Projeto Final Python

Sumário

Descrição

Este projeto implementa um aplicativo usando a biblioteca baseada no TKinter, o CustomTkinter oferece funcionalidades personalizadas para a criação de interfaces gráficas em Python. O aplicativo é um sistema básico de criação e leitura (CR) que utiliza um banco de dados MySQL para armazenar informações. Para a ideia do nosso projeto, vamos trabalhar voltado para o cadastro de alunos em uma escola de cursos.

Dependências

Para podermos iniciar a construção do nosso projeto, será necessário instalar todas as dependências necessárias. Você pode instalar as dependências necessárias executando os seguintes comandos no terminal:

```
pip install customtkinter
pip install mysql-connector
```

Estrutura e Criação do Projeto

Arquivo Principal

main.py

O arquivo principal do projeto. Ele é responsável por iniciar a aplicação e criar a instância da classe Application do módulo app.

Crie o arquivo main.py

```
import customtkinter as ctk
from view.App import Application

if __name__ == '__main__':
    # Cria uma instância da classe CTk() para criar uma janela principal
    root = ctk.CTk()

    # Cria uma instância da classe Application passando a janela principal como
argumento
    app = Application(root)

    # Inicia o loop principal da aplicação, onde a interface gráfica será atualizada
e responderá a eventos
    root.mainloop()
```

Conexão com o Banco de Dados

Banco.py

Este módulo contém a classe Banco, que é responsável por lidar com a conexão e operações no banco de dados MySQL. Ele fornece métodos para inicializar a conexão com o banco de dados, executar consultas SQL e manipular os dados.

Crie o arquivo Banco.py

```python
import mysql.connector

class Banco:
    def __init__(self):
        self.host = 'localhost'
        self.user = 'root'
        self.password = ''
        self.database = 'projeto_python'
        self.connection = None

    def conectar(self):
        try:
            self.connection = mysql.connector.connect(
                host=self.host,
                user=self.user,
                password=self.password,
                database=self.database
            )
            print("Conexão estabelecida com sucesso!")
        except mysql.connector.Error as e:
            print(f"Erro ao conectar ao banco de dados: {e}")

    # Método para executar uma operação de inserção no banco de dados
    def executar_insert(self, query, params=None):
        try:
            cursor = self.connection.cursor()
            if params:
                cursor.execute(query, params)
            else:
                cursor.execute(query)
            self.connection.commit()
            last_row_id = cursor.lastrowid
            cursor.close()
            return last_row_id  # Retorna o ID da última linha inserida
        except mysql.connector.Error as e:
            print(f"Erro ao executar consulta: {e}")
```

```python
    # Método para executar uma operação de seleção no banco de dados
    def executar_select(self, query, params=None):
        try:
            cursor = self.connection.cursor(dictionary=True)
            if params:
                cursor.execute(query, params)
            else:
                cursor.execute(query)
            result = cursor.fetchall()  # Obtém todos os resultados da consulta
            cursor.close()
            return result
        except mysql.connector.Error as e:
            print(f"Erro ao executar consulta: {e}")
```

Definições de Modelos de Dados

Model/models.py

Neste módulo, são definidos os modelos de dados utilizados pela aplicação. Cada modelo é representado por uma classe que mapeia uma tabela no banco de dados. Os modelos incluem métodos para realizar operações de criação e leitura específicas para cada entidade.

Crie a pasta Model, em seguida o arquivo models.py dentro dela

```python
class Aluno:
    def __init__(self, id, nome, sexo, email):
        self.id = id
        self.nome = nome
        self.sexo = sexo
        self.email = email

class Curso:
    def __init__(self, id, nome_curso):
        self.id = id
        self.nome_curso = nome_curso

class Matricula:
    def __init__(self, id, aluno_id, curso_id):
        self.id = id
        self.aluno_id = aluno_id
        self.curso_id = curso_id
```

Operações no Banco de Dados

CR_Ope.py

O arquivo CR_Ope implementa a classe CR_Ope, que é responsável por realizar operações de criação e leitura no banco de dados. Ele utiliza os modelos definidos em Model/models.py para criar e ler registros no banco de dados.

Crie o arquivo CR_Ope.py

```
from Banco import Banco

class CR_Ope:
    def __init__(self):
        self.banco = Banco()  # Cria uma instância da classe Banco
        self.banco.conectar()  # Conecta ao banco de dados

    def inserir_aluno(self, nome, sexo, email):
        query = "INSERT INTO aluno (nome, sexo, email) VALUES (%s, %s, %s)"  # Query SQL para
inserção de aluno
        params = (nome, sexo, email)  # Parâmetros da consulta
        self.banco.executar_insert(query, params)  # Executa a consulta de inserção

    def inserir_matricula(self, aluno_id, curso_id):
        query = "INSERT INTO matricula (aluno_id, curso_id) VALUES (%s, %s)"
        params = (aluno_id, curso_id)
        self.banco.executar_insert(query, params)

    def selecionar_cursos(self):
        query = "SELECT * FROM cursos"
        return self.banco.executar_select(query)

    def selecionar_alunos_com_cursos(self):
        query = """
        SELECT matricula.id, aluno.nome, aluno.sexo, aluno.email, cursos.nome_curso
        FROM aluno
        LEFT JOIN matricula ON aluno.id = matricula.aluno_id
        LEFT JOIN cursos ON matricula.curso_id = cursos.id
        """  # Query SQL para seleção de alunos matriculados
        return self.banco.executar_select(query)

    def obter_ultimo_id_aluno(self):
        query = "SELECT LAST_INSERT_ID() AS last_id"  # Query SQL para obter o último ID
inserido
        result = self.banco.executar_select(query)
        if result:
            return result[0]['last_id']  # Retorna o último ID inserido
          else:
            return None  # Retorna None se não houver resultado
```

Interface Gráfica da Aplicação

view/app.py

Este módulo contém a classe App, que representa a interface gráfica da aplicação. Ele utiliza CustomTkinter para criar janelas, botões, campos de entrada e outros componentes da GUI. Dentro dela temos a classe App interage com a classe CR_Ope para exibir e manipular os dados no banco de dados.

Crie a pasta view, em seguida o arquivo app.py dentro dela

```
import customtkinter as ctk
import tkinter as tk
from CR_Ope import CR_Ope as cr
from PIL import Image


class Application():
    def __init__(self, app):
        self.app = app  # Define a janela principal da aplicação
        app.title("Python GUI")  # Define o título da janela principal
        self.create_widgets()
        self.cr = cr()  # Instâcia da classe CR_Ope
        self.carregar_lista_alunos()


    # Método para criar os widgets da interface gráfica
    def create_widgets(self):
        # Cria um frame principal para organizar os widgets
        main_frame = ctk.CTkFrame(self.app)
        main_frame.pack(expand=True, fill='both', padx=10, pady=10)


        # Cria um frame para o título da aplicação
        title_frame = ctk.CTkFrame(main_frame, height=80)
        title_frame.pack(fill='x', padx=10, pady=(0, 10))
        ctk.CTkLabel(title_frame, text="Projeto Final Python",  font=('verdana', 20)).pack(padx=10, pady=5)


        # Criando o objeto CTkImage com a imagem
        my_image = ctk.CTkImage(dark_image=Image.open("./projeto_ctk/view/python.jpeg"), size=(50, 50))


        # Exibindo a imagem em um rótulo
        image_label = ctk.CTkLabel(main_frame, image=my_image, text="")
        image_label.pack(anchor='n', side='left',padx=10, pady=10)


        info_frame = ctk.CTkFrame(main_frame)
        info_frame.pack(anchor='w', side='top', padx=20, pady=5)
        ctk.CTkLabel(info_frame, text="Formulário Para Cadastro de Alunos", font=('Verdana', 16)).grid(row=0,
column=1, padx=50, pady=15)
        # Cria um frame para informações do aluno
        form_frame = ctk.CTkFrame(main_frame)
        form_frame.pack(anchor='w', side='left', padx=20, pady=10)
```

```
        # Cria labels e entradas para os campos
        labels = ["Nome:", "Gênero:", "Curso de Tecnologia:", "E-mail:"]
        self.entries = {}
        for i, label_text in enumerate(labels):
            ctk.CTkLabel(form_frame, text=label_text).grid(row=i, column=0, padx=5, pady=20, sticky="e")
            if label_text == "Gênero:":
                self.radio_var = tk.IntVar(value=0)
                ctk.CTkRadioButton(form_frame, text="Masculino", variable=self.radio_var, value=1).grid(row=i,
column=1, padx=0, pady=2, sticky="w")
                ctk.CTkRadioButton(form_frame, text="Feminino", variable=self.radio_var, value=2).grid(row=i,
column=2, padx=0, pady=2, sticky="e")
            elif label_text == "Curso de Tecnologia:":
                ctk.CTkLabel(form_frame, text=label_text).grid(row=i, column=0, padx=5, pady=5, sticky="e")
                self.curso_combobox = ctk.CTkComboBox(form_frame, values=["Python", "JavaScript", "Java",
"PHP", "Análise de dados"])
                self.curso_combobox.grid(row=i, column=1, padx=5, pady=5, columnspan=2, sticky="w")
            else:
                self.entries[label_text] = ctk.CTkEntry(form_frame)
                self.entries[label_text].grid(row=i, column=1, padx=5, pady=5, sticky="w")

        # Cria um frame para botão de adicionar aluno
        btn_frame = ctk.CTkFrame(main_frame)
        btn_frame.pack(anchor='w', side='left', padx=10, pady=10)
        ctk.CTkButton(btn_frame, text="Adicionar", command=self.adicionar_aluno).pack(padx=10, pady=10)
        ctk.CTkButton(btn_frame, text="Exportar Lista", command=self.exportar_lista_alunos).pack(padx=10,
pady=10)

        # Cria um frame para exibir a lista de alunos
        list_frame = ctk.CTkFrame(main_frame)
        list_frame.pack(fill='both', expand=True, padx=10, pady=10)
        self.lista_alunos = ctk.CTkTextbox(list_frame, width=540, height=200)
        self.lista_alunos.pack(fill='both', expand=True)

    def exportar_lista_alunos(self):
        alunos = self.cr.selecionar_alunos_com_cursos()
        with open("lista_alunos.csv", "w") as f:
            for aluno in alunos:
                linha =
f"{aluno['id']},{aluno['nome']},{aluno['sexo']},{aluno['nome_curso']},{aluno['email']}\n"
                f.write(linha)
        tk.messagebox.showinfo("Exportar", "Lista de alunos exportada com sucesso para lista_alunos.csv")

    # Método para adicionar um novo aluno
    def adicionar_aluno(self):
        # Obtém os valores dos campos de entrada
        nome = self.entries["Nome:"].get()
        sexo = "M" if self.radio_var.get() == 1 else "F"
        curso = self.curso_combobox.get()
```

```
        email = self.entries["E-mail:"].get()

        # Insere o aluno no banco de dados
        self.cr.inserir_aluno(nome, sexo, email)
        curso_id = self.obter_id_curso(curso)

        if curso_id:
            aluno_id = self.cr.obter_ultimo_id_aluno()
            self.cr.inserir_matricula(aluno_id, curso_id)

            for entry in self.entries.values():
                entry.delete(0, tk.END)
            self.curso_combobox.set("")

            tk.messagebox.showinfo("Sucesso", "Aluno cadastrado com sucesso!")

            # Carrega a lista de alunos após a inserção bem-sucedida
            self.carregar_lista_alunos()
        else:
            tk.messagebox.showinfo("Erro", "Falha ao realizar matrícula")

    # Método para obter o ID do curso a partir do nome do curso
    def obter_id_curso(self, nome_curso):
        cursos = self.cr.selecionar_cursos()
        for curso in cursos:
            if curso['nome_curso'] == nome_curso:
                return curso['id']
        return None

    # Método para carregar a lista de alunos na interface gráfica
    def carregar_lista_alunos(self):
        self.lista_alunos.configure(state="normal")  # Habilita edição temporariamente
        self.lista_alunos.delete("0.0", tk.END)  # Limpa o texto na área de texto
        alunos = self.cr.selecionar_alunos_com_cursos()  # Seleciona todos os alunos com cursos do banco de
dados
        for aluno in alunos:
            # Formata as informações do aluno e adiciona na área de texto
            info_aluno =
f"ID: {aluno['id']} Nome: {aluno['nome']} Curso: {aluno['nome_curso']} Email: {aluno['email']}\n"
            self.lista_alunos.insert(tk.END, info_aluno)
        self.lista_alunos.configure(state="disabled")  # Desabilita edição novamente
```

Execução

Para executar o aplicativo, basta rodar o arquivo main.py. Certifique-se de ter todas as dependências instaladas.

Resultado Final: