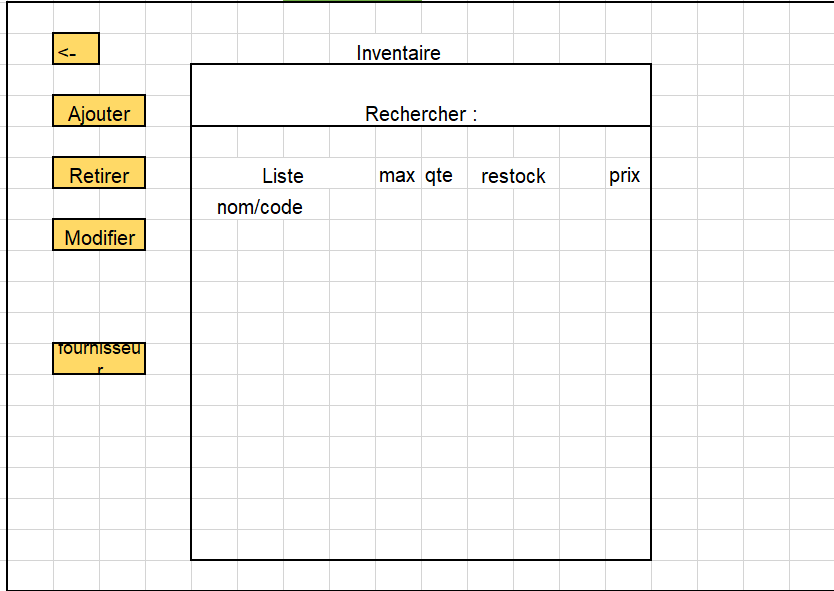
Arslan Khaoua

Mes prompts

# Question 1

## Prompt



Fais-moi en Qt python une interface ressemblant à la première image et ayant comme code d'usage : en rentrant dans la page stock affiche automatiquement les produit du stock du magasin, ensuite, vas prendre la succursale de l’employés et vas chercher dans la table sql le stock contenue dans son magasin

## Reponse

import sys

from PyQt5.QtWidgets import (

QApplication, QMainWindow, QPushButton, QLabel, QLineEdit, QTableWidget,

QTableWidgetItem, QVBoxLayout, QHBoxLayout, QWidget, QGridLayout

)

from PyQt5.QtCore import Qt

class StockManagementApp(QMainWindow):

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

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

self.setWindowTitle("Stock Management")

self.setGeometry(100, 100, 800, 600)

# Create the main layout

main\_layout = QGridLayout()

# Left-side buttons layout (Ajouter, Retirer, Modifier, etc.)

self.add\_button = QPushButton("Ajouter")

self.remove\_button = QPushButton("Retirer")

self.modify\_button = QPushButton("Modifier")

self.back\_button = QPushButton("<-")

self.supplier\_button = QPushButton("Fournisseur")

button\_layout = QVBoxLayout()

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

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

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

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

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

# Add button layout to the grid

main\_layout.addLayout(button\_layout, 0, 0)

# Title of the inventory

title\_label = QLabel("Inventaire")

title\_label.setAlignment(Qt.AlignCenter)

main\_layout.addWidget(title\_label, 0, 1)

# Search bar

search\_label = QLabel("Rechercher :")

self.search\_input = QLineEdit()

search\_layout = QHBoxLayout()

search\_layout.addWidget(search\_label)

search\_layout.addWidget(self.search\_input)

main\_layout.addLayout(search\_layout, 1, 1)

# Stock table (Liste, max, qte, restock, prix)

self.stock\_table = QTableWidget()

self.stock\_table.setColumnCount(5)

self.stock\_table.setHorizontalHeaderLabels(

["Liste (nom/code)", "max", "qte", "restock", "prix"]

)

# Dummy data to simulate stock items

self.stock\_table.setRowCount(3)

self.stock\_table.setItem(0, 0, QTableWidgetItem("Item 1"))

self.stock\_table.setItem(0, 1, QTableWidgetItem("100"))

self.stock\_table.setItem(0, 2, QTableWidgetItem("50"))

self.stock\_table.setItem(0, 3, QTableWidgetItem("20"))

self.stock\_table.setItem(0, 4, QTableWidgetItem("10.00"))

self.stock\_table.setItem(1, 0, QTableWidgetItem("Item 2"))

self.stock\_table.setItem(1, 1, QTableWidgetItem("200"))

self.stock\_table.setItem(1, 2, QTableWidgetItem("150"))

self.stock\_table.setItem(1, 3, QTableWidgetItem("50"))

self.stock\_table.setItem(1, 4, QTableWidgetItem("20.00"))

self.stock\_table.setItem(2, 0, QTableWidgetItem("Item 3"))

self.stock\_table.setItem(2, 1, QTableWidgetItem("50"))

self.stock\_table.setItem(2, 2, QTableWidgetItem("30"))

self.stock\_table.setItem(2, 3, QTableWidgetItem("10"))

self.stock\_table.setItem(2, 4, QTableWidgetItem("5.00"))

