import json

from tkinter import \*

from tkinter import messagebox, ttk

from tkinter import scrolledtext

from PIL import Image, ImageTk

import os

from datetime import datetime

# Create the main application window

win = Tk()

win.title("MVP Exterior Cleaning App")

win.geometry("1200x800")

win.configure(bg="#1e1e1e")

# Set a modern and attractive style for all UI components

style = ttk.Style()

style.theme\_use('clam')

style.configure('TButton', font=('Helvetica', 14), background='#DAA520', foreground='black')

style.configure('TLabel', background='#1e1e1e', foreground='gold', font=('Helvetica', 12))

style.configure('TEntry', font=('Helvetica', 12))

style.configure('TCombobox', font=('Helvetica', 12))

# File to store job and client information

jobs\_file = "jobs.json"

clients\_file = "clients.json"

# Load data from the JSON file

def load\_data(file\_path):

if os.path.exists(file\_path):

try:

with open(file\_path, 'r') as file:

return json.load(file)

except json.JSONDecodeError:

return []

return []

# Save data to the JSON file

def save\_data(file\_path, data):

with open(file\_path, 'w') as file:

json.dump(data, file)

# Load jobs and clients from files

jobs = load\_data(jobs\_file)

clients = load\_data(clients\_file)

# Function to create the login screen

def create\_login\_ui():

login\_frame = Frame(win, bg="#1e1e1e")

login\_frame.pack(expand=True)

# Create and place the logo

try:

logo\_image = Image.open("/Users/taylorscott/Desktop/MVPcrm/IMG\_6973.png")

logo\_image = logo\_image.resize((750, 375), Image.LANCZOS)

logo\_photo = ImageTk.PhotoImage(logo\_image)

logo\_label = Label(login\_frame, image=logo\_photo, bg="#1e1e1e")

logo\_label.image = logo\_photo

logo\_label.pack(pady=(20, 10))

except Exception as e:

print(f"Error loading image: {e}")

# Username entry

Label(login\_frame, text="Username:", bg="#1e1e1e", fg="gold", font=('Helvetica', 16, 'bold')).pack(pady=5)

username\_entry = Entry(login\_frame, width=30, font=('Helvetica', 14), highlightbackground="#DAA520", highlightthickness=2)

username\_entry.pack(pady=5)

# Password entry

Label(login\_frame, text="Password:", bg="#1e1e1e", fg="gold", font=('Helvetica', 16, 'bold')).pack(pady=5)

password\_entry = Entry(login\_frame, show='\*', width=30, font=('Helvetica', 14), highlightbackground="#DAA520", highlightthickness=2)

password\_entry.pack(pady=5)

# Login button

login\_button = Button(login\_frame, text="Login", command=lambda: login(username\_entry.get(), password\_entry.get()),

bg="#DAA520", fg="black", font=('Helvetica', 16, 'bold'), relief='flat', cursor='hand2')

login\_button.pack(pady=20)

# Function to handle login

def login(username, password):

if username == "1" and password == "1": # Temporary login for easier testing

messagebox.showinfo("Login Successful", "Welcome!")

create\_ui()

else:

messagebox.showerror("Login Failed", "Invalid username or password.")

# Function to create the main UI with navigation bar

def create\_ui():

for widget in win.winfo\_children():

widget.destroy()

# Navigation bar

nav\_bar = Frame(win, bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

nav\_bar.pack(side=TOP, fill=X)

button\_style = {

"bg": "#DAA520",

"fg": "black",

"font": ('Helvetica', 14, 'bold'),

"padx": 20,

"pady": 10,

"relief": "flat",

"cursor": "hand2"

}

buttons = [

("Dashboard", view\_dashboard),

("New Job", open\_new\_job\_window),

("All Jobs", view\_all\_jobs),

("New Client", open\_new\_client\_window),

("All Clients", view\_all\_clients),

("Service Statistics", view\_service\_statistics),

("Reminders", view\_reminders),

("Settings", open\_settings\_window),

("Quit", win.destroy)

]

for text, command in buttons:

Button(nav\_bar, text=text, command=command, \*\*button\_style).pack(side=LEFT, padx=10)

# Logo

try:

logo\_image = Image.open("/Users/taylorscott/Desktop/MVPcrm/IMG\_6973.png")

logo\_image = logo\_image.resize((250, 125), Image.LANCZOS)

logo\_photo = ImageTk.PhotoImage(logo\_image)

logo\_label = Label(win, image=logo\_photo, bg="#1e1e1e")

logo\_label.image = logo\_photo

logo\_label.pack(pady=(20, 10))

except Exception as e:

print(f"Error loading image: {e}")

