* **Adenilton Ferreira**

Saturday 8:27 PM

em baixo de qual linha de codigo devo colocar este trecho?: self.top\_level.mainloop()

* **OpenAI GPT4**

Saturday 8:28 PM

Peço desculpas pela confusão anterior. Você não deve adicionar **self.top\_level.mainloop()** ao **\_\_init\_\_** da classe **TaskWindow**. A chamada **mainloop()** já é feita para a instância principal da aplicação e não deve ser chamada novamente para as janelas toplevel. O uso da chamada **mainloop()** em mais de um local pode causar comportamentos inesperados e não é a maneira correta de manter a janela aberta.

A solução correta é usar o método **grab\_set**, como mencionei anteriormente. Você deve adicionar **self.top\_level.grab\_set()** logo após a criação e configuração dos widgets da janela **TaskWindow**. Aqui está onde você deve adicionar:

Python

*# Task Window Class for Add/Update Task*

class TaskWindow:

def \_\_init\_\_(self, parent, title, task=None, save\_path="", update=False):

self.top\_level = ctk.CTkToplevel(parent)

self.top\_level.title(title)

self.save\_path = save\_path

self.update = update

*# ... código para criar form fields ...*

*# ... código para criar Add or Update button ...*

*# Dá foco exclusivo à janela TaskWindow*

self.top\_level.grab\_set()

*# ... restante do código da classe TaskWindow ...*

Isso fará com que a janela **TaskWindow** mantenha o foco e não permita que o usuário interaja com a janela principal enquanto a **TaskWindow** estiver aberta, o que é o comportamento esperado em muitos diálogos modais.

Além disso, para garantir que a janela **TaskWindow** seja destruída corretamente quando o usuário fechar a janela ou clicar no botão de adicionar/atualizar, você deve garantir que o método **destroy()** seja chamado corretamente. Isso já está sendo feito no seu código no método **save\_task** com **self.top\_level.destroy()**.

Essas mudanças devem resolver o problema da janela que fecha imediatamente.

* 

**Adenilton Ferreira**

Saturday 8:46 PM

Testando a funcionalidade dos botões observei o seguinte, o botão “ adicionar tarefas” está funcionando perfeitamente consigo adicionar as tarefas ela é salva com todas as informações e está muito bem o botão também “ excluir tarefas” está funcionando perfeitamente sem nenhuma falha.

porém o botão “ atualizar tarefas” já não está funcionando perfeitamente, quando eu clico nele abre um pop-up para que eu escolha qual seria a tarefa que eu deveria adicionar, uma vez tendo escolhido Qual é o ID da tarefa que eu quero atualizar ele abre um outro pop-up porém não é da forma como eu gostaria que fosse, ele deveria abrir um formulário idêntico aquele que é aberto para adicionar tarefas porém agora com a intenção de trazendo o documento .json todas as informações já presente no formulário e ele vai fazer isso baseado no ID da tarefa que eu coloquei Que deve ser o mesmo tipo de ID que o documento .json está salvando, uma vez ou fazendo Qualquer mudança nesse formulário ele vai salvar o mesmo documento .json porém agora com as informações atualizadas.

verifica isso para mim por favor.

aqui embaixo eu recebi esse erros:

PS C:\Users\AFERR136\OneDrive - azureford\DEV PYTHON\MAPEAMENTO DB2> python web.py Exception in Tkinter callback Traceback (most recent call last): File "C:\Users\AFERR136\AppData\Local\Programs\Python\Python312\Lib\tkinter\_*init*\_.py", line 1948, in **call** return self.func(\*args) ^^^^^^^^^^^^^^^^ File "C:\Users\AFERR136\AppData\Local\Programs\Python\Python312\Lib\site-packages\customtkinter\windows\widgets\ctk\_button.py", line 554, in \_clicked self.\_command() File "C:\Users\AFERR136\OneDrive - azureford\DEV PYTHON\MAPEAMENTO DB2\web.py", line 153, in update\_task task\_window = TaskWindow(self, title="Atualizar Tarefa", task=task, save\_path=json\_save\_path, update=True) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\AFERR136\OneDrive - azureford\DEV PYTHON\MAPEAMENTO DB2\web.py", line 199, in **init** self.fields[field].set\_text(getattr(task, field)) ^^^^^^^^^^^^^^^^^^^^^^^^^^^ AttributeError: 'CTkEntry' object has no attribute 'set\_text'

E aqui abaixo está o código completo: import tkinter as tk from tkinter import messagebox from tkinter import simpledialog import customtkinter as ctk import json import os

**Path to save JSON files**

json\_save\_path = "C:/Users/AFERR136/OneDrive - azureford/DEV PYTHON/MAPEAMENTO DB2/BACKUP-2"

**Check if the path exists and create if it doesn't**

if not os.path.exists(json\_save\_path): os.makedirs(json\_save\_path)

**Task Class**

class Task: def **init**(self, task\_id, responsible, name, start\_date, end\_date, completed): self.task\_id = task\_id self.responsible = responsible self.name = name self.start\_date = start\_date self.end\_date = end\_date self.completed = completed

def to\_dict(self):

return {

"task\_id": self.task\_id,

"responsible": self.responsible,

"name": self.name,

"start\_date": self.start\_date,

"end\_date": self.end\_date,

"completed": self.completed

}

@staticmethod

def save\_to\_json(task, path):

with open(os.path.join(path, f"{task.task\_id}.json"), 'w') as f:

json.dump(task.to\_dict(), f, indent=4)