# Add table to layout

main\_layout.addWidget(self.stock\_table, 2, 1)

# Set central widget

central\_widget = QWidget()

central\_widget.setLayout(main\_layout)

self.setCentralWidget(central\_widget)

# Connect button actions to methods

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

self.remove\_button.clicked.connect(self.remove\_item)

self.modify\_button.clicked.connect(self.modify\_item)

def add\_item(self):

# Code to add an item to the stock

print("Add item clicked")

# Add logic to open a form to add stock item

def remove\_item(self):

# Code to remove the selected item from the stock

print("Remove item clicked")

# Add logic to remove selected row from the table

def modify\_item(self):

# Code to modify the selected item in the stock

print("Modify item clicked")

# Add logic to modify the selected row

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

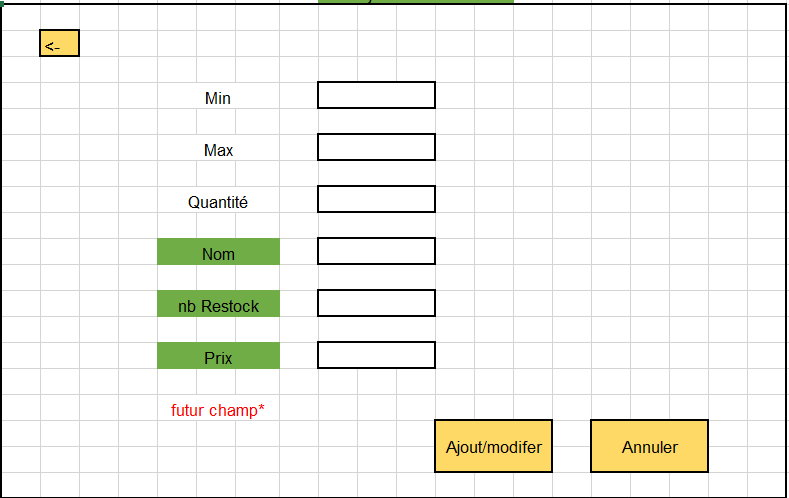
window = StockManagementApp()

window.show()

sys.exit(app.exec\_())

# Question 2

## Prompt



fait en sorte que quand on clique sur le bouton ajouter sa nous emmené sur une page comme celle de l'image

## response

import sys

from PyQt5.QtWidgets import (

QApplication, QMainWindow, QPushButton, QLabel, QLineEdit, QTableWidget,

QTableWidgetItem, QVBoxLayout, QHBoxLayout, QWidget, QGridLayout, QDialog

)

from PyQt5.QtCore import Qt

class AddModifyDialog(QDialog):

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

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

self.setWindowTitle("Ajouter/Modifier")

# Create the layout

layout = QGridLayout()

# Labels and input fields

labels = ["Min", "Max", "Quantité", "Nom", "nb Restock", "Prix", "futur champ\*"]

self.inputs = {}

for i, label in enumerate(labels):

lbl = QLabel(label)

input\_field = QLineEdit()

layout.addWidget(lbl, i, 0)

layout.addWidget(input\_field, i, 1)

self.inputs[label] = input\_field

# Buttons for 'Ajouter/Modifier' and 'Annuler'

self.add\_modify\_button = QPushButton("Ajouter/Modifier")

self.cancel\_button = QPushButton("Annuler")

button\_layout = QHBoxLayout()

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

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

layout.addLayout(button\_layout, len(labels), 1)

# Set the layout

self.setLayout(layout)

# Connect cancel button to close the dialog

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

class StockManagementApp(QMainWindow):

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

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

self.setWindowTitle("Stock Management")

self.setGeometry(100, 100, 800, 600)

# Create the main layout

main\_layout = QGridLayout()

# Left-side buttons layout (Ajouter, Retirer, Modifier, etc.)

self.add\_button = QPushButton("Ajouter")

self.remove\_button = QPushButton("Retirer")

self.modify\_button = QPushButton("Modifier")

self.back\_button = QPushButton("<-")

self.supplier\_button = QPushButton("Fournisseur")

button\_layout = QVBoxLayout()

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

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

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

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

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

# Add button layout to the grid

main\_layout.addLayout(button\_layout, 0, 0)

