons nous assurer que les onglets sont ajoutés correctement en fonction du rôle de l'utilisateur connecté.
2. **Mise à jour des onglets existants** : Les onglets doivent être affichés après la connexion en fonction du rôle.

Voici le code mis à jour :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = "Invité" # Valeur par défaut

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Admin":

return True # L'admin peut voir tous les onglets

elif self.current\_user\_role == "Boss":

return role == "Boss" # Boss peut voir son onglet

elif self.current\_user\_role == "Employé":

return role == "Employé" # Employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements apportés :**

1. **Rôle de l'utilisateur** : L'affichage des onglets a été simplifié pour s'assurer que chaque rôle ne peut voir que son onglet spécifique.
   * L'onglet "Admin" ne sera visible que pour l'admin.
   * L'onglet "Boss" ne sera visible que pour le boss.
   * L'onglet "Employé" ne sera visible que pour les employés.
2. **Affichage correct des onglets** : Assurez-vous que l'onglet correspondant à chaque rôle est affiché après connexion.

**Instructions :**

1. Installe PyQt6 si ce n'est pas déjà fait :

bash

Copier le code

pip install PyQt6

1. Copie le code mis à jour dans un fichier (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Cela devrait fonctionner comme prévu. Si tu rencontres d'autres problèmes, fais-le moi savoir !

4o mini

**Vous avez dit :**

je veux que quand je suis boss je vois TOUS les onglet admin voit seulement onglet employe et admin et employe voit juste employe

**ChatGPT a dit :**

ChatGPT

Pour que chaque rôle voit les onglets appropriés selon tes spécifications, nous devons modifier la fonction user\_role\_can\_access. Voici les règles que nous allons appliquer :

* **Boss** : voit tous les onglets (Admin, Boss, Employé).
* **Admin** : voit les onglets Admin et Employé.
* **Employé** : voit uniquement l'onglet Employé.

Voici le code mis à jour :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = "Invité" # Valeur par défaut

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True # Le boss peut voir tous les onglets

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"] # L'admin peut voir admin et employé

elif self.current\_user\_role == "Employé":

return role == "Employé" # L'employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements apportés :**

1. **Rôle du Boss** : Le Boss peut voir tous les onglets (Admin, Boss, Employé).
2. **Rôle de l'Admin** : L'Admin peut voir les onglets Admin et Employé.
3. **Rôle de l'Employé** : L'Employé ne voit que son onglet (Employé).

**Instructions :**

1. Installe PyQt6 si ce n'est pas déjà fait :

bash

Copier le code

pip install PyQt6

1. Copie le code mis à jour dans un fichier (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Cela devrait répondre à ta demande. Si tu as d'autres questions ou modifications, n'hésite pas à me le faire savoir !

4o mini

**Vous avez dit :**

je vuex que le bouton connexion soit la seulemnt si on est pas connecter et la meme pour deconnection je veux quil soit la seulement is on est connecter

**ChatGPT a dit :**

ChatGPT

Pour rendre le bouton de connexion visible uniquement lorsque l'utilisateur n'est pas connecté et le bouton de déconnexion visible uniquement lorsqu'il est connecté, nous devons ajouter quelques vérifications dans notre code. Voici comment procéder :

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = "Invité" # Valeur par défaut

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

# Initialiser l'état des boutons

self.update\_button\_visibility()

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True # Le boss peut voir tous les onglets

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"] # L'admin peut voir admin et employé

elif self.current\_user\_role == "Employé":

return role == "Employé" # L'employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

self.update\_button\_visibility() # Met à jour la visibilité des boutons

def update\_button\_visibility(self):

if self.current\_user\_role == "Invité":

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else:

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements apportés :**

1. **Mise à jour de la visibilité des boutons** :
   * La méthode update\_button\_visibility gère l'affichage des boutons de connexion et de déconnexion en fonction du rôle actuel de l'utilisateur.
   * Si l'utilisateur est "Invité", le bouton de connexion est visible et le bouton de déconnexion est masqué.
   * Si l'utilisateur est connecté (tout rôle sauf "Invité"), le bouton de déconnexion est visible et le bouton de connexion est masqué.

**Instructions :**

1. Installe PyQt6 si ce n'est pas déjà fait :

bash

Copier le code

pip install PyQt6

1. Copie le code mis à jour dans un fichier (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Cela devrait répondre à ta demande. Si tu as d'autres questions ou modifications, fais-le moi savoir !

4o mini

**Vous avez dit :**

Contenu supprimé

Ce contenu viole peut-être nos [politiques d'utilisation](https://openai.com/policies/usage-policies).

**ChatGPT a dit :**

ChatGPT

Je suis désolé pour la confusion. Voici comment nous pouvons corriger le code pour que le bouton de connexion soit visible uniquement lorsque l'utilisateur n'est pas connecté et que le bouton de déconnexion soit visible uniquement lorsqu'il l'est.

