main.py

import tkinter as tk

import customtkinter

import pygame

import sqlite3

from tkinter import ttk, filedialog, messagebox

from app.login import LoginScreen

from app.navigation import CreateNavigationPane

from app.dashboard import DashboardScreen

from app.settings import SettingsScreen

from app.profile import ProfileScreen

from app.terminate\_user import TerminateUserScreen

from app.backend import Backend

from app.report import GenerateReportPopup

from app.inventory\_management import ManageInventoryScreen

import csv

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from app.inventory\_management import ManageStaffScreen

from app.database import initialize\_database

import threading

from time import sleep

import time

from PIL import Image, ImageTk

# Initialize Pygame mixer at the start of the script

pygame.mixer.init()

customtkinter.set\_appearance\_mode("Light") # or "Dark"

customtkinter.set\_default\_color\_theme("blue") # or any other theme

# Initialize backend

backend = Backend()

backend.create\_database()

initialize\_database()

class LoadingScreen:

"""A modern, frameless loading screen design."""

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

self.root = root

self.stop\_animation = False # Flag to stop animation

# Overlay Frame for full-screen loading

self.loading\_frame = customtkinter.CTkFrame(

root,

fg\_color="#ffffff"

)

self.loading\_frame.place(relx=0.5, rely=0.5, anchor="center")

# Compact Container for the loading screen content

self.container = customtkinter.CTkFrame(

self.loading\_frame,

corner\_radius=10,

fg\_color="#ffffff"

)

self.container.pack(padx=20, pady=20)

# Loading Label with modern styling

self.loading\_label = customtkinter.CTkLabel(

self.container,

text="Please Wait...",

font=("Helvetica Neue", 14, "bold"),

text\_color="#c0392b"

)

self.loading\_label.pack(pady=(10, 5))

# Minimalist Progress Bar

self.progress\_bar = customtkinter.CTkProgressBar(

self.container,

orientation="horizontal",

mode="indeterminate",

width=150,

fg\_color="#c0392b",

progress\_color="#ffffff"

)

self.progress\_bar.pack(pady=(5, 10))

self.progress\_bar.start() # Start the animation

# Optional Animated Icon (Simulated as "GIF animation")

self.loading\_icon = customtkinter.CTkLabel(

self.container,

text="🔄",

font=("Arial", 20, "bold"),

text\_color="#c0392b"

)

self.loading\_icon.pack()

self.animate\_gif()

# Make the window frameless and center it

self.root.overrideredirect(True)

self.center\_window(300, 200)

def center\_window(self, width, height):

"""Center the window on the screen."""

screen\_width = self.root.winfo\_screenwidth()

screen\_height = self.root.winfo\_screenheight()

x = (screen\_width // 2) - (width // 2)

y = (screen\_height // 2) - (height // 2)

self.root.geometry(f"{width}x{height}+{x}+{y}")

def animate\_gif(self):

"""Simulate GIF animation using a simple text rotation."""

if not self.stop\_animation: # Check if the animation should stop

current\_text = self.loading\_icon.cget("text")

next\_text = "🔄" if current\_text != "🔄" else "🔁" # Alternate icons

self.loading\_icon.configure(text=next\_text)

self.root.after(200, self.animate\_gif) # Repeat every 200ms

def destroy(self):

"""Remove the loading screen and stop animation."""

self.stop\_animation = True # Stop the animation loop

self.loading\_frame.destroy()

class InventoryApp(customtkinter.CTk):

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

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

self.geometry("1000x700")

self.title("Inventory Management System")

self.resizable(False, False) # Disable maximize button on the login screen

customtkinter.set\_appearance\_mode("light")

customtkinter.set\_default\_color\_theme("blue")

self.current\_user = None

self.current\_user\_role = None

self.nav\_frame = None

self.is\_full\_screen = False

# Play startup jingle

self.play\_audio(r"Module 8\STABLE (TESTING)\Sales\_Inventory\_App\test.mp3")

# Apply the border with glow effect

self.configure(bg="#1abc9c") # Border color similar to login screen

self.update\_idletasks()

# Set the app to be on top at startup

self.lift() # Raise the window

self.attributes('-topmost', True) # Ensure the window stays on top

self.after(1000, lambda: self.attributes('-topmost', False)) # Remove the "always on top" after 1 second

self.show\_loading\_screen()

def show\_loading\_screen(self):

"""Show a loading screen while the app initializes."""

self.loading\_screen = LoadingScreen(self)

# Simulate initialization tasks in a separate thread

threading.Thread(target=self.initialize\_app).start()

def initialize\_app(self):

"""Simulate initialization tasks."""

for \_ in range(3): # Simulate a 4-second task in 1-second chunks

time.sleep(1) # Simulate delay

self.after(0, self.update\_loading\_message, "Still loading...")

# Complete initialization

self.after(0, self.complete\_initialization)

def update\_loading\_message(self, message):

"""Update the loading screen message."""

pass

def complete\_initialization(self):

"""Complete the app initialization and show the main screen."""

self.show\_login\_screen()

self.loading\_screen.destroy()

def play\_audio(self, file\_path):

"""Play an audio file using pygame."""

try:

pygame.mixer.music.load(r'static/audio/login.mp3')

pygame.mixer.music.play()

except Exception as e:

print(f"Error playing audio file: {e}")

def on\_close(self):

# Play logout sound before closing

pygame.mixer.music.load(r'static/audio/logout.mp3')

pygame.mixer.music.play()

self.after(1000, self.destroy)

def show\_login\_screen(self):

self.geometry("398x648")

self.resizable(False, False) # Keep maximize button disabled on the login screen

self.clear\_frames()

LoginScreen(self, self.login\_success\_callback)

def login\_success\_callback(self, username, role):

print(f"Logged in as {username} with role {role}")

self.current\_user = username

self.current\_user\_role = role

# Remove `overrideredirect` flag before moving to the next screen

self.overrideredirect(False)

# Show the dashboard screen after successful login

self.show\_dashboard\_screen()

def show\_dashboard\_screen(self):

self.geometry("1000x700") # Resize window for the dashboard

self.resizable(True, True) # Enable maximize button after login

self.clear\_frames(exclude\_nav=True)

self.create\_or\_update\_navigation\_pane()

DashboardScreen(self)

def show\_manage\_screen(self):

self.clear\_frames(exclude\_nav=True)

self.create\_or\_update\_navigation\_pane()

ManageInventoryScreen(self, backend, self.current\_user\_role, self.current\_user) # Pass username here

def show\_profile\_screen(self):

self.clear\_frames(exclude\_nav=True)

self.create\_or\_update\_navigation\_pane()

ProfileScreen(self, self.current\_user, self.current\_user\_role)

def show\_settings\_screen(self):

self.clear\_frames(exclude\_nav=True)

self.create\_or\_update\_navigation\_pane()

SettingsScreen(self, self.current\_user\_role, backend)

def show\_manage\_staff\_screen(self):

self.clear\_frames(exclude\_nav=True)

self.create\_or\_update\_navigation\_pane()

ManageStaffScreen(self, backend, self.current\_user\_role)

def refresh\_styles(self):

"""Refresh styles for all components after theme change."""

if hasattr(self, 'nav\_frame') and isinstance(self.nav\_frame, CreateNavigationPane):

self.nav\_frame.refresh\_styles()

def show\_terminate\_screen(self):

if self.current\_user\_role == 'Supervisor':

self.clear\_frames(exclude\_nav=True)

self.create\_or\_update\_navigation\_pane()

TerminateUserScreen(self, backend)

def show\_generate\_report\_screen(self):

# Launch the GenerateReportScreen module

report\_screen = GenerateReportPopup(self)

report\_screen.mainloop() # Start the main loop for the report screen

print("Reports screen launched successfully.")

def clear\_frames(self, exclude\_nav=False):

# Clear all frames except navigation if specified

for widget in self.winfo\_children():

if exclude\_nav and self.nav\_frame and widget == self.nav\_frame:

continue

if str(widget) in self.tk.call('winfo', 'children', self):

try:

widget.destroy()

except Exception as e:

print(f"Error destroying widget: {e}")

if not exclude\_nav and self.nav\_frame:

try:

self.nav\_frame.destroy()

self.nav\_frame = None

except Exception as e:

print(f"Error destroying nav\_frame: {e}")

def create\_or\_update\_navigation\_pane(self):

# Destroy the existing nav\_frame before creating a new one

if self.nav\_frame:

try:

self.nav\_frame.destroy()

except Exception as e:

print(f"Error destroying nav\_frame: {e}")

# Prepare parameters based on user role

if self.current\_user\_role == "Supervisor":

show\_manage\_staff\_screen = self.show\_manage\_staff\_screen

show\_generate\_report\_screen = self.show\_generate\_report\_screen

else:

show\_manage\_staff\_screen = None

show\_generate\_report\_screen = None

# Instantiate the navigation pane with the correct parameters

self.nav\_frame = CreateNavigationPane(

self,

self.show\_dashboard\_screen,

self.show\_manage\_screen,

self.show\_profile\_screen,

self.show\_settings\_screen,

show\_manage\_staff\_screen=show\_manage\_staff\_screen,

show\_generate\_report\_screen=show\_generate\_report\_screen

)

def toggle\_fullscreen(self):

# Toggle between fullscreen and windowed mode

if self.is\_full\_screen:

self.attributes("-fullscreen", False)

self.is\_full\_screen = False

else:

self.attributes("-fullscreen", True)

self.is\_full\_screen = True

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

app = InventoryApp()

app.protocol("WM\_DELETE\_WINDOW", app.on\_close) # Set up proper window close handling

app.mainloop()

Setup.py

from setuptools import setup, find\_packages

setup(

name="SalesInventoryApp",

version="1.0.0",

description="Sales Inventory Management Application",

author="Your Name",

packages=find\_packages(include=["app", "app.\*"]),

include\_package\_data=True,

install\_requires=[

"customtkinter",

"pygame",

"pillow"

],

entry\_points={

"console\_scripts": [

"sales\_inventory\_app=main:main",

],

},

)