# Title of the inventory

title\_label = QLabel("Inventaire")

title\_label.setAlignment(Qt.AlignCenter)

main\_layout.addWidget(title\_label, 0, 1)

# Search bar

search\_label = QLabel("Rechercher :")

self.search\_input = QLineEdit()

search\_layout = QHBoxLayout()

search\_layout.addWidget(search\_label)

search\_layout.addWidget(self.search\_input)

main\_layout.addLayout(search\_layout, 1, 1)

# Stock table (Liste, max, qte, restock, prix)

self.stock\_table = QTableWidget()

self.stock\_table.setColumnCount(5)

self.stock\_table.setHorizontalHeaderLabels(

["Liste (nom/code)", "max", "qte", "restock", "prix"]

)

# Dummy data to simulate stock items

self.stock\_table.setRowCount(3)

self.stock\_table.setItem(0, 0, QTableWidgetItem("Item 1"))

self.stock\_table.setItem(0, 1, QTableWidgetItem("100"))

self.stock\_table.setItem(0, 2, QTableWidgetItem("50"))

self.stock\_table.setItem(0, 3, QTableWidgetItem("20"))

self.stock\_table.setItem(0, 4, QTableWidgetItem("10.00"))

self.stock\_table.setItem(1, 0, QTableWidgetItem("Item 2"))

self.stock\_table.setItem(1, 1, QTableWidgetItem("200"))

self.stock\_table.setItem(1, 2, QTableWidgetItem("150"))

self.stock\_table.setItem(1, 3, QTableWidgetItem("50"))

self.stock\_table.setItem(1, 4, QTableWidgetItem("20.00"))

self.stock\_table.setItem(2, 0, QTableWidgetItem("Item 3"))

self.stock\_table.setItem(2, 1, QTableWidgetItem("50"))

self.stock\_table.setItem(2, 2, QTableWidgetItem("30"))

self.stock\_table.setItem(2, 3, QTableWidgetItem("10"))

self.stock\_table.setItem(2, 4, QTableWidgetItem("5.00"))

# Add table to layout

main\_layout.addWidget(self.stock\_table, 2, 1)

# Set central widget

central\_widget = QWidget()

central\_widget.setLayout(main\_layout)

self.setCentralWidget(central\_widget)

# Connect button actions to methods

self.add\_button.clicked.connect(self.open\_add\_modify\_dialog)

def open\_add\_modify\_dialog(self):

# Open the Add/Modify dialog

dialog = AddModifyDialog()

dialog.exec\_()

def remove\_item(self):

# Code to remove the selected item from the stock

print("Remove item clicked")

# Add logic to remove selected row from the table

def modify\_item(self):

# Code to modify the selected item in the stock

print("Modify item clicked")

# Add logic to modify the selected row

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

window = StockManagementApp()

window.show()

sys.exit(app.exec\_())

# Question 3

## Prompt

Fais-moi une fonction qui fait en sorte que quand je clique sur le bouton supprimer je rentre dans un mode sélection qui fait en sorte que si je clique sur un item sa me demande si je veux le supprimer ou non si je dis oui l'item et retirer de la base de données sinon sa annule ma sélection

## Response

from PySide6.QtWidgets import QMessageBox

class QStock(QWidget):

def \_\_init\_\_(self, parent=None):

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

# ... (Autres parties du code)

# Connexion à la base de données SQLite

self.conn = sqlite3.connect("erp\_database.db")

self.cursor = self.conn.cursor()

# Créer les tables si elles n'existent pas

self.create\_tables()

# ... (Autres parties du code)

# Tableau pour afficher les stocks

self.stock\_table = QTableWidget()

self.stock\_table.setColumnCount(6)

self.stock\_table.setHorizontalHeaderLabels(

["Nom", "Code Produit", "Max", "Quantité", "Restock", "Prix"]

)

# Charger les données du stock depuis la base de données

self.load\_stock\_data()

# ... (Autres parties du code)

# Connecter les boutons à des méthodes

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

remove\_button.clicked.connect(self.remove\_item)

modify\_button.clicked.connect(self.modify\_item)

def remove\_item(self):

"""Activer le mode sélection pour supprimer un produit."""

# Activer la sélection dans le tableau

self.stock\_table.setSelectionBehavior(QTableWidget.SelectRows)

self.stock\_table.setSelectionMode(QTableWidget.SingleSelection)

# Connecter l'événement de clic à la méthode `confirm\_deletion`

self.stock\_table.itemClicked.connect(self.confirm\_deletion)

def confirm\_deletion(self, item):

"""Demander confirmation avant de supprimer un élément."""