# Function to view all clients

def view\_all\_clients():

for widget in win.winfo\_children():

widget.destroy()

create\_ui()

all\_clients\_frame = Frame(win, bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

all\_clients\_frame.pack(expand=True, fill=BOTH)

Label(all\_clients\_frame, text="All Clients", font=('Helvetica', 20, 'bold'), bg="#2b2b2b", fg="gold").pack(pady=10)

# Search bar

search\_frame = Frame(all\_clients\_frame, bg="#2b2b2b")

search\_frame.pack(pady=10)

search\_label = Label(search\_frame, text="Search Clients:", font=('Helvetica', 14), bg="#2b2b2b", fg="gold")

search\_label.pack(side=LEFT, padx=5)

search\_var = StringVar()

search\_entry = Entry(search\_frame, textvariable=search\_var, font=('Helvetica', 14), width=30, highlightbackground='#DAA520', highlightthickness=2)

search\_entry.pack(side=LEFT, padx=5)

def search\_clients():

query = search\_var.get().lower()

filtered\_clients = [client for client in clients if query in client['Name'].lower() or query in client['Phone'] or query in client['Email']]

display\_clients(filtered\_clients)

search\_button = Button(search\_frame, text="Search", command=search\_clients, bg="#DAA520", fg="black", font=('Helvetica', 14, 'bold'), relief='flat', cursor='hand2')

search\_button.pack(side=LEFT, padx=5)

# Display clients

def display\_clients(client\_list):

for widget in all\_clients\_frame.winfo\_children():

if isinstance(widget, Frame) and widget != search\_frame:

widget.destroy()

for client in client\_list:

client\_frame = Frame(all\_clients\_frame, bg="#3a3a3a", padx=10, pady=10)

client\_frame.pack(pady=5, fill=X)

client\_label = Label(client\_frame, text=f"{client['Name']} - {client['Phone']} - {client['Email']}", font=('Helvetica', 14), bg="#3a3a3a", fg="white")

client\_label.pack(side=LEFT, padx=5)

view\_button = Button(client\_frame, text="View Profile", command=lambda c=client: view\_client\_profile(c), bg="#DAA520", fg="black", font=('Helvetica', 12, 'bold'), relief='flat', cursor='hand2')

view\_button.pack(side=RIGHT, padx=5)

# Show all clients initially

display\_clients(clients)

# Function to view a detailed client profile

def view\_client\_profile(client):

for widget in win.winfo\_children():

widget.destroy()

create\_ui()

client\_profile\_frame = Frame(win, bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

client\_profile\_frame.pack(expand=True, fill=BOTH)

Label(client\_profile\_frame, text=f"Client Profile: {client['Name']}", font=('Helvetica', 20, 'bold'), bg="#2b2b2b", fg="gold").pack(pady=10)

# Client details

details\_frame = Frame(client\_profile\_frame, bg="#2b2b2b")

details\_frame.pack(pady=10, padx=20, fill=BOTH)

details = [

("Name", client['Name']),

("Phone", client['Phone']),

("Email", client['Email']),

("Address", client.get('Address', 'n/a')),

("Notes", client.get('Notes', 'n/a'))

]

for label, value in details:

Label(details\_frame, text=f"{label}:", bg="#2b2b2b", fg="gold", font=('Helvetica', 16, 'bold')).pack(anchor=W)

Label(details\_frame, text=value, bg="#2b2b2b", fg="white", font=('Helvetica', 14)).pack(anchor=W, padx=10)

# Job history

Label(client\_profile\_frame, text="Job History", font=('Helvetica', 18, 'bold'), bg="#2b2b2b", fg="gold").pack(pady=10)

job\_history\_frame = Frame(client\_profile\_frame, bg="#2b2b2b")

job\_history\_frame.pack(pady=10, padx=20, fill=BOTH)

client\_jobs = [job for job in jobs if job['Client Name'] == client['Name']]

if client\_jobs:

for job in client\_jobs:

job\_label = Button(job\_history\_frame, text=f"{job['Name']} - {job['Scheduled Date']} - {job['Status']}",

font=('Helvetica', 14), bg="#DAA520", fg="black", padx=10, pady=5, relief='flat', cursor='hand2',

command=lambda j=job: edit\_or\_delete\_job(j))

job\_label.pack(fill=X, pady=2)

else:

Label(job\_history\_frame, text="No job history available.", font=('Helvetica', 14), bg="#2b2b2b", fg="white").pack()

# Delete client button

delete\_button = Button(client\_profile\_frame, text="Delete Client", command=lambda: confirm\_delete\_client(client),

bg="#FF6347", fg="black", font=('Helvetica', 14, 'bold'), relief='flat', cursor='hand2')

delete\_button.pack(pady=10)

# Back button

back\_button = Button(client\_profile\_frame, text="Back to All Clients", command=view\_all\_clients, bg="#DAA520", fg="black", font=('Helvetica', 14, 'bold'), relief='flat', cursor='hand2')

back\_button.pack(pady=20)

# Function to confirm delete client with password

def confirm\_delete\_client(client):

password\_window = Toplevel(win)

password\_window.title("Confirm Client Deletion")

password\_window.geometry("300x200")

password\_window.configure(bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

Label(password\_window, text="Enter password to delete client:", bg="#2b2b2b", fg="gold", font=('Helvetica', 16)).pack(pady=10)

password\_entry = Entry(password\_window, show='\*', width=20, font=('Helvetica', 14), highlightbackground='#DAA520', highlightthickness=2)

password\_entry.pack(pady=5)

def delete\_client():

entered\_password = password\_entry.get()

if entered\_password == "1":

clients.remove(client)

save\_data(clients\_file, clients)

password\_window.destroy()

messagebox.showinfo("Deleted", "Client deleted successfully.")

view\_all\_clients()

else:

messagebox.showerror("Error", "Incorrect password.")

confirm\_button = Button(password\_window, text="Confirm Delete", bg="#FF6347", fg="black",

font=('Helvetica', 14, 'bold'), command=delete\_client, relief='flat', cursor='hand2')

confirm\_button.pack(pady=10)

# Function to view reminders

def view\_reminders():

for widget in win.winfo\_children():

widget.destroy()

create\_ui()

reminders\_frame = Frame(win, bg="grey20")

reminders\_frame.pack(expand=True, fill=BOTH)

Label(reminders\_frame, text="Reminders", font=('Helvetica', 16, 'bold'), bg="grey20", fg="gold").pack(pady=10)

reminder\_interval = int(settings.get("reminder\_interval", 7))

upcoming\_jobs = [job for job in jobs if (datetime.strptime(job['Scheduled Date'], "%m/%d/%Y") - datetime.now()).days <= reminder\_interval]

if not upcoming\_jobs:

Label(reminders\_frame, text="No upcoming reminders.", font=('Helvetica', 12), bg="grey20", fg="white").pack(pady=10)

else:

for job in upcoming\_jobs:

Label(reminders\_frame, text=f"Job: {job['Name']} on {job['Scheduled Date']}", font=('Helvetica', 12), bg="grey20", fg="white").pack(pady=5)

# Function to view the Dashboard

def view\_dashboard():

for widget in win.winfo\_children():

widget.destroy()

create\_ui()

dashboard\_frame = Frame(win, bg="#1e1e1e", highlightbackground="#DAA520", highlightthickness=2)

dashboard\_frame.pack(expand=True, fill=BOTH)

# Total Jobs Completed

total\_jobs = len(jobs)

Label(dashboard\_frame, text=f"Total Jobs: {total\_jobs}", font=('Helvetica', 20, 'bold'), bg="#1e1e1e", fg="gold").pack(pady=10)

# Monthly Revenue (Sum of all job prices)

monthly\_revenue = sum(float(job.get('Price', 0)) for job in jobs if job.get('Status') == "Completed")

Label(dashboard\_frame, text=f"Monthly Revenue: ${monthly\_revenue:.2f}", font=('Helvetica', 20, 'bold'), bg="#1e1e1e", fg="gold").pack(pady=10)

# Most Popular Services

service\_count = {}

for job in jobs:

for service in job['Services']:

service\_name = service['Service']

service\_count[service\_name] = service\_count.get(service\_name, 0) + 1

if service\_count:

popular\_service = max(service\_count, key=service\_count.get)

Label(dashboard\_frame, text=f"Most Popular Service: {popular\_service}", font=('Helvetica', 20, 'bold'), bg="#1e1e1e", fg="gold").pack(pady=10)

else:

Label(dashboard\_frame, text="Most Popular Service: N/A", font=('Helvetica', 20, 'bold'), bg="#1e1e1e", fg="gold").pack(pady=10)

# Function to view service statistics

def view\_service\_statistics():

for widget in win.winfo\_children():

widget.destroy()

create\_ui()

service\_stats\_frame = Frame(win, bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

service\_stats\_frame.pack(expand=True, fill=BOTH)

Label(service\_stats\_frame, text="Service Statistics", font=('Helvetica', 20, 'bold'), bg="#2b2b2b", fg="gold").pack(pady=10)

# Group jobs by year and month

jobs\_by\_year\_month = {}

for job in jobs:

date = datetime.strptime(job['Scheduled Date'], "%m/%d/%Y")

year\_month = (date.year, date.strftime("%B"))

if year\_month not in jobs\_by\_year\_month:

jobs\_by\_year\_month[year\_month] = []

jobs\_by\_year\_month[year\_month].append(job)

# Display statistics by year and month

for (year, month), month\_jobs in sorted(jobs\_by\_year\_month.items()):

month\_frame = Frame(service\_stats\_frame, bg="#3a3a3a", padx=10, pady=10)

month\_frame.pack(pady=5, fill=X)

Label(month\_frame, text=f"{month} {year}", font=('Helvetica', 18, 'bold'), bg="#3a3a3a", fg="gold").pack(anchor=W)

service\_earnings = {}

total\_earnings = 0

for job in month\_jobs:

for service in job['Services']:

service\_name = service['Service']

service\_price = float(service['Price'])

service\_earnings[service\_name] = service\_earnings.get(service\_name, 0) + service\_price

total\_earnings += service\_price

stats\_frame = Frame(month\_frame, bg="#2b2b2b")

stats\_frame.pack(pady=5, padx=20, fill=X)

Label(stats\_frame, text="Service", font=('Helvetica', 14, 'bold'), bg="#2b2b2b", fg="gold").grid(row=0, column=0, padx=10, pady=5, sticky=W)

Label(stats\_frame, text="Earnings ($)", font=('Helvetica', 14, 'bold'), bg="#2b2b2b", fg="gold").grid(row=0, column=1, padx=10, pady=5, sticky=W)

row = 1

for service, earnings in service\_earnings.items():

Label(stats\_frame, text=service, font=('Helvetica', 12), bg="#2b2b2b", fg="white").grid(row=row, column=0, padx=10, pady=5, sticky=W)

Label(stats\_frame, text=f"${earnings:.2f}", font=('Helvetica', 12), bg="#2b2b2b", fg="white").grid(row=row, column=1, padx=10, pady=5, sticky=W)

row += 1

Label(stats\_frame, text="Total Earnings", font=('Helvetica', 14, 'bold'), bg="#2b2b2b", fg="gold").grid(row=row, column=0, padx=10, pady=10, sticky=W)

Label(stats\_frame, text=f"${total\_earnings:.2f}", font=('Helvetica', 14, 'bold'), bg="#2b2b2b", fg="gold").grid(row=row, column=1, padx=10, pady=10, sticky=W)

# Function to open settings window

def open\_settings\_window():

settings\_window = Toplevel(win)

settings\_window.title("Settings")

settings\_window.geometry("400x300")

settings\_window.configure(bg="grey20")

label\_style = {'bg': 'grey20', 'fg': 'gold', 'font': ('Helvetica', 12, 'bold')}

entry\_style = {'font': ('Helvetica', 12), 'width': 35}

# Reminder interval

Label(settings\_window, text="Reminder Interval (days):", \*\*label\_style).pack(pady=5)

reminder\_interval\_entry = Entry(settings\_window, \*\*entry\_style)

reminder\_interval\_entry.pack(pady=5)

# Save button

save\_button = Button(settings\_window, text="Save Settings",

command=lambda: save\_settings(reminder\_interval\_entry.get()), bg="#DAA520", fg="black",

font=('Helvetica', 12, 'bold'))

save\_button.pack(pady=20)

# Function to save settings

def save\_settings(reminder\_interval):

settings = {

"reminder\_interval": reminder\_interval

}

save\_data('settings.json', settings)

messagebox.showinfo("Success", "Settings saved successfully!")

# Function to load settings

def load\_settings():

return load\_data('settings.json')

# Load settings

settings = load\_settings()

# Function to open the New Job window

def open\_new\_job\_window():

new\_job\_window = Toplevel(win)

new\_job\_window.title("New Job")

new\_job\_window.geometry("600x800")

new\_job\_window.configure(bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

label\_style = {'bg': '#2b2b2b', 'fg': 'gold', 'font': ('Helvetica', 14, 'bold')}

entry\_style = {'font': ('Helvetica', 14), 'width': 35, 'highlightbackground': '#DAA520', 'highlightthickness': 2}

# Job Name

Label(new\_job\_window, text="Job Name:", \*\*label\_style).pack(pady=5)

job\_name\_entry = Entry(new\_job\_window, \*\*entry\_style)

job\_name\_entry.pack(pady=5)

# Address

Label(new\_job\_window, text="Address:", \*\*label\_style).pack(pady=5)

address\_entry = Entry(new\_job\_window, \*\*entry\_style)

address\_entry.pack(pady=5)

# Client Selection or Entry

Label(new\_job\_window, text="Client Name (Select or Enter New):", \*\*label\_style).pack(pady=5)

client\_name\_var = StringVar()

client\_dropdown = ttk.Combobox(new\_job\_window, textvariable=client\_name\_var,

values=[client['Name'] for client in clients], width=33, font=('Helvetica', 14))

client\_dropdown.set('Select or enter a client')

client\_dropdown.pack(pady=5)

# Client Phone and Email

Label(new\_job\_window, text="Client Phone Number:", \*\*label\_style).pack(pady=5)

client\_phone\_entry = Entry(new\_job\_window, \*\*entry\_style)

client\_phone\_entry.pack(pady=5)

Label(new\_job\_window, text="Client Email Address:", \*\*label\_style).pack(pady=5)

client\_email\_entry = Entry(new\_job\_window, \*\*entry\_style)

client\_email\_entry.pack(pady=5)

# Job Status

Label(new\_job\_window, text="Job Status:", \*\*label\_style).pack(pady=5)

job\_status\_var = StringVar(value="Scheduled")

job\_status\_dropdown = ttk.Combobox(new\_job\_window, textvariable=job\_status\_var,

values=["Scheduled", "In Progress", "Completed"], width=33,

font=('Helvetica', 14))

job\_status\_dropdown.pack(pady=5)

# Services

services\_frame = Frame(new\_job\_window, bg="#2b2b2b")

services\_frame.pack(pady=5, padx=10)

Label(services\_frame, text="Services Requested:", \*\*label\_style).pack(pady=5)

services\_entries = []

def add\_service\_row():

service\_var = StringVar()

price\_var = StringVar()

service\_row\_frame = Frame(services\_frame, bg="#2b2b2b")

service\_row\_frame.pack(pady=5, padx=10, fill=X)

service\_dropdown = ttk.Combobox(service\_row\_frame, textvariable=service\_var,

values=["Surface Cleaning", "Exterior Window Cleaning", "Roof Wash",

"House Wash", "Gutter Cleaning", "Fence Cleaning"], width=20,

font=('Helvetica', 14))

service\_dropdown.pack(side=LEFT, padx=5)

price\_entry = Entry(service\_row\_frame, textvariable=price\_var, width=10, font=('Helvetica', 14), highlightbackground='#DAA520', highlightthickness=2)

price\_entry.pack(side=LEFT, padx=5)

services\_entries.append((service\_var, price\_var))

update\_total\_price()

add\_button = Button(services\_frame, text="+ Add Service", command=add\_service\_row, bg="#DAA520", fg="black",

font=('Helvetica', 14, 'bold'), relief='flat', cursor='hand2')

add\_button.pack(pady=5)

# Price Calculation

Label(new\_job\_window, text="Price Estimate ($):", \*\*label\_style).pack(pady=5)

price\_var = StringVar(value="0.00")

price\_entry = Entry(new\_job\_window, textvariable=price\_var, \*\*entry\_style)

price\_entry.pack(pady=5)

# Date Selection

Label(new\_job\_window, text="Scheduled Date:", \*\*label\_style).pack(pady=5)

date\_entry = Entry(new\_job\_window, \*\*entry\_style)

date\_entry.insert(0, datetime.now().strftime("%m/%d/%Y")) # Default to today's date

date\_entry.pack(pady=5)

# Additional Notes

Label(new\_job\_window, text="Additional Notes:", \*\*label\_style).pack(pady=5)

notes\_entry = scrolledtext.ScrolledText(new\_job\_window, wrap=WORD, width=50, height=5, font=('Helvetica', 14), highlightbackground='#DAA520', highlightthickness=2)

notes\_entry.pack(pady=5)

# Save button

save\_button = Button(new\_job\_window, text="Save Job",

command=lambda: save\_job(job\_name\_entry.get(), address\_entry.get(), client\_name\_var.get(),

client\_phone\_entry.get(), client\_email\_entry.get(),

job\_status\_var.get(), services\_entries, price\_var, date\_entry.get(),

notes\_entry.get("1.0", END).strip()),

bg="#DAA520", fg="black", font=('Helvetica', 14, 'bold'), relief='flat', cursor='hand2')

save\_button.pack(pady=20)

# Function to update the total price

def update\_total\_price():

total\_price = sum(float(price.get()) for service, price in services\_entries if price.get())

price\_var.set(f"{total\_price:.2f}")

# Function to save the job information

def save\_job(job\_name, address, client\_name, client\_phone, client\_email, job\_status, services\_entries, price\_var,

scheduled\_date, notes):

if job\_name and address and client\_name and client\_phone and client\_email:

services = [{"Service": service.get(), "Price": price.get()} for service, price in services\_entries if

service.get() and price.get()]

total\_price = sum(float(price.get()) for service, price in services\_entries if price.get())

job = {

"Name": job\_name,

"Address": address,

"Client Name": client\_name,

"Client Phone": client\_phone,

"Client Email": client\_email,

"Status": job\_status,

"Services": services,

"Price": f"{total\_price:.2f}",

"Scheduled Date": scheduled\_date,

"Notes": notes

}

jobs.append(job)

save\_data(jobs\_file, jobs)

messagebox.showinfo("Success", "Job saved successfully!")

else:

messagebox.showwarning("Input Error", "Please fill in all fields.")

# Function to view all jobs

def view\_all\_jobs():

for widget in win.winfo\_children():

widget.destroy()

create\_ui()

all\_jobs\_frame = Frame(win, bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

all\_jobs\_frame.pack(expand=True, fill=BOTH)

Label(all\_jobs\_frame, text="All Jobs", font=('Helvetica', 20, 'bold'), bg="#2b2b2b", fg="gold").pack(pady=10)

# Organize jobs by year and month

jobs\_by\_year\_month = {}

for job in jobs:

date = datetime.strptime(job['Scheduled Date'], "%m/%d/%Y")

year\_month = (date.year, date.strftime("%B"))

if year\_month not in jobs\_by\_year\_month:

jobs\_by\_year\_month[year\_month] = []

jobs\_by\_year\_month[year\_month].append(job)

# Create buttons for each year

years = sorted(set(year for year, \_ in jobs\_by\_year\_month.keys()))

def show\_months\_for\_year(selected\_year):

for widget in all\_jobs\_frame.winfo\_children():

if isinstance(widget, Button) and widget.cget("text") not in years:

widget.destroy()

months = sorted(set(month for year, month in jobs\_by\_year\_month.keys() if year == selected\_year))

for month in months:

Button(all\_jobs\_frame, text=month, font=('Helvetica', 16), bg="#DAA520", fg="black",

command=lambda m=month: show\_jobs\_for\_month(selected\_year, m), relief='flat', cursor='hand2').pack(pady=5)

def show\_jobs\_for\_month(selected\_year, selected\_month):

for widget in all\_jobs\_frame.winfo\_children():

if isinstance(widget, Button) and widget.cget("text") not in years + [selected\_month]:

widget.destroy()

jobs\_list = sorted(jobs\_by\_year\_month.get((selected\_year, selected\_month), []),

key=lambda j: datetime.strptime(j['Scheduled Date'], "%m/%d/%Y"))

for job in jobs\_list:

job\_label = Button(all\_jobs\_frame, text=f"{job['Name']} - {job['Client Name']} ({job['Status']})",

bg="#DAA520", fg="black", font=('Helvetica', 14), relief='flat', cursor='hand2', command=lambda j=job: edit\_or\_delete\_job(j))

job\_label.pack(pady=2)

for year in years:

Button(all\_jobs\_frame, text=str(year), font=('Helvetica', 16), bg="#DAA520", fg="black",

command=lambda y=year: show\_months\_for\_year(y), relief='flat', cursor='hand2').pack(pady=5)

# Function to edit or delete job information

def edit\_or\_delete\_job(job):

edit\_job\_window = Toplevel(win)

edit\_job\_window.title("Edit Job")

edit\_job\_window.geometry("500x600")

edit\_job\_window.configure(bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

Label(edit\_job\_window, text=f"Job: {job['Name']}", bg="#2b2b2b", fg="gold", font=('Helvetica', 20, 'bold')).pack(pady=10)

details\_frame = Frame(edit\_job\_window, bg="#2b2b2b")

details\_frame.pack(pady=10, padx=20, fill=BOTH)

details = [

("Address", job['Address']),

("Client Name", job['Client Name']),

("Client Phone", job['Client Phone']),

("Client Email", job['Client Email']),

("Status", job['Status']),

("Scheduled Date", job['Scheduled Date']),

("Price", f"${job['Price']}")

]

for label, value in details:

Label(details\_frame, text=f"{label}:", bg="#2b2b2b", fg="gold", font=('Helvetica', 16, 'bold')).pack(anchor=W)

Label(details\_frame, text=value, bg="#2b2b2b", fg="white", font=('Helvetica', 14)).pack(anchor=W, padx=10)

Label(details\_frame, text="Services Requested:", bg="#2b2b2b", fg="gold", font=('Helvetica', 16, 'bold')).pack(anchor=W, pady=(10, 0))

for service in job['Services']:

Label(details\_frame, text=f"{service['Service']} - ${service['Price']}", bg="#2b2b2b", fg="white", font=('Helvetica', 14)).pack(anchor=W, padx=10)

# Edit button

edit\_button = Button(edit\_job\_window, text="Edit Job", bg="#4682B4", fg="black", font=('Helvetica', 16, 'bold'),

command=lambda: open\_edit\_job\_window(job), relief='flat', cursor='hand2')

edit\_button.pack(pady=10)

# Delete button

delete\_button = Button(edit\_job\_window, text="Delete Job", bg="#FF6347", fg="black", font=('Helvetica', 16, 'bold'),

command=lambda: confirm\_delete(job), relief='flat', cursor='hand2')

delete\_button.pack(pady=10)

# Function to open the Edit Job window

def open\_edit\_job\_window(job):

edit\_window = Toplevel(win)

edit\_window.title("Edit Job Details")

edit\_window.geometry("600x800")

edit\_window.configure(bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

label\_style = {'bg': '#2b2b2b', 'fg': 'gold', 'font': ('Helvetica', 12, 'bold')}

entry\_style = {'font': ('Helvetica', 12), 'width': 35, 'highlightbackground': '#DAA520', 'highlightthickness': 2}

# Job Name

Label(edit\_window, text="Job Name:", \*\*label\_style).pack(pady=5)

job\_name\_entry = Entry(edit\_window, \*\*entry\_style)

job\_name\_entry.insert(0, job['Name'])

job\_name\_entry.pack(pady=5)

# Address

Label(edit\_window, text="Address:", \*\*label\_style).pack(pady=5)

address\_entry = Entry(edit\_window, \*\*entry\_style)

address\_entry.insert(0, job['Address'])

address\_entry.pack(pady=5)

# Client Phone

Label(edit\_window, text="Client Phone Number:", \*\*label\_style).pack(pady=5)

client\_phone\_entry = Entry(edit\_window, \*\*entry\_style)

client\_phone\_entry.insert(0, job['Client Phone'])

client\_phone\_entry.pack(pady=5)

# Client Email

Label(edit\_window, text="Client Email Address:", \*\*label\_style).pack(pady=5)

client\_email\_entry = Entry(edit\_window, \*\*entry\_style)

client\_email\_entry.insert(0, job['Client Email'])

client\_email\_entry.pack(pady=5)

# Job Status

Label(edit\_window, text="Job Status:", \*\*label\_style).pack(pady=5)

job\_status\_var = StringVar(value=job['Status'])

job\_status\_dropdown = ttk.Combobox(edit\_window, textvariable=job\_status\_var,

values=["Scheduled", "In Progress", "Completed"], width=33,

font=('Helvetica', 12))

job\_status\_dropdown.pack(pady=5)

# Services

services\_frame = Frame(edit\_window, bg="#2b2b2b")

services\_frame.pack(pady=5, padx=10)

Label(services\_frame, text="Services Requested:", \*\*label\_style).pack(pady=5)

services\_entries = []

for service in job['Services']:

service\_var = StringVar(value=service['Service'])

price\_var = StringVar(value=service['Price'])

service\_row\_frame = Frame(services\_frame, bg="#2b2b2b")

service\_row\_frame.pack(pady=5, padx=10, fill=X)

service\_dropdown = ttk.Combobox(service\_row\_frame, textvariable=service\_var,

values=["Surface Cleaning", "Exterior Window Cleaning", "Roof Wash",

"House Wash", "Gutter Cleaning", "Fence Cleaning"], width=20,

font=('Helvetica', 12))

service\_dropdown.pack(side=LEFT, padx=5)

price\_entry = Entry(service\_row\_frame, textvariable=price\_var, width=10, font=('Helvetica', 12))

price\_entry.pack(side=LEFT, padx=5)

services\_entries.append((service\_var, price\_var))

# Date Selection

Label(edit\_window, text="Scheduled Date:", \*\*label\_style).pack(pady=5)

date\_entry = Entry(edit\_window, \*\*entry\_style)

date\_entry.insert(0, job['Scheduled Date'])

date\_entry.pack(pady=5)

# Additional Notes

Label(edit\_window, text="Additional Notes:", \*\*label\_style).pack(pady=5)

notes\_entry = scrolledtext.ScrolledText(edit\_window, wrap=WORD, width=50, height=5, font=('Helvetica', 14), highlightbackground='#DAA520', highlightthickness=2)

notes\_entry.insert(1.0, job.get('Notes', ''))

notes\_entry.pack(pady=5)

# Save button

save\_button = Button(edit\_window, text="Save Changes",

command=lambda: save\_edited\_job(job, job\_name\_entry.get(), address\_entry.get(),

client\_phone\_entry.get(), client\_email\_entry.get(),

job\_status\_var.get(), services\_entries, date\_entry.get(),

notes\_entry.get("1.0", END).strip()),

bg="#DAA520", fg="black", font=('Helvetica', 12, 'bold'))

save\_button.pack(pady=20)

# Function to save the edited job information

def save\_edited\_job(job, job\_name, address, client\_phone, client\_email, job\_status, services\_entries, scheduled\_date, notes):

job['Name'] = job\_name

job['Address'] = address

job['Client Phone'] = client\_phone

job['Client Email'] = client\_email

job['Status'] = job\_status

job['Scheduled Date'] = scheduled\_date

job['Services'] = [{"Service": service.get(), "Price": price.get()} for service, price in services\_entries if

service.get() and price.get()]

job['Price'] = f"{sum(float(price.get()) for service, price in services\_entries if price.get()):.2f}"

job['Notes'] = notes

save\_data(jobs\_file, jobs)

messagebox.showinfo("Success", "Job details updated successfully!")

view\_all\_jobs()

# Function to confirm delete with password

def confirm\_delete(job):

password\_window = Toplevel(win)

password\_window.title("Confirm Deletion")

password\_window.geometry("300x200")

password\_window.configure(bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

Label(password\_window, text="Enter password to delete job:", bg="#2b2b2b", fg="gold", font=('Helvetica', 16)).pack(pady=10)

password\_entry = Entry(password\_window, show='\*', width=20, font=('Helvetica', 14), highlightbackground='#DAA520', highlightthickness=2)

password\_entry.pack(pady=5)

def delete\_job():

entered\_password = password\_entry.get()

if entered\_password == "1":

jobs.remove(job)

save\_data(jobs\_file, jobs)

password\_window.destroy()

messagebox.showinfo("Deleted", "Job deleted successfully.")

view\_all\_jobs()

else:

messagebox.showerror("Error", "Incorrect password.")

confirm\_button = Button(password\_window, text="Confirm Delete", bg="#FF6347", fg="black",

font=('Helvetica', 14, 'bold'), command=delete\_job, relief='flat', cursor='hand2')

confirm\_button.pack(pady=10)

# Function to open the New Client window

def open\_new\_client\_window():

new\_client\_window = Toplevel(win)

new\_client\_window.title("New Client")

new\_client\_window.geometry("400x400")

new\_client\_window.configure(bg="#2b2b2b", highlightbackground="#DAA520", highlightthickness=2)

label\_style = {'bg': '#2b2b2b', 'fg': 'gold', 'font': ('Helvetica', 16, 'bold')}

entry\_style = {'font': ('Helvetica', 14), 'width': 35, 'highlightbackground': '#DAA520', 'highlightthickness': 2}

# Client Name

Label(new\_client\_window, text="Client Name:", \*\*label\_style).pack(pady=5)

client\_name\_entry = Entry(new\_client\_window, \*\*entry\_style)

client\_name\_entry.pack(pady=5)

# Client Phone

Label(new\_client\_window, text="Client Phone Number:", \*\*label\_style).pack(pady=5)

client\_phone\_entry = Entry(new\_client\_window, \*\*entry\_style)

client\_phone\_entry.pack(pady=5)

# Client Email

Label(new\_client\_window, text="Client Email Address:", \*\*label\_style).pack(pady=5)

client\_email\_entry = Entry(new\_client\_window, \*\*entry\_style)

client\_email\_entry.pack(pady=5)

# Client Address

Label(new\_client\_window, text="Client Address:", \*\*label\_style).pack(pady=5)

client\_address\_entry = Entry(new\_client\_window, \*\*entry\_style)

client\_address\_entry.pack(pady=5)

# Additional Notes

Label(new\_client\_window, text="Additional Notes:", \*\*label\_style).pack(pady=5)

client\_notes\_entry = scrolledtext.ScrolledText(new\_client\_window, wrap=WORD, width=50, height=3, font=('Helvetica', 14), highlightbackground='#DAA520', highlightthickness=2)

client\_notes\_entry.pack(pady=5)

# Save button

save\_button = Button(new\_client\_window, text="Save Client",

command=lambda: save\_client(client\_name\_entry.get(), client\_phone\_entry.get(),

client\_email\_entry.get(), client\_address\_entry.get(),

client\_notes\_entry.get("1.0", END).strip()),

bg="#DAA520", fg="black", font=('Helvetica', 14, 'bold'), relief='flat', cursor='hand2')

save\_button.pack(pady=20)

# Function to save the client information

def save\_client(client\_name, client\_phone, client\_email, client\_address, client\_notes):

if client\_name and client\_phone and client\_email:

client = {

"Name": client\_name,

"Phone": client\_phone,

"Email": client\_email,

"Address": client\_address,

"Notes": client\_notes

}

clients.append(client)

save\_data(clients\_file, clients)

messagebox.showinfo("Success", "Client saved successfully!")

else:

messagebox.showwarning("Input Error", "Please fill in all required fields.")

# Run the login UI function

create\_login\_ui()

# Start the main loop

win.mainloop()