@staticmethod

def load\_from\_json(task\_id, path):

try:

with open(os.path.join(path, f"{task\_id}.json"), 'r') as f:

data = json.load(f)

return Task(\*\*data)

except FileNotFoundError:

return None

@staticmethod

def delete\_json(task\_id, path):

os.remove(os.path.join(path, f"{task\_id}.json"))

**Main Application Class**

class MyApp(ctk.CTk): def **init**(self): super().**init**()

self.title("MY DAILY TASKS")

self.geometry("700x500")

self.minsize(700, 500)

# Left frame with buttons

self.frame\_buttons = ctk.CTkFrame(self, width=200, height=500)

self.frame\_buttons.pack(side="left", fill="y", padx=10, pady=10)

self.btn\_add\_task = ctk.CTkButton(self.frame\_buttons, text="Adicionar tarefas", command=self.add\_task)

self.btn\_add\_task.pack(pady=10)

self.btn\_update\_task = ctk.CTkButton(self.frame\_buttons, text="Atualizar tarefas", command=self.update\_task)

self.btn\_update\_task.pack(pady=10)

self.btn\_delete\_task = ctk.CTkButton(self.frame\_buttons, text="Excluir tarefas", command=self.delete\_task)

self.btn\_delete\_task.pack(pady=10)

# Central frame with task table

self.frame\_table = ctk.CTkFrame(self, width=500, height=500)

self.frame\_table.pack(side="left", fill="both", expand=True, padx=10, pady=10)

# Table headers

self.table\_headers = [

"ID da tarefa", "Nome do responsável", "Nome da tarefa",

"Data de início", "Data de fim", "Status"

]

self.columns = len(self.table\_headers)

self.table = ctk.CTkFrame(self.frame\_table)

self.table.pack(fill="both", expand=True)

for i, header in enumerate(self.table\_headers):

label = ctk.CTkLabel(self.table, text=header, anchor="w", fg\_color=("gray75", "gray30"))

label.grid(row=0, column=i, sticky="nsew", padx=2, pady=2)

# Configure grid column weights

for i in range(self.columns):

self.table.grid\_columnconfigure(i, weight=1)

self.table.grid\_rowconfigure(0, weight=0) # Header row should not expand

# List to store current tasks

self.tasks = []

self.task\_labels = []

# Load existing tasks from JSON files

self.load\_tasks()

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing) # Handle window closing

def load\_tasks(self):

# Clear existing tasks and labels

for label in self.task\_labels:

label.destroy()

self.tasks.clear()

self.task\_labels.clear()

# Load tasks from JSON

for filename in os.listdir(json\_save\_path):

if filename.endswith(".json"):

task\_id = filename.split(".")[0]

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

self.tasks.append(task)

self.display\_task(task)

def display\_task(self, task):

row = len(self.tasks)

data = task.to\_dict()

for col, key in enumerate(data):

label = ctk.CTkLabel(self.table, text=str(data[key]), anchor="w")

label.grid(row=row, column=col, sticky="nsew", padx=2, pady=2)

self.task\_labels.append(label)

# Add status label with color

status\_label = ctk.CTkLabel(self.table, text=data["completed"], anchor="w", fg\_color=("green" if data["completed"] == "Concluído" else "red"))

status\_label.grid(row=row, column=self.columns-1, sticky="nsew", padx=2, pady=2)

self.task\_labels.append(status\_label)

# Update grid row weights

for i in range(1, len(self.tasks) + 1):

self.table.grid\_rowconfigure(i, weight=1)

def add\_task(self):

# Open new window to add task

task\_window = TaskWindow(self, title="Adicionar Tarefa", save\_path=json\_save\_path)

self.wait\_window(task\_window.top\_level)

# Reload tasks after adding

self.load\_tasks()

def update\_task(self):

# Open new window to update task

task\_id = simpledialog.askstring("Atualizar Tarefa", "Digite o ID da tarefa que deseja atualizar:")

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

task\_window = TaskWindow(self, title="Atualizar Tarefa", task=task, save\_path=json\_save\_path, update=True)

self.wait\_window(task\_window.top\_level)

# Reload tasks after updating

self.load\_tasks()

else:

messagebox.showerror("Erro", "Tarefa não encontrada.")