# Obtenir la ligne de l'élément sélectionné

row = item.row()

# Récupérer les informations du produit dans la ligne sélectionnée

nom\_produit = self.stock\_table.item(row, 0).text()

code\_produit = self.stock\_table.item(row, 1).text()

# Boîte de dialogue de confirmation

confirmation\_dialog = QMessageBox()

confirmation\_dialog.setWindowTitle("Confirmer la suppression")

confirmation\_dialog.setText(f"Voulez-vous vraiment supprimer le produit '{nom\_produit}' (Code: {code\_produit}) ?")

confirmation\_dialog.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

confirmation\_dialog.setIcon(QMessageBox.Warning)

# Si l'utilisateur confirme la suppression

if confirmation\_dialog.exec\_() == QMessageBox.Yes:

self.delete\_product(code\_produit)

else:

# Annuler la sélection si l'utilisateur ne veut pas supprimer

self.stock\_table.clearSelection()

# Désactiver la connexion à l'événement après la suppression ou l'annulation

self.stock\_table.itemClicked.disconnect()

def delete\_product(self, code\_produit):

"""Supprimer le produit de la base de données et mettre à jour le tableau."""

try:

# Supprimer le produit de la base de données

self.cursor.execute("DELETE FROM Produits WHERE code\_produit = ?", (code\_produit,))

self.cursor.execute("DELETE FROM Stocks WHERE id\_produit = (SELECT id\_produit FROM Produits WHERE code\_produit = ?)", (code\_produit,))

self.conn.commit()

# Recharger les données dans le tableau

self.load\_stock\_data()

# Message de confirmation

print(f"Le produit avec le code {code\_produit} a été supprimé.")

except Exception as e:

print(f"Erreur lors de la suppression du produit: {e}")

# Question 4

## Prompt

Fait en sorte que quand je clique sur le bouton modifier je passe comme en mode sélection qui fait en sorte que quand je clique sur un item il m'ouvre une boite comme celle d’ajouter produit, mais qui vas, une fois valider, modifier les valeur de l'item sélectionner dans la base de données

## response

class QStock(QWidget):

def \_\_init\_\_(self, parent=None):

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

# ... (Autres parties du code)

# Connexion à la base de données SQLite

self.conn = sqlite3.connect("erp\_database.db")

self.cursor = self.conn.cursor()

# Créer les tables si elles n'existent pas

self.create\_tables()

# Tableau pour afficher les stocks

self.stock\_table = QTableWidget()

self.stock\_table.setColumnCount(6)

self.stock\_table.setHorizontalHeaderLabels(

["Nom", "Code Produit", "Max", "Quantité", "Restock", "Prix"]

)

# Charger les données du stock depuis la base de données

self.load\_stock\_data()

# ... (Autres parties du code)

# Connecter les boutons à des méthodes

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

remove\_button.clicked.connect(self.remove\_item)

modify\_button.clicked.connect(self.modify\_item)

def modify\_item(self):

"""Activer le mode sélection pour modifier un produit."""

# Activer la sélection dans le tableau

self.stock\_table.setSelectionBehavior(QTableWidget.SelectRows)

self.stock\_table.setSelectionMode(QTableWidget.SingleSelection)

# Connecter l'événement de clic à la méthode `open\_modify\_dialog`

self.stock\_table.itemClicked.connect(self.open\_modify\_dialog)

def open\_modify\_dialog(self, item):

"""Ouvrir la boîte de dialogue avec les informations de l'élément sélectionné pour modification."""

# Obtenir la ligne de l'élément sélectionné

row = item.row()

# Récupérer les informations du produit dans la ligne sélectionnée

nom\_produit = self.stock\_table.item(row, 0).text()

code\_produit = self.stock\_table.item(row, 1).text()

max\_qte = self.stock\_table.item(row, 2).text()

quantite = self.stock\_table.item(row, 3).text()

restock = self.stock\_table.item(row, 4).text()

prix = self.stock\_table.item(row, 5).text()

# Créer et afficher la boîte de dialogue pour modification

dialog = AddModifyDialog()

# Pré-remplir les champs avec les informations existantes

dialog.inputs["Nom"].setText(nom\_produit)

dialog.inputs["Min"].setText(max\_qte) # Ou ajustez en fonction de vos champs

dialog.inputs["Quantité"].setText(quantite)

dialog.inputs["nb Restock"].setText(restock)

dialog.inputs["Prix"].setText(prix)

# Changer le texte du bouton pour 'Modifier'

dialog.add\_modify\_button.setText("Modifier")

# Si l'utilisateur valide la modification

if dialog.exec\_() == QDialog.Accepted:

# Récupérer les nouvelles valeurs entrées par l'utilisateur

new\_nom\_produit = dialog.inputs["Nom"].text()

new\_max\_qte = dialog.inputs["Min"].text()

new\_quantite = dialog.inputs["Quantité"].text()

new\_restock = dialog.inputs["nb Restock"].text()

new\_prix = dialog.inputs["Prix"].text()

# Mettre à jour les informations dans la base de données

self.update\_product(

code\_produit, new\_nom\_produit, new\_max\_qte, new\_quantite, new\_restock, new\_prix

)

# Recharger les données du tableau

self.load\_stock\_data()

# Désactiver la connexion à l'événement après la modification ou l'annulation

self.stock\_table.itemClicked.disconnect()

def update\_product(self, code\_produit, nom, max\_qte, quantite, restock, prix):

"""Mettre à jour le produit dans la base de données."""

try:

self.cursor.execute("""

UPDATE Produits

SET nom = ?, max\_qte = ?, quantite = ?, restock = ?, prix = ?

WHERE code\_produit = ?

""", (nom, max\_qte, quantite, restock, prix, code\_produit))

self.conn.commit()

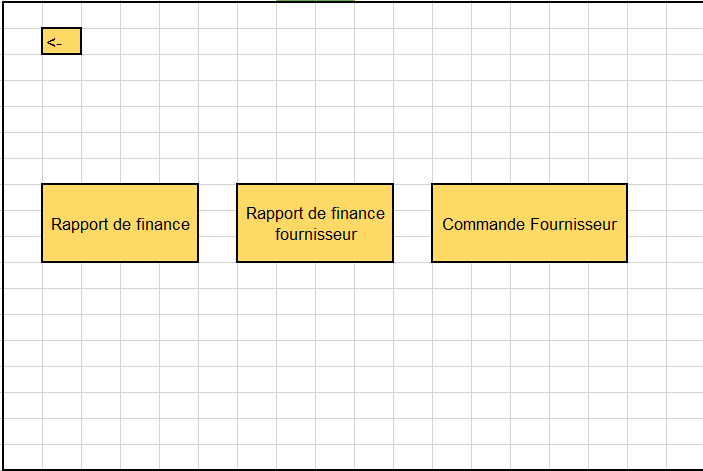
print(f"Produit {nom} mis à jour avec succès.")

except Exception as e:

print(f"Erreur lors de la mise à jour du produit: {e}")

# Question 5

## Prompt



Dans la main page rajoute un button Finance qui, une fois cliquer, nous emmène sur une page qui ressemble à l'image

## Response

### Étape 1 : Créer la page de finance

Ajoutez une classe QFinance qui définit l'interface de la page de finance :

python

Copier le code

from PySide6.QtWidgets import QWidget, QVBoxLayout, QGridLayout, QPushButton, QLabel

from PySide6.QtCore import Qt, Signal

class QFinance(QWidget):

go\_back = Signal()

def \_\_init\_\_(self, parent=None):

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

# Layout principal

layout = QGridLayout()

# Bouton de retour

back\_button = QPushButton("<-")

back\_button.clicked.connect(self.go\_back.emit) # Émet le signal pour revenir en arrière

layout.addWidget(back\_button, 0, 0)

# Titre

title = QLabel("Finance")

title.setAlignment(Qt.AlignCenter)

layout.addWidget(title, 0, 1)

# Boutons principaux

report\_finance\_button = QPushButton("Rapport de finance")

report\_supplier\_finance\_button = QPushButton("Rapport de finance fournisseur")

supplier\_order\_button = QPushButton("Commande Fournisseur")

# Style des boutons (facultatif)

button\_style = "background-color: #f0a500; padding: 20px; font-size: 14px; font-weight: bold;"

report\_finance\_button.setStyleSheet(button\_style)

report\_supplier\_finance\_button.setStyleSheet(button\_style)

supplier\_order\_button.setStyleSheet(button\_style)

# Ajout des boutons au layout

layout.addWidget(report\_finance\_button, 1, 1)

layout.addWidget(report\_supplier\_finance\_button, 1, 2)

layout.addWidget(supplier\_order\_button, 1, 3)

self.setLayout(layout)

### Étape 2 : Intégrer la Page de Finance dans Vue

Ajoutez une instance de QFinance dans le QStackedWidget de Vue, et connectez le signal go\_back pour revenir à la page d'accueil.

python

Copier le code

class Vue(QMainWindow):

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

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

self.controleur = controleur

self.setWindowTitle("Application ERP")

self.setGeometry(100, 100, 800, 600)

self.stacked\_widget = QStackedWidget()

self.setCentralWidget(self.stacked\_widget)

# Création des différents frames

self.frame\_connexion = self.creer\_frame\_connexion()

self.frame\_vente = self.creer\_frame\_vente()

self.frame\_stock = QStock(self)

self.frame\_produit = QProduit(self, self.controleur.db\_manager)

self.frame\_splash = self.creer\_frame\_splash()

self.frame\_greant\_global = QGerantGlobal(self)

self.frame\_succursale = QSuccursale(self)

self.frame\_fournisseur = QFournisseur(self, self.controleur.db\_manager)

self.frame\_finance = QFinance(self) # Instanciation de la page finance

# Ajout des frames au QStackedWidget

self.stacked\_widget.addWidget(self.frame\_connexion)

self.stacked\_widget.addWidget(self.frame\_splash)

self.stacked\_widget.addWidget(self.frame\_vente)

self.stacked\_widget.addWidget(self.frame\_stock)

self.stacked\_widget.addWidget(self.frame\_greant\_global)

self.stacked\_widget.addWidget(self.frame\_succursale)

self.stacked\_widget.addWidget(self.frame\_produit)

self.stacked\_widget.addWidget(self.frame\_fournisseur)

self.stacked\_widget.addWidget(self.frame\_finance) # Ajouter la page finance

# Connecter le signal `go\_back` de `frame\_finance` pour revenir à la page d'accueil

self.frame\_finance.go\_back.connect(self.basculer\_vers\_splash)

# Affichage initial

self.basculer\_vers\_connexion()

# ... (autres méthodes)

def basculer\_vers\_finance(self):

"""Afficher la page de finance."""

self.stacked\_widget.setCurrentWidget(self.frame\_finance)

### Étape 3 : Ajouter un Bouton "Finance" dans la Page d'Accueil

Modifiez creer\_frame\_splash pour inclure le bouton "Finance" qui appelle basculer\_vers\_finance.

python

Copier le code

def creer\_frame\_splash(self):

widget = QWidget()

layout = QVBoxLayout()

titre = QLabel("ERP Manager")

titre.setAlignment(Qt.AlignCenter)

titre.setStyleSheet("font-size: 24px; font-weight: bold;")

layout.addWidget(titre)

sous\_titre = QLabel("Système de gestion intégré pour votre entreprise")

sous\_titre.setAlignment(Qt.AlignCenter)

layout.addWidget(sous\_titre)

buttons\_layout = QHBoxLayout()

self.button\_gestion = QPushButton("Gestion interne")

self.button\_gestion.clicked.connect(lambda: self.controleur.action\_splash("gestion"))

self.button\_options = QPushButton("Options d'utilisation")

self.button\_options.clicked.connect(lambda: self.controleur.action\_splash("options"))

self.button\_formulaire = QPushButton("Formulaire")

self.button\_formulaire.clicked.connect(lambda: self.controleur.action\_splash("formulaire"))

self.button\_stock = QPushButton("Stock")

self.button\_stock.clicked.connect(lambda: self.controleur.action\_splash("stock"))

self.button\_produit = QPushButton("Produit")

self.button\_produit.clicked.connect(lambda: self.controleur.action\_splash("produit"))

self.button\_fournisseur = QPushButton("Fournisseur")

self.button\_fournisseur.clicked.connect(lambda: self.controleur.action\_splash("fournisseur"))

# Nouveau bouton Finance

self.button\_finance = QPushButton("Finance")

self.button\_finance.clicked.connect(self.basculer\_vers\_finance)

buttons\_layout.addWidget(self.button\_gestion)

buttons\_layout.addWidget(self.button\_options)

buttons\_layout.addWidget(self.button\_formulaire)

buttons\_layout.addWidget(self.button\_stock)

buttons\_layout.addWidget(self.button\_produit)

buttons\_layout.addWidget(self.button\_fournisseur)

buttons\_layout.addWidget(self.button\_finance) # Ajout du bouton Finance

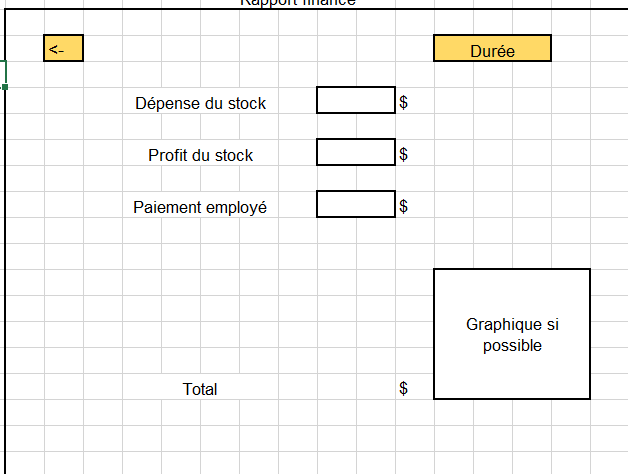
layout.addLayout(buttons\_layout)

widget.setLayout(layout)

return widget

# Question 6

## Prompt



Quand on clique sur le bouton rapport de finance lance une page qui ressemble à l'image et qui affiche, dans les champs, le total des dépenses du stock, le total des ventes et le total des salaires des employés faire une addition des trois informations et le mettre dans total

## Response

from PySide6.QtWidgets import QWidget, QVBoxLayout, QGridLayout, QPushButton, QLabel, QLineEdit

from PySide6.QtCore import Qt, Signal

class QFinanceReport(QWidget):

go\_back = Signal()

def \_\_init\_\_(self, parent=None):

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

# Layout principal

layout = QGridLayout()

# Bouton de retour

back\_button = QPushButton("<-")

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

layout.addWidget(back\_button, 0, 0)

# Bouton Durée (on peut ajouter un signal ici pour appliquer des filtres si nécessaire)

duration\_button = QPushButton("Durée")

layout.addWidget(duration\_button, 0, 2)

# Champs pour les données financières

self.expense\_field = QLineEdit()

self.profit\_field = QLineEdit()

self.salary\_field = QLineEdit()

# Définit les champs en lecture seule

self.expense\_field.setReadOnly(True)

self.profit\_field.setReadOnly(True)

self.salary\_field.setReadOnly(True)

# Etiquettes et champs

layout.addWidget(QLabel("Dépense du stock"), 1, 0)

layout.addWidget(self.expense\_field, 1, 1)

layout.addWidget(QLabel("$"), 1, 2)

layout.addWidget(QLabel("Profit du stock"), 2, 0)

layout.addWidget(self.profit\_field, 2, 1)

layout.addWidget(QLabel("$"), 2, 2)

layout.addWidget(QLabel("Paiement employé"), 3, 0)

layout.addWidget(self.salary\_field, 3, 1)

layout.addWidget(QLabel("$"), 3, 2)

# Champ pour le total

self.total\_field = QLineEdit()

self.total\_field.setReadOnly(True)

layout.addWidget(QLabel("Total"), 4, 0)

layout.addWidget(self.total\_field, 4, 1)

layout.addWidget(QLabel("$"), 4, 2)

# Bouton pour afficher un graphique

layout.addWidget(QLabel("Graphique si possible"), 5, 1, 2, 1)

self.setLayout(layout)

def set\_financial\_data(self, expenses, profits, salaries):

"""Remplit les champs financiers et calcule le total."""

self.expense\_field.setText(f"{expenses:.2f}")

self.profit\_field.setText(f"{profits:.2f}")

self.salary\_field.setText(f"{salaries:.2f}")

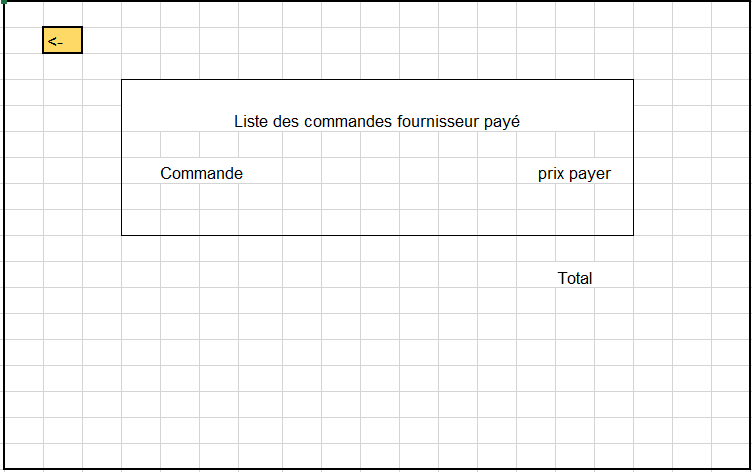
# Calculer le total

total = expenses + profits + salaries

self.total\_field.setText(f"{total:.2f}")

# Question 7

## Prompt



Fait en sorte que quand je clique dans la page finance sur le bouton Rapport de finance fournisseur sa vas m'emmener sur la page ressemblant à la photo qui vas lister les commandes et leur prix depuis la table Fourniseur\_produits

## Réponse

### Étape 1 : Créer QFinanceSupplierReport

Voici une classe QFinanceSupplierReport qui correspond au design de la page de la photo. Cette page liste les commandes fournisseurs payées avec leur prix, et montre le total en bas.

python

Copier le code

from PySide6.QtWidgets import QWidget, QVBoxLayout, QGridLayout, QPushButton, QLabel, QTableWidget, QTableWidgetItem

from PySide6.QtCore import Qt, Signal

from ERP\_data\_base import DatabaseManager

class QFinanceSupplierReport(QWidget):

go\_back = Signal() # Signal pour retourner à la page précédente

def \_\_init\_\_(self, parent=None):

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

# Initialisation de la base de données

self.db\_manager = DatabaseManager("erp\_database.db")

# Configuration de l'interface

layout = QVBoxLayout()

# Bouton de retour

back\_button = QPushButton("<-")

back\_button.setStyleSheet("background-color: #f0a500; font-weight: bold;")

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

layout.addWidget(back\_button, alignment=Qt.AlignLeft)

# Titre

title = QLabel("Liste des commandes fournisseur payées")

title.setAlignment(Qt.AlignCenter)

title.setStyleSheet("font-size: 18px; font-weight: bold;")

layout.addWidget(title)

# Tableau pour les commandes et leurs prix

self.table = QTableWidget()

self.table.setColumnCount(2)

self.table.setHorizontalHeaderLabels(["Commande", "Prix payé"])

layout.addWidget(self.table)

# Étiquette pour le total

self.total\_label = QLabel("Total : ")

self.total\_label.setAlignment(Qt.AlignRight)

layout.addWidget(self.total\_label)

# Ajouter le layout principal

self.setLayout(layout)

# Charger les données

self.load\_supplier\_payments()

def load\_supplier\_payments(self):

"""Charge les commandes payées depuis la base de données et affiche dans le tableau."""

query = """

SELECT commande\_id, prix\_paye

FROM CommandesFournisseur

WHERE statut = 'Payé'

"""

results = self.db\_manager.execute\_query(query, ())

# Remplissage du tableau

self.table.setRowCount(len(results))

total = 0

for row, data in enumerate(results):

commande\_item = QTableWidgetItem(str(data["commande\_id"]))

prix\_item = QTableWidgetItem(f"{data['prix\_paye']:.2f} €")

self.table.setItem(row, 0, commande\_item)

self.table.setItem(row, 1, prix\_item)

total += data["prix\_paye"]

# Mettre à jour l'affichage du total

self.total\_label.setText(f"Total : {total:.2f} €")

Cette classe QFinanceSupplierReport :

* Affiche un tableau de commandes fournisseurs payées (à partir de la table CommandesFournisseur).
* Calcule et affiche le total des prix payés.
* Dispose d’un bouton de retour pour revenir à la page précédente.

### Étape 2 : Modifier QFinance pour naviguer vers QFinanceSupplierReport

Dans la classe QFinance, ajoutez un slot pour afficher cette page quand le bouton "Rapport de finance fournisseur" est cliqué :

1. **Importer la classe QFinanceSupplierReport**.
2. **Connecter le bouton report\_supplier\_finance\_button** pour ouvrir la nouvelle page.

Voici le code mis à jour pour la classe QFinance :

python

Copier le code

from PySide6.QtWidgets import QWidget, QVBoxLayout, QGridLayout, QPushButton, QLabel

from PySide6.QtCore import Qt, Signal

from ERP\_data\_base import DatabaseManager

from .QFinanceSupplierReport import QFinanceSupplierReport # Import de la nouvelle page

class QFinance(QWidget):

go\_back = Signal()

def \_\_init\_\_(self, parent=None):

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

self.vue = parent

self.db\_manager = DatabaseManager("erp\_database.db")

# Layout principal

layout = QGridLayout()

# Bouton de retour

back\_button = QPushButton("<-")

back\_button.clicked.connect(self.go\_back.emit) # Émet le signal pour revenir en arrière

layout.addWidget(back\_button, 0, 0)

# Bouton pour le rapport de finance fournisseur

report\_supplier\_finance\_button = QPushButton("Rapport de finance fournisseur")

report\_supplier\_finance\_button.clicked.connect(self.show\_supplier\_report) # Connexion au slot

layout.addWidget(report\_supplier\_finance\_button, 1, 2)

self.setLayout(layout)

def show\_supplier\_report(self):

"""Affiche la page des commandes fournisseur payées."""

self.supplier\_report\_page = QFinanceSupplierReport(self)

self.supplier\_report\_page.go\_back.connect(self.show) # Retour à la page finance

self.supplier\_report\_page.show()

self.hide()

# Question 8

## Prompt

## Réponse