**Code corrigé :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = "Invité" # Valeur par défaut

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

# Initialiser l'état des boutons

self.update\_button\_visibility()

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True # Le boss peut voir tous les onglets

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"] # L'admin peut voir admin et employé

elif self.current\_user\_role == "Employé":

return role == "Employé" # L'employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

self.update\_button\_visibility() # Met à jour la visibilité des boutons

def update\_button\_visibility(self):

if self.current\_user\_role == "Invité":

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else:

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", "Aucun") # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements apportés :**

* Les boutons de connexion et de déconnexion sont maintenant gérés correctement. Le bouton de connexion n'apparaît que lorsque l'utilisateur est "Invité", et le bouton de déconnexion n'apparaît que lorsqu'un utilisateur est connecté.

**Instructions :**

1. Assure-toi d'avoir PyQt6 installé :

bash

Copier le code

pip install PyQt6

1. Copie ce code dans un fichier (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Si tu as d'autres modifications ou questions, fais-le moi savoir !

4o mini

**Vous avez dit :**

le bouton connexion soit disponible lors ce quaucun role est conneceter (employe/boos/admin)

**ChatGPT a dit :**

ChatGPT

D'accord ! Voici comment modifier le code pour que le bouton de connexion soit disponible uniquement lorsque **aucun rôle** (Employé, Boss, Admin) n'est connecté.

**Code mis à jour :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = None # Valeur par défaut pour aucun utilisateur connecté

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", "Aucun") # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

# Initialiser l'état des boutons

self.update\_button\_visibility()

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True # Le boss peut voir tous les onglets

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"] # L'admin peut voir admin et employé

elif self.current\_user\_role == "Employé":

return role == "Employé" # L'employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

self.update\_button\_visibility() # Met à jour la visibilité des boutons

def update\_button\_visibility(self):

if self.current\_user\_role is None: # Aucun utilisateur connecté

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else: # Un utilisateur est connecté

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", None) # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Modifications apportées :**

1. **État du rôle utilisateur** :
   * J'ai initialisé self.current\_user\_role à None pour indiquer qu'aucun utilisateur n'est connecté.
   * Les boutons de connexion et de déconnexion sont affichés correctement en fonction de l'état de connexion.
2. **Visibilité des boutons** :
   * Le bouton de connexion est visible uniquement lorsque self.current\_user\_role est None.

**Instructions :**

1. Assure-toi que PyQt6 est installé :

bash

Copier le code

pip install PyQt6

1. Copie ce code dans un fichier (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Si tu as d'autres questions ou souhaites encore d'autres modifications, n'hésite pas à demander !

4o mini

**Vous avez dit :**

premier ouverture a la base on est deconncecter

**ChatGPT a dit :**

ChatGPT

Pour s'assurer que l'application démarre toujours en tant qu'utilisateur déconnecté, même lors de la première ouverture, il est essentiel de s'assurer que la variable d'état de connexion est bien initialisée. Voici le code révisé pour garantir que l'utilisateur commence toujours déconnecté :

**Code final :**

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog

)

from PyQt6.QtCore import QTimer, Qt

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

# Attributs pour stocker l'état

self.current\_user\_role = None # Valeur par défaut pour aucun utilisateur connecté

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

# Création d'un widget pour les onglets

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

# Boutons

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

# Mise en page pour les boutons

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

# Mise en page principale

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

# Affichage de l'heure

self.time\_label = QLabel()

self.update\_time() # Mise à jour initiale

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

# Section d'informations utilisateur

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", None) # Valeurs par défaut

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

# Index du dernier onglet actif

self.last\_index = 0

# Timer pour actualiser l'heure

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000) # Actualise toutes les secondes

# Initialiser l'état des boutons

self.update\_button\_visibility()

def afficher\_onglets\_existants(self):

self.tab\_widget.clear() # Efface les onglets existants avant d'afficher à nouveau

for onglet in self.onglets\_existants:

# Affiche l'onglet seulement si le rôle de l'utilisateur a accès

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True # Le boss peut voir tous les onglets

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"] # L'admin peut voir admin et employé

elif self.current\_user\_role == "Employé":

return role == "Employé" # L'employé ne voit que son onglet

return False

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

# Affiche l'utilisateur et l'heure de dernière connexion

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

# Ajoute l'onglet au QTabWidget

self.tab\_widget.addTab(tab, name)

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role # Met à jour le rôle de l'utilisateur connecté

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})" if role else "Connecté : Invité")

self.afficher\_onglets\_existants() # Affiche les onglets en fonction du rôle

self.update\_button\_visibility() # Met à jour la visibilité des boutons

def update\_button\_visibility(self):

if self.current\_user\_role is None: # Aucun utilisateur connecté

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else: # Un utilisateur est connecté

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", None) # Réinitialiser les infos utilisateur

self.tab\_widget.setCurrentIndex(0) # Optionnel : revenir au premier onglet

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

# Vérification des identifiants (remplace par une logique réelle)

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur", "Identifiant ou mot de passe incorrect.")

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Points Clés :**

* **État Initial** : L'application démarre avec self.current\_user\_role défini à None, indiquant que l'utilisateur est déconnecté. Le message de connexion affiche "Connecté : Invité".
* **Visibilité des Boutons** : Les boutons de connexion et de déconnexion sont correctement affichés ou masqués en fonction de l'état de connexion.

**Instructions :**

1. Assure-toi d'avoir PyQt6 installé :

bash

Copier le code

pip install PyQt6

1. Copie ce code dans un fichier (par exemple, tabs\_with\_visibility.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python tabs\_with\_visibility.py

Si tu as d'autres ajustements ou questions, n'hésite pas à demander !

4o mini

**Vous avez dit :**

nouvel fichier: cest une classe qui va nous permettre de creer des graphique et de les afficher lors ce l'on appel

**ChatGPT a dit :**

ChatGPT

Pour créer une classe en Python qui permet de générer et d'afficher des graphiques, nous allons utiliser la bibliothèque matplotlib, qui est très populaire pour la création de graphiques en 2D. Voici comment procéder :

**Installation**

Si tu n'as pas encore installé matplotlib, tu peux le faire avec la commande suivante :

bash

Copier le code

pip install matplotlib

**Classe pour créer des graphiques**

Voici une classe simple qui utilise matplotlib pour créer et afficher des graphiques :

python

Copier le code

import matplotlib.pyplot as plt

import numpy as np

class Graphique:

def \_\_init\_\_(self):

pass

def tracer\_ligne(self, x, y, titre='Graphique Linéaire', xlabel='X', ylabel='Y'):

plt.figure(figsize=(10, 5))

plt.plot(x, y, marker='o')

plt.title(titre)

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.grid(True)

plt.show()

def tracer\_barres(self, categories, valeurs, titre='Graphique à Barres', xlabel='Catégories', ylabel='Valeurs'):

plt.figure(figsize=(10, 5))

plt.bar(categories, valeurs, color='skyblue')

plt.title(titre)

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.grid(axis='y')

plt.show()

def tracer\_histogramme(self, donnees, titre='Histogramme', xlabel='Valeurs', ylabel='Fréquence'):

plt.figure(figsize=(10, 5))

plt.hist(donnees, bins=10, color='lightgreen', edgecolor='black')

plt.title(titre)

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.grid(axis='y')

plt.show()

# Exemple d'utilisation

if \_\_name\_\_ == '\_\_main\_\_':

graphique = Graphique()

# Exemple pour un graphique linéaire

x = np.linspace(0, 10, 100)

y = np.sin(x)

graphique.tracer\_ligne(x, y, titre='Graphique de Sinus')

# Exemple pour un graphique à barres

categories = ['A', 'B', 'C', 'D']

valeurs = [10, 20, 15, 25]

graphique.tracer\_barres(categories, valeurs, titre='Graphique à Barres')

# Exemple pour un histogramme

donnees = np.random.randn(1000)

graphique.tracer\_histogramme(donnees, titre='Histogramme des Données Aléatoires')

**Explications :**

1. **Importations** :
   * matplotlib.pyplot pour tracer les graphiques.
   * numpy pour générer des données numériques.
2. **Classe Graphique** :
   * **Méthode tracer\_ligne** : Crée un graphique linéaire.
   * **Méthode tracer\_barres** : Crée un graphique à barres.
   * **Méthode tracer\_histogramme** : Crée un histogramme.
3. **Exemple d'utilisation** :
   * À la fin du fichier, des exemples d'appels à chaque méthode sont donnés pour démontrer comment utiliser la classe.

**Instructions :**

1. Copie ce code dans un fichier (par exemple, graphique.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python graphique.py

Cela affichera les graphiques dans des fenêtres séparées. Si tu as besoin de fonctionnalités supplémentaires ou d'autres types de graphiques, fais-le moi savoir !

4o mini

**Vous avez dit :**

on pourra choisir le type graphique (le nom, les nom des axe, les valeur)

**ChatGPT a dit :**

ChatGPT

Bien sûr ! Voici une version modifiée de la classe Graphique qui te permet de choisir le type de graphique (linéaire, à barres ou histogramme), ainsi que d'entrer le titre, les noms des axes et les valeurs souhaitées.

**Classe Graphique améliorée**

python

Copier le code

import matplotlib.pyplot as plt

import numpy as np

class Graphique:

def \_\_init\_\_(self):

pass

def tracer\_graphique(self, type\_graphique, titre, xlabel, ylabel, valeurs, categories=None):

plt.figure(figsize=(10, 5))

if type\_graphique == 'ligne':

x = np.arange(len(valeurs)) # Utilisation des indices si aucune valeur X n'est fournie

plt.plot(x, valeurs, marker='o')

elif type\_graphique == 'barres':

if categories is None:

raise ValueError("Pour un graphique à barres, 'categories' doit être fourni.")

plt.bar(categories, valeurs, color='skyblue')

elif type\_graphique == 'histogramme':

plt.hist(valeurs, bins=10, color='lightgreen', edgecolor='black')

else:

raise ValueError("Type de graphique non reconnu. Utilisez 'ligne', 'barres' ou 'histogramme'.")

plt.title(titre)

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.grid(True)

plt.show()

# Exemple d'utilisation

if \_\_name\_\_ == '\_\_main\_\_':

graphique = Graphique()

# Graphique linéaire

valeurs\_ligne = np.sin(np.linspace(0, 10, 100))

graphique.tracer\_graphique(

type\_graphique='ligne',

titre='Graphique de Sinus',

xlabel='Index',

ylabel='Valeur',

valeurs=valeurs\_ligne

)

# Graphique à barres

categories = ['A', 'B', 'C', 'D']

valeurs\_barres = [10, 20, 15, 25]

graphique.tracer\_graphique(

type\_graphique='barres',

titre='Graphique à Barres',

xlabel='Catégories',

ylabel='Valeurs',

valeurs=valeurs\_barres,

categories=categories

)

# Histogramme

donnees\_histogramme = np.random.randn(1000)

graphique.tracer\_graphique(

type\_graphique='histogramme',

titre='Histogramme des Données Aléatoires',

xlabel='Valeurs',

ylabel='Fréquence',

valeurs=donnees\_histogramme

)

**Explications des modifications :**

1. **Méthode tracer\_graphique** :
   * Prend en paramètre le type de graphique (type\_graphique), le titre (titre), les noms des axes (xlabel, ylabel), les valeurs (valeurs) et les catégories (uniquement pour les graphiques à barres).
   * Utilise une structure conditionnelle pour déterminer quel type de graphique tracer.
2. **Exemples d'utilisation** :
   * Montre comment appeler tracer\_graphique pour créer différents types de graphiques en utilisant des valeurs et des catégories personnalisées.

**Instructions :**

1. Copie ce code dans un fichier (par exemple, graphique\_personnalise.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python graphique\_personnalise.py

Cette version te permettra de choisir le type de graphique, le titre et les noms des axes, ainsi que les valeurs à afficher. Si tu souhaites d'autres fonctionnalités ou modifications, n'hésite pas à demander !

4o mini

**Vous avez dit :**

je veux dasn longlet boss un bouton qui me permet dafficher un graphique selon le type que je veux et les valeur ((le nom, les nom des axe, les valeur)

**ChatGPT a dit :**

ChatGPT

Pour ajouter un bouton dans l'onglet "Boss" qui permet d'afficher un graphique selon le type souhaité et d'entrer les valeurs (nom, noms des axes, valeurs), nous allons intégrer la classe Graphique dans l'application précédente. Voici comment procéder :

**Code Complet avec le Graphique dans l'Onglet Boss**

Voici le code modifié :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog, QComboBox

)

from PyQt6.QtCore import QTimer, Qt

import matplotlib.pyplot as plt

import numpy as np

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class Graphique:

def \_\_init\_\_(self):

pass

def tracer\_graphique(self, type\_graphique, titre, xlabel, ylabel, valeurs, categories=None):

plt.figure(figsize=(10, 5))

if type\_graphique == 'ligne':

x = np.arange(len(valeurs)) # Utilisation des indices si aucune valeur X n'est fournie

plt.plot(x, valeurs, marker='o')

elif type\_graphique == 'barres':

if categories is None:

raise ValueError("Pour un graphique à barres, 'categories' doit être fourni.")

plt.bar(categories, valeurs, color='skyblue')

elif type\_graphique == 'histogramme':

plt.hist(valeurs, bins=10, color='lightgreen', edgecolor='black')

else:

raise ValueError("Type de graphique non reconnu. Utilisez 'ligne', 'barres' ou 'histogramme'.")

plt.title(titre)

plt.xlabel(xlabel)

plt.ylabel(ylabel)

plt.grid(True)

plt.show()

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

self.current\_user\_role = None

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

self.time\_label = QLabel()

self.update\_time()

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", None)

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

self.last\_index = 0

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000)

self.update\_button\_visibility()

# Création d'onglets

self.create\_tabs()

def create\_tabs(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role)

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

if user\_role == "Boss":

# Ajouter bouton pour afficher un graphique

self.graphique\_button = QPushButton("Afficher Graphique")

self.graphique\_button.clicked.connect(self.show\_graphique\_dialog)

layout.addWidget(self.graphique\_button)

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

self.tab\_widget.addTab(tab, name)

def show\_graphique\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Créer un Graphique")

layout = QVBoxLayout()

# Sélecteur de type de graphique

type\_graphique\_label = QLabel("Type de Graphique:")

self.type\_graphique\_combo = QComboBox()

self.type\_graphique\_combo.addItems(['ligne', 'barres', 'histogramme'])

titre\_label = QLabel("Titre:")

self.titre\_input = QLineEdit()

xlabel\_label = QLabel("Nom de l'axe X:")

self.xlabel\_input = QLineEdit()

ylabel\_label = QLabel("Nom de l'axe Y:")

self.ylabel\_input = QLineEdit()

valeurs\_label = QLabel("Valeurs (séparées par des virgules):")

self.valeurs\_input = QLineEdit()

categories\_label = QLabel("Catégories (pour barres, séparées par des virgules):")

self.categories\_input = QLineEdit()

create\_button = QPushButton("Créer Graphique")

create\_button.clicked.connect(lambda: self.create\_graphique(dialog))

layout.addWidget(type\_graphique\_label)

layout.addWidget(self.type\_graphique\_combo)

layout.addWidget(titre\_label)

layout.addWidget(self.titre\_input)

layout.addWidget(xlabel\_label)

layout.addWidget(self.xlabel\_input)

layout.addWidget(ylabel\_label)

layout.addWidget(self.ylabel\_input)

layout.addWidget(valeurs\_label)

layout.addWidget(self.valeurs\_input)

layout.addWidget(categories\_label)

layout.addWidget(self.categories\_input)

layout.addWidget(create\_button)

dialog.setLayout(layout)

dialog.exec()

def create\_graphique(self, dialog):

type\_graphique = self.type\_graphique\_combo.currentText()

titre = self.titre\_input.text()

xlabel = self.xlabel\_input.text()

ylabel = self.ylabel\_input.text()

valeurs = list(map(float, self.valeurs\_input.text().split(',')))

categories = self.categories\_input.text().split(',') if type\_graphique == 'barres' else None

graphique = Graphique()

graphique.tracer\_graphique(type\_graphique, titre, xlabel, ylabel, valeurs, categories)

dialog.accept()

def afficher\_onglets\_existants(self):

self.tab\_widget.clear()

for onglet in self.onglets\_existants:

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"]

elif self.current\_user\_role == "Employé":

return role == "Employé"

return False

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})" if role else "Connecté : Invité")

self.afficher\_onglets\_existants()

self.update\_button\_visibility()

def update\_button\_visibility(self):

if self.current\_user\_role is None:

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else:

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", None)

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur de connexion", "Identifiant ou mot de passe incorrect.")

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Explications des ajouts :**

1. **Classe Graphique** : Reste inchangée, permet de tracer différents types de graphiques.
2. **Ajout du bouton dans l'onglet Boss** :
   * Dans la méthode add\_tab, un bouton "Afficher Graphique" est ajouté à l'onglet Boss, qui ouvre une boîte de dialogue pour créer un graphique.
3. **Boîte de dialogue pour le graphique** :
   * La méthode show\_graphique\_dialog crée une boîte de dialogue avec des champs pour entrer le type de graphique, le titre, les noms des axes et les valeurs.
4. **Création du graphique** :
   * La méthode create\_graphique prend les valeurs entrées par l'utilisateur, les convertit en format approprié, et appelle la méthode de traçage de la classe Graphique.
5. **Visibilité des boutons** :
   * Le bouton de connexion est affiché uniquement lorsque l'utilisateur n'est pas connecté, et le bouton de déconnexion est affiché uniquement lorsque l'utilisateur est connecté.

**Instructions :**

1. Copie ce code dans un fichier (par exemple, app\_graphique.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python app\_graphique.py

Tu pourras maintenant te connecter en tant que Boss et utiliser le bouton pour créer des graphiques selon tes spécifications. Si tu as d'autres demandes ou ajustements, n'hésite pas !

4o mini

**Vous avez dit :**

le graphique doit safficher dans longlet boss

**ChatGPT a dit :**

ChatGPT

Pour afficher le graphique directement dans l'onglet "Boss" au lieu de l'ouvrir dans une nouvelle fenêtre, nous pouvons utiliser un widget de matplotlib intégré dans le QWidget. Voici comment modifier le code pour intégrer le graphique dans l'onglet "Boss".

**Code Modifié pour Afficher le Graphique dans l'Onglet Boss**

Voici le code avec les modifications nécessaires :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog, QComboBox

)

from PyQt6.QtCore import QTimer, Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.figure import Figure

import numpy as np

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class Graphique:

def \_\_init\_\_(self):

pass

def tracer\_graphique(self, type\_graphique, valeurs, categories=None):

fig = Figure(figsize=(10, 5))

canvas = FigureCanvas(fig)

ax = fig.add\_subplot(111)

if type\_graphique == 'ligne':

x = np.arange(len(valeurs))

ax.plot(x, valeurs, marker='o')

elif type\_graphique == 'barres':

if categories is None:

raise ValueError("Pour un graphique à barres, 'categories' doit être fourni.")

ax.bar(categories, valeurs, color='skyblue')

elif type\_graphique == 'histogramme':

ax.hist(valeurs, bins=10, color='lightgreen', edgecolor='black')

else:

raise ValueError("Type de graphique non reconnu.")

ax.set\_title('Graphique')

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.grid(True)

return canvas

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

self.current\_user\_role = None

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

self.time\_label = QLabel()

self.update\_time()

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", None)

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

self.last\_index = 0

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000)

self.update\_button\_visibility()

# Création d'onglets

self.create\_tabs()

def create\_tabs(self):

for onglet in self.onglets\_existants:

self.add\_tab(onglet.name, onglet.role)

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

if user\_role == "Boss":

# Ajouter bouton pour afficher un graphique

self.graphique\_button = QPushButton("Afficher Graphique")

self.graphique\_button.clicked.connect(self.show\_graphique\_dialog)

layout.addWidget(self.graphique\_button)

self.graphique\_canvas = None # Pour stocker le canvas du graphique

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

self.tab\_widget.addTab(tab, name)

self.tab\_widget.setTabText(self.tab\_widget.indexOf(tab), name)

def show\_graphique\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Créer un Graphique")

layout = QVBoxLayout()

# Sélecteur de type de graphique

type\_graphique\_label = QLabel("Type de Graphique:")

self.type\_graphique\_combo = QComboBox()

self.type\_graphique\_combo.addItems(['ligne', 'barres', 'histogramme'])

titre\_label = QLabel("Titre:")

self.titre\_input = QLineEdit()

xlabel\_label = QLabel("Nom de l'axe X:")

self.xlabel\_input = QLineEdit()

ylabel\_label = QLabel("Nom de l'axe Y:")

self.ylabel\_input = QLineEdit()

valeurs\_label = QLabel("Valeurs (séparées par des virgules):")

self.valeurs\_input = QLineEdit()

categories\_label = QLabel("Catégories (pour barres, séparées par des virgules):")

self.categories\_input = QLineEdit()

create\_button = QPushButton("Créer Graphique")

create\_button.clicked.connect(lambda: self.create\_graphique(dialog))

layout.addWidget(type\_graphique\_label)

layout.addWidget(self.type\_graphique\_combo)

layout.addWidget(titre\_label)

layout.addWidget(self.titre\_input)

layout.addWidget(xlabel\_label)

layout.addWidget(self.xlabel\_input)

layout.addWidget(ylabel\_label)

layout.addWidget(self.ylabel\_input)

layout.addWidget(valeurs\_label)

layout.addWidget(self.valeurs\_input)

layout.addWidget(categories\_label)

layout.addWidget(self.categories\_input)

layout.addWidget(create\_button)

dialog.setLayout(layout)

dialog.exec()

def create\_graphique(self, dialog):

type\_graphique = self.type\_graphique\_combo.currentText()

valeurs = list(map(float, self.valeurs\_input.text().split(',')))

categories = self.categories\_input.text().split(',') if type\_graphique == 'barres' else None

# Trouver l'onglet Boss

boss\_tab\_index = self.tab\_widget.indexOf(self.tab\_widget.widget(1)) # Assumer que "Boss" est à l'index 1

if boss\_tab\_index != -1:

boss\_tab = self.tab\_widget.widget(boss\_tab\_index)

# Si un graphique existe déjà, le supprimer

if self.graphique\_canvas is not None:

self.graphique\_canvas.deleteLater()

graphique = Graphique()

self.graphique\_canvas = graphique.tracer\_graphique(type\_graphique, valeurs, categories)

boss\_tab.layout().addWidget(self.graphique\_canvas)

dialog.accept()

def afficher\_onglets\_existants(self):

self.tab\_widget.clear()

for onglet in self.onglets\_existants:

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"]

elif self.current\_user\_role == "Employé":

return role == "Employé"

return False

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})" if role else "Connecté : Invité")

self.afficher\_onglets\_existants()

self.update\_button\_visibility()

def update\_button\_visibility(self):

if self.current\_user\_role is None:

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else:

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", None)

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur de connexion", "Identifiant ou mot de passe incorrect.")

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements Apportés :**

1. **Intégration de Matplotlib dans l'Onglet Boss** :
   * Utilisation de FigureCanvas pour afficher le graphique directement dans l'onglet.
   * Le graphique est créé et ajouté à l'onglet "Boss" lorsqu'on clique sur le bouton "Afficher Graphique".
2. **Stockage du Canvas** :
   * Le canvas du graphique est stocké dans self.graphique\_canvas pour pouvoir le supprimer et éviter les doublons lors de l'affichage de nouveaux graphiques.
3. **Simplification** :
   * La logique pour tracer le graphique et l'afficher dans l'onglet est maintenant centralisée dans la méthode create\_graphique.

**Instructions :**

1. Copie le code ci-dessus dans un fichier (par exemple, app\_graphique\_dans\_onglet.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python app\_graphique\_dans\_onglet.py

Tu devrais maintenant voir le graphique s'afficher directement dans l'onglet "Boss" après avoir entré les valeurs nécessaires. Si tu as d'autres modifications ou besoins, fais-le moi savoir !

4o mini

**Vous avez dit :**

Les onglets sont visible même si on est connecter avec le mauvais utilisateur

**ChatGPT a dit :**

ChatGPT

Pour s'assurer que les onglets ne sont visibles que par les utilisateurs ayant le rôle approprié, nous devons modifier la logique qui affiche les onglets en fonction du rôle de l'utilisateur connecté. Voici comment procéder :

**Modifications à Apporter**

1. **Vérification des Rôles lors de l'Affichage des Onglets** : Nous allons ajuster la méthode afficher\_onglets\_existants pour qu'elle ne crée et n'affiche des onglets que si l'utilisateur est correctement connecté.
2. **Mise à jour des Onglets après la Connexion** : Nous appellerons afficher\_onglets\_existants uniquement après qu'un utilisateur valide se soit connecté.

**Code Mis à Jour**

Voici le code mis à jour avec ces ajustements :

python

Copier le code

import sys

from datetime import datetime

from PyQt6.QtWidgets import (

QApplication, QMainWindow, QTabWidget, QWidget, QVBoxLayout,

QPushButton, QLabel, QHBoxLayout, QMessageBox, QLineEdit, QDialog, QComboBox

)

from PyQt6.QtCore import QTimer, Qt

from matplotlib.backends.backend\_qt5agg import FigureCanvasQTAgg as FigureCanvas

from matplotlib.figure import Figure

import numpy as np

class Onglet:

def \_\_init\_\_(self, name, role):

self.name = name

self.role = role

self.last\_login = datetime.now()

class Graphique:

def \_\_init\_\_(self):

pass

def tracer\_graphique(self, type\_graphique, valeurs, categories=None):

fig = Figure(figsize=(10, 5))

canvas = FigureCanvas(fig)

ax = fig.add\_subplot(111)

if type\_graphique == 'ligne':

x = np.arange(len(valeurs))

ax.plot(x, valeurs, marker='o')

elif type\_graphique == 'barres':

if categories is None:

raise ValueError("Pour un graphique à barres, 'categories' doit être fourni.")

ax.bar(categories, valeurs, color='skyblue')

elif type\_graphique == 'histogramme':

ax.hist(valeurs, bins=10, color='lightgreen', edgecolor='black')

else:

raise ValueError("Type de graphique non reconnu.")

ax.set\_title('Graphique')

ax.set\_xlabel('X')

ax.set\_ylabel('Y')

ax.grid(True)

return canvas

class MainWindow(QMainWindow):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.setWindowTitle("Fenêtre avec Onglets Dynamiques")

self.resize(600, 800)

self.current\_user\_role = None

self.onglets\_existants = [

Onglet("admin", "Admin"),

Onglet("boss", "Boss"),

Onglet("employe", "Employé"),

]

self.tab\_widget = QTabWidget()

self.tab\_widget.currentChanged.connect(self.update\_last\_index)

self.quit\_button = QPushButton("Quitter")

self.quit\_button.clicked.connect(self.close)

self.back\_button = QPushButton("Retour")

self.back\_button.clicked.connect(self.go\_back)

self.sign\_out\_button = QPushButton("Déconnexion")

self.sign\_out\_button.clicked.connect(self.sign\_out)

self.connect\_button = QPushButton("Connexion")

self.connect\_button.clicked.connect(self.show\_login\_dialog)

button\_layout = QHBoxLayout()

button\_layout.addWidget(self.back\_button)

button\_layout.addWidget(self.quit\_button)

button\_layout.addWidget(self.sign\_out\_button)

button\_layout.addWidget(self.connect\_button)

main\_layout = QVBoxLayout()

main\_layout.addWidget(self.tab\_widget)

main\_layout.addLayout(button\_layout)

self.time\_label = QLabel()

self.update\_time()

self.time\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.time\_label)

self.user\_info\_label = QLabel()

self.update\_user\_info("Invité", None)

self.user\_info\_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

main\_layout.addWidget(self.user\_info\_label)

container = QWidget()

container.setLayout(main\_layout)

self.setCentralWidget(container)

self.last\_index = 0

self.timer = QTimer(self)

self.timer.timeout.connect(self.update\_time)

self.timer.start(1000)

self.update\_button\_visibility()

# Création d'onglets

self.create\_tabs()

def create\_tabs(self):

for onglet in self.onglets\_existants:

if self.user\_role\_can\_access(onglet.role):

self.add\_tab(onglet.name, onglet.role)

def add\_tab(self, name, user\_role):

tab = QWidget()

layout = QVBoxLayout()

if user\_role == "Boss":

self.graphique\_button = QPushButton("Afficher Graphique")

self.graphique\_button.clicked.connect(self.show\_graphique\_dialog)

layout.addWidget(self.graphique\_button)

layout.addWidget(QLabel(f"Onglet créé par : {user\_role}"))

layout.addWidget(QLabel(f"Heure de dernière connexion : {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"))

tab.setLayout(layout)

self.tab\_widget.addTab(tab, name)

self.tab\_widget.setTabText(self.tab\_widget.indexOf(tab), name)

def show\_graphique\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Créer un Graphique")

layout = QVBoxLayout()

type\_graphique\_label = QLabel("Type de Graphique:")

self.type\_graphique\_combo = QComboBox()

self.type\_graphique\_combo.addItems(['ligne', 'barres', 'histogramme'])

titre\_label = QLabel("Titre:")

self.titre\_input = QLineEdit()

xlabel\_label = QLabel("Nom de l'axe X:")

self.xlabel\_input = QLineEdit()

ylabel\_label = QLabel("Nom de l'axe Y:")

self.ylabel\_input = QLineEdit()

valeurs\_label = QLabel("Valeurs (séparées par des virgules):")

self.valeurs\_input = QLineEdit()

categories\_label = QLabel("Catégories (pour barres, séparées par des virgules):")

self.categories\_input = QLineEdit()

create\_button = QPushButton("Créer Graphique")

create\_button.clicked.connect(lambda: self.create\_graphique(dialog))

layout.addWidget(type\_graphique\_label)

layout.addWidget(self.type\_graphique\_combo)

layout.addWidget(titre\_label)

layout.addWidget(self.titre\_input)

layout.addWidget(xlabel\_label)

layout.addWidget(self.xlabel\_input)

layout.addWidget(ylabel\_label)

layout.addWidget(self.ylabel\_input)

layout.addWidget(valeurs\_label)

layout.addWidget(self.valeurs\_input)

layout.addWidget(categories\_label)

layout.addWidget(self.categories\_input)

layout.addWidget(create\_button)

dialog.setLayout(layout)

dialog.exec()

def create\_graphique(self, dialog):

type\_graphique = self.type\_graphique\_combo.currentText()

valeurs = list(map(float, self.valeurs\_input.text().split(',')))

categories = self.categories\_input.text().split(',') if type\_graphique == 'barres' else None

boss\_tab\_index = self.tab\_widget.indexOf(self.tab\_widget.widget(1)) # Assumer que "Boss" est à l'index 1

if boss\_tab\_index != -1:

boss\_tab = self.tab\_widget.widget(boss\_tab\_index)

if hasattr(self, 'graphique\_canvas') and self.graphique\_canvas is not None:

self.graphique\_canvas.deleteLater()

graphique = Graphique()

self.graphique\_canvas = graphique.tracer\_graphique(type\_graphique, valeurs, categories)

boss\_tab.layout().addWidget(self.graphique\_canvas)

dialog.accept()

def user\_role\_can\_access(self, role):

if self.current\_user\_role == "Boss":

return True

elif self.current\_user\_role == "Admin":

return role in ["Admin", "Employé"]

elif self.current\_user\_role == "Employé":

return role == "Employé"

return False

def update\_last\_index(self, index):

self.last\_index = index

def go\_back(self):

if self.last\_index > 0:

self.tab\_widget.setCurrentIndex(self.last\_index - 1)

def update\_time(self):

current\_time = datetime.now().strftime("%H:%M:%S")

self.time\_label.setText(f"Heure actuelle : {current\_time}")

def update\_user\_info(self, name, role):

self.current\_user\_role = role

self.user\_info\_label.setText(f"Connecté : {name} (Rôle : {role})" if role else "Connecté : Invité")

self.tab\_widget.clear() # Vider les onglets existants

self.create\_tabs() # Recréer les onglets en fonction du rôle

self.update\_button\_visibility()

def update\_button\_visibility(self):

if self.current\_user\_role is None:

self.sign\_out\_button.setVisible(False)

self.connect\_button.setVisible(True)

else:

self.sign\_out\_button.setVisible(True)

self.connect\_button.setVisible(False)

def sign\_out(self):

QMessageBox.information(self, "Déconnexion", "Vous êtes déconnecté.")

self.update\_user\_info("Invité", None)

def show\_login\_dialog(self):

dialog = QDialog(self)

dialog.setWindowTitle("Connexion")

layout = QVBoxLayout()

username\_label = QLabel("Identifiant:")

self.username\_input = QLineEdit()

password\_label = QLabel("Mot de passe:")

self.password\_input = QLineEdit()

self.password\_input.setEchoMode(QLineEdit.EchoMode.Password)

login\_button = QPushButton("Se connecter")

login\_button.clicked.connect(lambda: self.login(dialog))

layout.addWidget(username\_label)

layout.addWidget(self.username\_input)

layout.addWidget(password\_label)

layout.addWidget(self.password\_input)

layout.addWidget(login\_button)

dialog.setLayout(layout)

dialog.exec()

def login(self, dialog):

username = self.username\_input.text()

password = self.password\_input.text()

if username == "admin" and password == "admin":

self.update\_user\_info(username, "Admin")

dialog.accept()

elif username == "boss" and password == "boss":

self.update\_user\_info(username, "Boss")

dialog.accept()

elif username == "employe" and password == "employe":

self.update\_user\_info(username, "Employé")

dialog.accept()

else:

QMessageBox.warning(self, "Erreur de connexion", "Identifiant ou mot de passe incorrect.")

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = MainWindow()

window.show()

sys.exit(app.exec())

**Changements Clés :**

1. **Création d'Onglets en Fonction du Rôle** :
   * Dans la méthode create\_tabs, les onglets sont maintenant ajoutés uniquement si l'utilisateur connecté a le droit d'accéder à ces onglets.
2. **Vider et Recréer les Onglets** :
   * Dans la méthode update\_user\_info, on vide d'abord les onglets existants et on les recrée en fonction du rôle de l'utilisateur connecté.

**Instructions :**

1. Copie le code mis à jour dans un fichier (par exemple, app\_graphique\_dans\_onglet\_v2.py).
2. Exécute le fichier avec Python :

bash

Copier le code

python app\_graphique\_dans\_onglet\_v2.py

Avec ces modifications, seuls les onglets auxquels l'utilisateur a accès seront affichés en fonction de son rôle. Si tu as d'autres questions ou ajustements, fais-le moi savoir !

4o mini