# Esta função deve estar no mesmo nível de indentação que as outras funções da classe MyApp

def delete\_task(self):

# Ask for task ID to delete

task\_id = simpledialog.askstring("Excluir Tarefa", "Digite o ID da tarefa que deseja excluir:")

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

confirm = messagebox.askyesno("Confirmar Exclusão", "Você tem certeza que deseja excluir esta tarefa?")

if confirm:

Task.delete\_json(task\_id, json\_save\_path)

messagebox.showinfo("Sucesso", "Tarefa excluída com sucesso.")

self.load\_tasks()

else:

messagebox.showerror("Erro", "Tarefa não encontrada.")

def on\_closing(self):

# Handle any cleanup here if necessary before closing

self.destroy()

**Task Window Class for Add/Update Task**

class TaskWindow: def **init**(self, parent, title, task=None, save\_path="", update=False): self.top\_level = ctk.CTkToplevel(parent) self.top\_level.title(title) self.save\_path = save\_path self.update = update

# If updating, preload the task info

if task:

self.task = task

else:

self.task = Task(task\_id="", responsible="", name="", start\_date="", end\_date="", completed="Não Concluído")

# Create form fields

self.fields = {}

for field in ["task\_id", "responsible", "name", "start\_date", "end\_date"]:

self.fields[field] = ctk.CTkEntry(self.top\_level, placeholder\_text=field.replace("\_", " ").title())

self.fields[field].pack(pady=5)

if task:

self.fields[field].set\_text(getattr(task, field))

self.fields["completed"] = ctk.CTkComboBox(self.top\_level, values=["Concluído", "Não Concluído"])

self.fields["completed"].set(self.task.completed)

self.fields["completed"].pack(pady=5)

# Add or Update button

action\_text = "Atualizar" if update else "Adicionar"

self.btn\_action = ctk.CTkButton(self.top\_level, text=action\_text, command=self.save\_task)

self.btn\_action.pack(pady=10)

self.top\_level.grab\_set()

def save\_task(self):

# Get data from form fields

data = {field: self.fields[field].get() for field in self.fields}

# Create Task object and save to JSON

self.task = Task(\*\*data)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", f"Tarefa {'atualizada' if self.update else 'adicionada'} com sucesso.")

self.top\_level.destroy()

**Run the application**

if **name** == "**main**": app = MyApp() app.mainloop()

* **OpenAI GPT4**

Saturday 8:47 PM

O erro que você está encontrando é devido ao método **set\_text** que não existe na classe **CTkEntry** da biblioteca **customtkinter**. Em **tkinter**, você normalmente usaria o método **insert** para adicionar texto a um campo de entrada ou um objeto **StringVar** associado para definir o valor. Em **customtkinter**, você deve usar o método **insert\_text** para adicionar texto a um campo de entrada **CTkEntry**.

Vamos corrigir o erro no método **\_\_init\_\_** da classe **TaskWindow**, substituindo **set\_text** por **insert\_text** e garantindo que o campo de entrada esteja limpo antes de inserir o novo texto. Além disso, como estamos lidando com a atualização, devemos desabilitar a edição do campo **task\_id**, pois ele serve como chave única para identificar a tarefa:

Python

*# Task Window Class for Add/Update Task*

class TaskWindow:

def \_\_init\_\_(self, parent, title, task=None, save\_path="", update=False):

self.top\_level = ctk.CTkToplevel(parent)

self.top\_level.title(title)

self.save\_path = save\_path

self.update = update

*# If updating, preload the task info*

if task:

self.task = task

else:

self.task = Task(task\_id="", responsible="", name="", start\_date="", end\_date="", completed="Não Concluído")

*# Create form fields*

self.fields = {}

for field in ["task\_id", "responsible", "name", "start\_date", "end\_date"]:

self.fields[field] = ctk.CTkEntry(self.top\_level, placeholder\_text=field.replace("\_", " ").title())

self.fields[field].pack(pady=5)

if task:

self.fields[field].delete(0, tk.END) *# Clear the current text*

self.fields[field].insert\_text(getattr(task, field)) *# Insert new text*

if field == "task\_id":

self.fields[field].configure(state='disabled') *# Disable editing of task\_id*

self.fields["completed"] = ctk.CTkComboBox(self.top\_level, values=["Concluído", "Não Concluído"])

self.fields["completed"].set(self.task.completed)

self.fields["completed"].pack(pady=5)

*# Add or Update button*

action\_text = "Atualizar" if update else "Adicionar"

self.btn\_action = ctk.CTkButton(self.top\_level, text=action\_text, command=self.save\_task)

self.btn\_action.pack(pady=10)

self.top\_level.grab\_set()

def save\_task(self):

*# Get data from form fields*

data = {field: self.fields[field].get() for field in self.fields if field != "task\_id"}

data["task\_id"] = self.task.task\_id *# Use the existing task\_id*

*# Update the Task object and save to JSON*

for key, value in data.items():

setattr(self.task, key, value)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", "Tarefa atualizada com sucesso.")

self.top\_level.destroy()

*# ... restante do código ...*

No método **save\_task**, agora atualizamos o objeto **task** existente com os novos valores dos campos de entrada e salvamos novamente no arquivo JSON correspondente. Note que não pegamos o valor de **task\_id** dos campos de entrada, uma vez que não permitimos a edição desse valor e usamos o ID da tarefa existente.

Com essas correções, o botão "Atualizar tarefas" deve funcionar conforme o esperado,

* 

**Adenilton Ferreira**

Saturday 8:57 PM

O erro continua ainda na hora de fazer atualização das tarefas quando eu clico no botão “ atualizar tarefas” aparece um pop-up pedindo para que eu coloque qual é o ID da tarefa, eu coloco o ID da tarefa e ele me retorna dizendo que não foi encontrado a tarefa.

eu fui então olhar o documento .json para ver se ele estava com o ID da tarefa e o que eu encontrei é isso abaixo:

{ "task\_id": "", "responsible": "Adenilton", "name": "Atualizar Formulario", "start\_date": "22/02/2024", "end\_date": "25/02/2024", "completed": "N\u00e3o Conclu\u00eddo" }

O campo "task\_id" está vazio ou seja o ID da tarefa que eu coloco não está sendo salvado dentro do .json e é por isso que o botão “ atualizar a tarefas” não está encontrando a tarefa

por favor conserte esse erro. Quando eu colocar na hora de adicionar a tarefa o Adir da tarefa ele tem que ser salvo junto com o .json porque ele será o referencial para depois excluir e também para depois atualizar.

Segue o código completo como estar até o presente momento: import tkinter as tk from tkinter import messagebox from tkinter import simpledialog import customtkinter as ctk import json import os

**Path to save JSON files**

json\_save\_path = "C:/Users/AFERR136/OneDrive - azureford/DEV PYTHON/MAPEAMENTO DB2/BACKUP-2"

**Check if the path exists and create if it doesn't**

if not os.path.exists(json\_save\_path): os.makedirs(json\_save\_path)

**Task Class**

class Task: def **init**(self, task\_id, responsible, name, start\_date, end\_date, completed): self.task\_id = task\_id self.responsible = responsible self.name = name self.start\_date = start\_date self.end\_date = end\_date self.completed = completed

def to\_dict(self):

return {

"task\_id": self.task\_id,

"responsible": self.responsible,

"name": self.name,

"start\_date": self.start\_date,

"end\_date": self.end\_date,

"completed": self.completed

}

@staticmethod

def save\_to\_json(task, path):

with open(os.path.join(path, f"{task.task\_id}.json"), 'w') as f:

json.dump(task.to\_dict(), f, indent=4)

@staticmethod

def load\_from\_json(task\_id, path):

try:

with open(os.path.join(path, f"{task\_id}.json"), 'r') as f:

data = json.load(f)

return Task(\*\*data)

except FileNotFoundError:

return None

@staticmethod

def delete\_json(task\_id, path):

os.remove(os.path.join(path, f"{task\_id}.json"))

**Main Application Class**

class MyApp(ctk.CTk): def **init**(self): super().**init**()

self.title("MY DAILY TASKS")

self.geometry("700x500")

self.minsize(700, 500)

# Left frame with buttons

self.frame\_buttons = ctk.CTkFrame(self, width=200, height=500)

self.frame\_buttons.pack(side="left", fill="y", padx=10, pady=10)

self.btn\_add\_task = ctk.CTkButton(self.frame\_buttons, text="Adicionar tarefas", command=self.add\_task)

self.btn\_add\_task.pack(pady=10)

self.btn\_update\_task = ctk.CTkButton(self.frame\_buttons, text="Atualizar tarefas", command=self.update\_task)

self.btn\_update\_task.pack(pady=10)

self.btn\_delete\_task = ctk.CTkButton(self.frame\_buttons, text="Excluir tarefas", command=self.delete\_task)

self.btn\_delete\_task.pack(pady=10)

# Central frame with task table

self.frame\_table = ctk.CTkFrame(self, width=500, height=500)

self.frame\_table.pack(side="left", fill="both", expand=True, padx=10, pady=10)

# Table headers

self.table\_headers = [

"ID da tarefa", "Nome do responsável", "Nome da tarefa",

"Data de início", "Data de fim", "Status"

]

self.columns = len(self.table\_headers)

self.table = ctk.CTkFrame(self.frame\_table)

self.table.pack(fill="both", expand=True)

for i, header in enumerate(self.table\_headers):

label = ctk.CTkLabel(self.table, text=header, anchor="w", fg\_color=("gray75", "gray30"))

label.grid(row=0, column=i, sticky="nsew", padx=2, pady=2)

# Configure grid column weights

for i in range(self.columns):

self.table.grid\_columnconfigure(i, weight=1)

self.table.grid\_rowconfigure(0, weight=0) # Header row should not expand

# List to store current tasks

self.tasks = []

self.task\_labels = []

# Load existing tasks from JSON files

self.load\_tasks()

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing) # Handle window closing

def load\_tasks(self):

# Clear existing tasks and labels

for label in self.task\_labels:

label.destroy()

self.tasks.clear()

self.task\_labels.clear()

# Load tasks from JSON

for filename in os.listdir(json\_save\_path):

if filename.endswith(".json"):

task\_id = filename.split(".")[0]

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

self.tasks.append(task)

self.display\_task(task)

def display\_task(self, task):

row = len(self.tasks)

data = task.to\_dict()

for col, key in enumerate(data):

label = ctk.CTkLabel(self.table, text=str(data[key]), anchor="w")

label.grid(row=row, column=col, sticky="nsew", padx=2, pady=2)

self.task\_labels.append(label)

# Add status label with color

status\_label = ctk.CTkLabel(self.table, text=data["completed"], anchor="w", fg\_color=("green" if data["completed"] == "Concluído" else "red"))

status\_label.grid(row=row, column=self.columns-1, sticky="nsew", padx=2, pady=2)

self.task\_labels.append(status\_label)

# Update grid row weights

for i in range(1, len(self.tasks) + 1):

self.table.grid\_rowconfigure(i, weight=1)

def add\_task(self):

# Open new window to add task

task\_window = TaskWindow(self, title="Adicionar Tarefa", save\_path=json\_save\_path)

self.wait\_window(task\_window.top\_level)

# Reload tasks after adding

self.load\_tasks()

def update\_task(self):

# Open new window to update task

task\_id = simpledialog.askstring("Atualizar Tarefa", "Digite o ID da tarefa que deseja atualizar:")

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

task\_window = TaskWindow(self, title="Atualizar Tarefa", task=task, save\_path=json\_save\_path, update=True)

self.wait\_window(task\_window.top\_level)

# Reload tasks after updating

self.load\_tasks()

else:

messagebox.showerror("Erro", "Tarefa não encontrada.")

# Esta função deve estar no mesmo nível de indentação que as outras funções da classe MyApp

def delete\_task(self):

# Ask for task ID to delete

task\_id = simpledialog.askstring("Excluir Tarefa", "Digite o ID da tarefa que deseja excluir:")

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

confirm = messagebox.askyesno("Confirmar Exclusão", "Você tem certeza que deseja excluir esta tarefa?")

if confirm:

Task.delete\_json(task\_id, json\_save\_path)

messagebox.showinfo("Sucesso", "Tarefa excluída com sucesso.")

self.load\_tasks()

else:

messagebox.showerror("Erro", "Tarefa não encontrada.")

def on\_closing(self):

# Handle any cleanup here if necessary before closing

self.destroy()

**Task Window Class for Add/Update Task**

class TaskWindow: def **init**(self, parent, title, task=None, save\_path="", update=False): self.top\_level = ctk.CTkToplevel(parent) self.top\_level.title(title) self.save\_path = save\_path self.update = update

# If updating, preload the task info

if task:

self.task = task

else:

self.task = Task(task\_id="", responsible="", name="", start\_date="", end\_date="", completed="Não Concluído")

# Create form fields

self.fields = {}

for field in ["task\_id", "responsible", "name", "start\_date", "end\_date"]:

self.fields[field] = ctk.CTkEntry(self.top\_level, placeholder\_text=field.replace("\_", " ").title())

self.fields[field].pack(pady=5)

if task:

self.fields[field].delete(0, tk.END) # Clear the current text

self.fields[field].insert\_text(getattr(task, field)) # Insert new text

if field == "task\_id":

self.fields[field].configure(state='disabled') # Disable editing of task\_id

self.fields["completed"] = ctk.CTkComboBox(self.top\_level, values=["Concluído", "Não Concluído"])

self.fields["completed"].set(self.task.completed)

self.fields["completed"].pack(pady=5)

# Add or Update button

action\_text = "Atualizar" if update else "Adicionar"

self.btn\_action = ctk.CTkButton(self.top\_level, text=action\_text, command=self.save\_task)

self.btn\_action.pack(pady=10)

self.top\_level.grab\_set()

def save\_task(self):

# Get data from form fields

data = {field: self.fields[field].get() for field in self.fields if field != "task\_id"}

data["task\_id"] = self.task.task\_id # Use the existing task\_id

# Update the Task object and save to JSON

for key, value in data.items():

setattr(self.task, key, value)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", "Tarefa atualizada com sucesso.")

self.top\_level.destroy()

**Run the application**

if **name** == "**main**": app = MyApp() app.mainloop()

* **OpenAI GPT4**

Saturday 8:57 PM

O problema que você está enfrentando está relacionado ao fato de que o ID da tarefa não está sendo definido corretamente ao adicionar uma nova tarefa. Isso pode estar acontecendo porque o campo **task\_id** não está sendo preenchido pelo usuário ou porque o valor não está sendo capturado corretamente ao salvar a tarefa.

Para corrigir esse problema, você precisa garantir que o **task\_id** seja solicitado e salvo corretamente ao adicionar uma nova tarefa. Vamos modificar o método **save\_task** na classe **TaskWindow** para garantir que o **task\_id** seja capturado e utilizado:

Python

class TaskWindow:

*# ... código anterior da classe TaskWindow ...*

def save\_task(self):

*# Get data from form fields*

data = {field: self.fields[field].get() for field in self.fields}

*# Se estiver atualizando, use o task\_id existente*

if self.update:

data["task\_id"] = self.task.task\_id

else:

*# Se for adicionar uma nova tarefa, garanta que o task\_id foi preenchido*

if not data["task\_id"]:

messagebox.showerror("Erro", "O ID da tarefa é obrigatório.")

return

*# Create Task object and save to JSON*

self.task = Task(\*\*data)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", f"Tarefa {'atualizada' if self.update else 'adicionada'} com sucesso.")

self.top\_level.destroy()

Dessa forma, ao adicionar uma nova tarefa, o sistema verificará se o campo **task\_id** foi preenchido. Se estiver vazio, será exibida uma mensagem de erro e a tarefa não será salva.

Além disso, vamos garantir que o campo **task\_id** seja habilitado para edição ao adicionar uma nova tarefa e desabilitado ao atualizar uma tarefa existente:

Python

class TaskWindow:

*# ... código anterior da classe TaskWindow ...*

def \_\_init\_\_(self, parent, title, task=None, save\_path="", update=False):

*# ... código anterior do método \_\_init\_\_ ...*

if update:

*# Se estiver atualizando, desabilite a edição do task\_id e preencha os campos*

self.fields["task\_id"].configure(state='disabled')

self.fields["task\_id"].insert\_text(self.task.task\_id)

for field in ["responsible", "name", "start\_date", "end\_date"]:

self.fields[field].delete(0, tk.END)

self.fields[field].insert\_text(getattr(task, field))

self.fields["completed"].set(self.task.completed)

else:

*# Se estiver adicionando, certifique-se de que todos os campos estão habilitados*

self.fields["task\_id"].configure(state='normal')

*# ... restante do código do método \_\_init\_\_ ...*

Essas alterações devem resolver o problema do campo **task\_id** vazio e garantir que as tarefas possam ser adicionadas e atualizadas corretamente.

* 

**Adenilton Ferreira**

Saturday 9:02 PM

Me dê por favor esse trecho do código abaixo completo com a modificação que você fez:

class TaskWindow: def **init**(self, parent, title, task=None, save\_path="", update=False): self.top\_level = ctk.CTkToplevel(parent) self.top\_level.title(title) self.save\_path = save\_path self.update = update

# If updating, preload the task info

if task:

self.task = task

else:

self.task = Task(task\_id="", responsible="", name="", start\_date="", end\_date="", completed="Não Concluído")

# Create form fields

self.fields = {}

for field in ["task\_id", "responsible", "name", "start\_date", "end\_date"]:

self.fields[field] = ctk.CTkEntry(self.top\_level, placeholder\_text=field.replace("\_", " ").title())

self.fields[field].pack(pady=5)

if task:

self.fields[field].delete(0, tk.END) # Clear the current text

self.fields[field].insert\_text(getattr(task, field)) # Insert new text

if field == "task\_id":

self.fields[field].configure(state='disabled') # Disable editing of task\_id

self.fields["completed"] = ctk.CTkComboBox(self.top\_level, values=["Concluído", "Não Concluído"])

self.fields["completed"].set(self.task.completed)

self.fields["completed"].pack(pady=5)

# Add or Update button

action\_text = "Atualizar" if update else "Adicionar"

self.btn\_action = ctk.CTkButton(self.top\_level, text=action\_text, command=self.save\_task)

self.btn\_action.pack(pady=10)

self.top\_level.grab\_set()

def save\_task(self):

# Get data from form fields

data = {field: self.fields[field].get() for field in self.fields}

# Se estiver atualizando, use o task\_id existente

if self.update:

data["task\_id"] = self.task.task\_id

else:

# Se for adicionar uma nova tarefa, garanta que o task\_id foi preenchido

if not data["task\_id"]:

messagebox.showerror("Erro", "O ID da tarefa é obrigatório.")

return

# Create Task object and save to JSON

self.task = Task(\*\*data)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", f"Tarefa {'atualizada' if self.update else 'adicionada'} com sucesso.")

self.top\_level.destroy()

Eu preciso desse trecho completo com todas as linhas escritas sem abreviação com a alteração que você faz para podermos salvar o ID da task de forma correta para que ele possa ser usado na hora de atualizar eu preciso documento pelo menos esse trecho todo completo para que eu possa apenas copiá-la

* **OpenAI GPT4**

Saturday 9:03 PM

Claro, aqui está o trecho completo da classe **TaskWindow** com as alterações necessárias para garantir que o **task\_id** seja corretamente preenchido ao adicionar uma nova tarefa e para prevenir a edição do **task\_id** ao atualizar uma tarefa existente:

Python

class TaskWindow:

def \_\_init\_\_(self, parent, title, task=None, save\_path="", update=False):

self.top\_level = ctk.CTkToplevel(parent)

self.top\_level.title(title)

self.save\_path = save\_path

self.update = update

*# If updating, preload the task info*

if task:

self.task = task

else:

self.task = Task(task\_id="", responsible="", name="", start\_date="", end\_date="", completed="Não Concluído")

*# Create form fields*

self.fields = {}

for field in ["task\_id", "responsible", "name", "start\_date", "end\_date"]:

self.fields[field] = ctk.CTkEntry(self.top\_level, placeholder\_text=field.replace("\_", " ").title())

self.fields[field].pack(pady=5)

if task: *# Pre-fill fields if updating*

self.fields[field].delete(0, tk.END) *# Clear the current text*

self.fields[field].insert(0, getattr(task, field)) *# Insert new text*

if field == "task\_id":

self.fields[field].configure(state='disabled') *# Disable editing of task\_id*

else: *# If adding, make sure the task\_id field is enabled*

self.fields[field].configure(state='normal')

self.fields["completed"] = ctk.CTkComboBox(self.top\_level, values=["Concluído", "Não Concluído"])

if task:

self.fields["completed"].set(self.task.completed)

else:

self.fields["completed"].set("Não Concluído")

self.fields["completed"].pack(pady=5)

*# Add or Update button*

action\_text = "Atualizar" if update else "Adicionar"

self.btn\_action = ctk.CTkButton(self.top\_level, text=action\_text, command=self.save\_task)

self.btn\_action.pack(pady=10)

self.top\_level.grab\_set()

def save\_task(self):

*# Get data from form fields*

data = {field: self.fields[field].get() for field in self.fields}

*# If updating, use the existing task\_id*

if self.update:

data["task\_id"] = self.task.task\_id

else:

*# If adding a new task, ensure the task\_id has been filled*

if not data["task\_id"]:

messagebox.showerror("Erro", "O ID da tarefa é obrigatório.")

return

*# Create Task object and save to JSON*

self.task = Task(\*\*data)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", f"Tarefa {'atualizada' if self.update else 'adicionada'} com sucesso.")

self.top\_level.destroy()

Por favor, note que fizemos o seguinte:

* + No método **\_\_init\_\_**, quando não estamos atualizando (ou seja, adicionando uma nova tarefa), nós configuramos o estado do campo **task\_id** para **normal**, permitindo a edição.
  + No método **save\_task**, adicionamos uma verificação para garantir que o campo **task\_id** esteja preenchido ao adicionar uma nova tarefa. Se o campo estiver vazio, uma mensagem de erro será exibida e a função retornará imediatamente, impedindo que a tarefa seja salva.
  + A função **insert\_text** foi substituí
* 

**Adenilton Ferreira**

Saturday 9:11 PM

Tudo está funcionando perfeitamente quando eu gostaria, agora apenas para uma de mais uma outra que eu vou solicitar para melhorar a experiência do usuário

no formulário na hora de colocar a data Inicial e a data de encerramento de fim eu tenho que colocar a data inteira digitando não seria possível eu clicar ele abrir um calendário pequeno e eu escolher no calendário Qual é a data e ele já colocar a data para mim isso tanto para a data de start como para data de end? Segue o código completo para análise: import tkinter as tk from tkinter import messagebox from tkinter import simpledialog import customtkinter as ctk import json import os

**Path to save JSON files**

json\_save\_path = "C:/Users/AFERR136/OneDrive - azureford/DEV PYTHON/MAPEAMENTO DB2/BACKUP-2"

**Check if the path exists and create if it doesn't**

if not os.path.exists(json\_save\_path): os.makedirs(json\_save\_path)

**Task Class**

class Task: def **init**(self, task\_id, responsible, name, start\_date, end\_date, completed): self.task\_id = task\_id self.responsible = responsible self.name = name self.start\_date = start\_date self.end\_date = end\_date self.completed = completed

def to\_dict(self):

return {

"task\_id": self.task\_id,

"responsible": self.responsible,

"name": self.name,

"start\_date": self.start\_date,

"end\_date": self.end\_date,

"completed": self.completed

}

@staticmethod

def save\_to\_json(task, path):

with open(os.path.join(path, f"{task.task\_id}.json"), 'w') as f:

json.dump(task.to\_dict(), f, indent=4)

@staticmethod

def load\_from\_json(task\_id, path):

try:

with open(os.path.join(path, f"{task\_id}.json"), 'r') as f:

data = json.load(f)

return Task(\*\*data)

except FileNotFoundError:

return None

@staticmethod

def delete\_json(task\_id, path):

os.remove(os.path.join(path, f"{task\_id}.json"))

**Main Application Class**

class MyApp(ctk.CTk): def **init**(self): super().**init**()

self.title("MY DAILY TASKS")

self.geometry("700x500")

self.minsize(700, 500)

# Left frame with buttons

self.frame\_buttons = ctk.CTkFrame(self, width=200, height=500)

self.frame\_buttons.pack(side="left", fill="y", padx=10, pady=10)

self.btn\_add\_task = ctk.CTkButton(self.frame\_buttons, text="Adicionar tarefas", command=self.add\_task)

self.btn\_add\_task.pack(pady=10)

self.btn\_update\_task = ctk.CTkButton(self.frame\_buttons, text="Atualizar tarefas", command=self.update\_task)

self.btn\_update\_task.pack(pady=10)

self.btn\_delete\_task = ctk.CTkButton(self.frame\_buttons, text="Excluir tarefas", command=self.delete\_task)

self.btn\_delete\_task.pack(pady=10)

# Central frame with task table

self.frame\_table = ctk.CTkFrame(self, width=500, height=500)

self.frame\_table.pack(side="left", fill="both", expand=True, padx=10, pady=10)

# Table headers

self.table\_headers = [

"ID da tarefa", "Nome do responsável", "Nome da tarefa",

"Data de início", "Data de fim", "Status"

]

self.columns = len(self.table\_headers)

self.table = ctk.CTkFrame(self.frame\_table)

self.table.pack(fill="both", expand=True)

for i, header in enumerate(self.table\_headers):

label = ctk.CTkLabel(self.table, text=header, anchor="w", fg\_color=("gray75", "gray30"))

label.grid(row=0, column=i, sticky="nsew", padx=2, pady=2)

# Configure grid column weights

for i in range(self.columns):

self.table.grid\_columnconfigure(i, weight=1)

self.table.grid\_rowconfigure(0, weight=0) # Header row should not expand

# List to store current tasks

self.tasks = []

self.task\_labels = []

# Load existing tasks from JSON files

self.load\_tasks()

self.protocol("WM\_DELETE\_WINDOW", self.on\_closing) # Handle window closing

def load\_tasks(self):

# Clear existing tasks and labels

for label in self.task\_labels:

label.destroy()

self.tasks.clear()

self.task\_labels.clear()

# Load tasks from JSON

for filename in os.listdir(json\_save\_path):

if filename.endswith(".json"):

task\_id = filename.split(".")[0]

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

self.tasks.append(task)

self.display\_task(task)

def display\_task(self, task):

row = len(self.tasks)

data = task.to\_dict()

for col, key in enumerate(data):

label = ctk.CTkLabel(self.table, text=str(data[key]), anchor="w")

label.grid(row=row, column=col, sticky="nsew", padx=2, pady=2)

self.task\_labels.append(label)

# Add status label with color

status\_label = ctk.CTkLabel(self.table, text=data["completed"], anchor="w", fg\_color=("green" if data["completed"] == "Concluído" else "red"))

status\_label.grid(row=row, column=self.columns-1, sticky="nsew", padx=2, pady=2)

self.task\_labels.append(status\_label)

# Update grid row weights

for i in range(1, len(self.tasks) + 1):

self.table.grid\_rowconfigure(i, weight=1)

def add\_task(self):

# Open new window to add task

task\_window = TaskWindow(self, title="Adicionar Tarefa", save\_path=json\_save\_path)

self.wait\_window(task\_window.top\_level)

# Reload tasks after adding

self.load\_tasks()

def update\_task(self):

# Open new window to update task

task\_id = simpledialog.askstring("Atualizar Tarefa", "Digite o ID da tarefa que deseja atualizar:")

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

task\_window = TaskWindow(self, title="Atualizar Tarefa", task=task, save\_path=json\_save\_path, update=True)

self.wait\_window(task\_window.top\_level)

# Reload tasks after updating

self.load\_tasks()

else:

messagebox.showerror("Erro", "Tarefa não encontrada.")

# Esta função deve estar no mesmo nível de indentação que as outras funções da classe MyApp

def delete\_task(self):

# Ask for task ID to delete

task\_id = simpledialog.askstring("Excluir Tarefa", "Digite o ID da tarefa que deseja excluir:")

task = Task.load\_from\_json(task\_id, json\_save\_path)

if task:

confirm = messagebox.askyesno("Confirmar Exclusão", "Você tem certeza que deseja excluir esta tarefa?")

if confirm:

Task.delete\_json(task\_id, json\_save\_path)

messagebox.showinfo("Sucesso", "Tarefa excluída com sucesso.")

self.load\_tasks()

else:

messagebox.showerror("Erro", "Tarefa não encontrada.")

def on\_closing(self):

# Handle any cleanup here if necessary before closing

self.destroy()

**Task Window Class for Add/Update Task**

class TaskWindow: def **init**(self, parent, title, task=None, save\_path="", update=False): self.top\_level = ctk.CTkToplevel(parent) self.top\_level.title(title) self.save\_path = save\_path self.update = update

# If updating, preload the task info

if task:

self.task = task

else:

self.task = Task(task\_id="", responsible="", name="", start\_date="", end\_date="", completed="Não Concluído")

# Create form fields

self.fields = {}

for field in ["task\_id", "responsible", "name", "start\_date", "end\_date"]:

self.fields[field] = ctk.CTkEntry(self.top\_level, placeholder\_text=field.replace("\_", " ").title())

self.fields[field].pack(pady=5)

if task: # Pre-fill fields if updating

self.fields[field].delete(0, tk.END) # Clear the current text

self.fields[field].insert(0, getattr(task, field)) # Insert new text

if field == "task\_id":

self.fields[field].configure(state='disabled') # Disable editing of task\_id

else: # If adding, make sure the task\_id field is enabled

self.fields[field].configure(state='normal')

self.fields["completed"] = ctk.CTkComboBox(self.top\_level, values=["Concluído", "Não Concluído"])

if task:

self.fields["completed"].set(self.task.completed)

else:

self.fields["completed"].set("Não Concluído")

self.fields["completed"].pack(pady=5)

# Add or Update button

action\_text = "Atualizar" if update else "Adicionar"

self.btn\_action = ctk.CTkButton(self.top\_level, text=action\_text, command=self.save\_task)

self.btn\_action.pack(pady=10)

self.top\_level.grab\_set()

def save\_task(self):

# Get data from form fields

data = {field: self.fields[field].get() for field in self.fields}

# If updating, use the existing task\_id

if self.update:

data["task\_id"] = self.task.task\_id

else:

# If adding a new task, ensure the task\_id has been filled

if not data["task\_id"]:

messagebox.showerror("Erro", "O ID da tarefa é obrigatório.")

return

# Create Task object and save to JSON

self.task = Task(\*\*data)

Task.save\_to\_json(self.task, self.save\_path)

messagebox.showinfo("Sucesso", f"Tarefa {'atualizada' if self.update else 'adicionada'} com sucesso.")

self.top\_level.destroy()

**Run the application**

if **name** == "**main**": app = MyApp() app.mainloop()

* **OpenAI GPT4**

Saturday 9:12 PM

Para adicionar um seletor de datas (date picker) ao seu formulário, você pode usar o módulo **tkcalendar**. Primeiro, você precisará instalar este módulo, que fornece o widget **Calendar** e o widget **DateEntry** para escolher datas.

Instale o **tkcalendar** com o seguinte comando:

Bash

pip install tkcalendar

Agora, vamos modificar a classe **TaskWindow** para incluir o **DateEntry** para os campos de data. Aqui está como você poderia fazer isso: