from xmlrpc.client import \_binary

import streamlit as st

import pandas as pd

from sqlalchemy import DATETIME, create\_engine, text

from streamlit import connection

from sqlalchemy.orm import declarative\_base

from sqlalchemy.orm import sessionmaker, session

from streamlit\_option\_menu import option\_menu

from sqlalchemy.orm import sessionmaker, declarative\_base

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.orm import sessionmaker, declarative\_base

import io

import json

from urllib.parse import quote\_plus

import plotly.express as px

from datetime import datetime

from io import StringIO

from sqlalchemy import Column, Integer, String, TIMESTAMP, LargeBinary, func

#tabela Unidade

dados = [['10201000', 'Praia'],

         ['10202000', 'São Domingos'],

         ['10203000', 'Santa Catarina'],

         ['10204000', 'Tarrafal'],

         ['10205000', 'Calheta'],

         ['10206000', 'Santa Cruz'],

         ['10701000', 'Mosteiros'],

         ['10702000', 'São Filipe'],

         ['10801000', 'Maio'],

         ['10901000', 'Brava'],

         ['10101000', 'Mindelo'],

         ['10301000', 'SAL'],

         ['10401000', 'BOAVISTA'],

         ['10501000', 'VILA DA RIBEIRA BRAVA'],

         ['10601000', 'R.GRANDE - N.S.Rosário'],

         ['10602000', 'PORTO NOVO - S.João Baptista'],

         ['10603000', 'PAUL - S.António das Pombas'],

         ['10502000', 'Tarrafal S.Nicolau'],

         ['10000000', 'Electra SUL']

         ]

unidade = pd.DataFrame(dados, columns=['UC', 'Unidade'])

unidade['UC'] = unidade['UC'].astype(int)

unidade.set\_index('UC', inplace=True)

#tabela Produto

dados2 = [['EB', 'Baixa Tensão'],

          ['EE', 'Baixa Tensão Especial'],

          ['EM', 'Media Tensão'],

          ['AG', 'Agua']

          ]

produto = pd.DataFrame(dados2, columns=['Prod', 'Produto'])

produto.set\_index('Prod', inplace=True)

#tabela Tipo Factura

dados3 = [['11', 'Em Ciclo Leitura'],

          ['12', 'Em Ciclo Estimativa'],

          ['22', 'Baixa Voluntária'],

          ['23', 'Baixa por Dívida'],

          ['24', 'Alterações Contratuais'],

          ['28', 'Baixa Forçada'],

          ['29', 'Substit. Modif.'],

          ['30', 'Substituição'],

          ['33', 'Acerto de Cobrança'],

          ['39', 'Facturação Diversa'],

          ['99', 'Lig Relig CompPg']

          ]

tp\_fact = pd.DataFrame(dados3, columns=['TP\_FACT', 'Tipo\_Factura'])

tp\_fact['TP\_FACT'] = tp\_fact['TP\_FACT'].astype(int)

tp\_fact.set\_index('TP\_FACT', inplace=True)

#tabela Tipo Cliente

dados4 = [['72', 'Empresa Publica'],

          ['82', 'Colectivos'],

          ['93', 'Industriais'],

          ['94', 'Construção'],

          ['73', 'Estado-Patrimonio'],

          ['91', 'Domésticos'],

          ['92', 'Comércio, Industria, Agricul.'],

          ['21', 'Consumos Próprios'],

          ['31', 'Autarquias'],

          ['51', 'Instituições'],

          ['71', 'Estado-Tesouro'],

          ['XX', 'Clientes Senhas de Água'],

          ]

tip\_client = pd.DataFrame(dados4, columns=['TP\_CLI', 'Tipo\_Cliente'])

tip\_client['TP\_CLI'] = tip\_client['TP\_CLI'].astype(str)

tip\_client.set\_index('TP\_CLI', inplace=True)

#tabela Tarifa

dados5 = [['A1', 'Tarifa Água I'],

            ['A2', 'Tarifa Água II'],

            ['A3', 'Tarifa Água III B'],

            ['A4', 'Tarifa Água III A'],

            ['A5', 'Tarifa Água II (Turismo)'],

            ['AD', 'ADA'],

            ['AP', 'Água Praia'],

            ['B4', 'Autotanques II'],

            ['CD', 'Central Dessalinizadora'],

            ['CP', 'Consumos Proprios'],

            ['R4', 'Autotanques I (Social)'],

            ['SA', 'Senhas de Água'],

            ['XX', 'Venda de Água Avulso'],

            ['AV', 'Avença'],

            ['CE', 'Caixa de Escada'],

            ['CP', 'Consumos Proprios'],

            ['D1', 'Tarifa D'],

            ['D11', 'Tarifa D'],

            ['D2', 'Tarifa D-S. Nicolau'],

            ['D3', 'Tarifa D-Social-S. Nicolau'],

            ['D4', 'Tarifa D - Maio'],

            ['D5', 'Tarifa D - Social - Maio'],

            ['DS', 'Tarifa D - Social'],

            ['IP', 'Iluminação Publica'],

            ['LM', 'Ligação Provisória - MONO'],

            ['LP', 'Ligação Provisória'],

            ['LT', 'Ligação Provisória - TRI'],

            ['LU', 'Ligação Provisória - MONO URG'],

            ['S1', 'Tarifa Social'],

            ['SF', 'Semáfores'],

            ['T1', 'Trabalhador Electra-S. Nicolau'],

            ['T2', 'Trab. Electra Is.RTC-S.Nicolau'],

            ['T3', 'Trabalhador Electra - Maio'],

            ['T4', 'Trab. Electra Is. RTC - Maio'],

            ['TB', 'Trabalhador Electra'],

            ['TI', 'Trab. Electra Isento RTC'],

            ['TU', 'Ligação Provisória - TRI URG'],

            ['AV', 'Tarifa Avença'],

            ['E1', 'Tarifa BTE 1'],

            ['E2', 'Tarifa BTE'],

            ['S1', 'Tarifa Social'],

            ['M1', 'Tarifa MT 1'],

            ['M2', 'Tarifa MT'],

            ['M3', 'Tarifa MT'],

            ['S1', 'Tarifa Social'],

            ['TBP', 'Trabalhador Partilhado'],

            ['TBB', 'Trab. Beneficiário']

          ]

tarifa = pd.DataFrame(dados5, columns=['COD\_TARIFA', 'Tarifa'])

tarifa['COD\_TARIFA'] = tarifa['COD\_TARIFA'].astype(str)

tarifa.set\_index('COD\_TARIFA', inplace=True)

#tabela de mês de numero para texto

dados6 = [['1', 'Janeiro'],

          ['2', 'Fevereiro'],

          ['3', 'Março'],

          ['4', 'Abril'],

          ['5', 'Maio'],

          ['6', 'Junho'],

          ['7', 'Julho'],

          ['8', 'Agosto'],

          ['9', 'Setembro'],

          ['10', 'Outubro'],

          ['11', 'Novembro'],

          ['12', 'Dezembro'],

]

refmes = pd.DataFrame(dados6, columns=['Me', 'Mês'])

refmes.set\_index('Me', inplace=True)

# definir roteiro qual unidade comercial usar

dados7 = [["Avenças PRAIA", "Praia"],

            ["GC REMOTA", "Praia"],

            ["MICRO PR", "Praia"],

            ["IP PRAIA REMOTA", "Praia"],

            ["IP PRAIA", "Praia"],

            ["MT", "Praia"],

            ["Praia GC I", "Praia"],

            ["Praia GC II", "Praia"],

            ["Praia GC III", "Praia"],

            ["Praia", "Praia"],

            ["Rural", "Praia"],

            ["ADS - S.Domingos", "São Domingos"],

            ["MICRO SD", "São Domingos"],

            ["Avenças S.DOMINGOS", "São Domingos"],

            ["MT S.DOMINGOS", "São Domingos"],

            ["Milho Branco", "São Domingos"],

            ["Praia Baixo e Praia Formosa", "São Domingos"],

            ["Roteiro IP S.DOMINGOS REMOTA", "São Domingos"],

            ["S.DOMINGOS GC", "São Domingos"],

            ["S.DOMINGOS", "São Domingos"],

            ["ADS - Assomada", "Santa Catarina"],

            ["MICRO AS", "Santa Catarina"],

            ["Assomada", "Santa Catarina"],

            ["GC Santa Catarina", "Santa Catarina"],

            ["Picos", "Santa Catarina"],

            ["Roteiro IP Santa Catarina", "Santa Catarina"],

            ["Rª da Barca", "Santa Catarina"],

            ["ADS - Tarrafal", "Tarrafal"],

            ["MICRO TA", "Tarrafal"],

            ["Roteiro IP TARRAFAL", "Tarrafal"],

            ["Tarrafal - CHÃO BOM", "Tarrafal"],

            ["Tarrafal - VILA", "Tarrafal"],

            ["Tarrafal-Rural", "Tarrafal"],

            ["ADS - Calheta", "Calheta"],

            ["MICRO CA", "Calheta"],

            ["CALHETA I", "Calheta"],

            ["CALHETA II", "Calheta"],

            ["Calheta GCI", "Calheta"],

            ["Calheta", "Calheta"],

            ["Flamengos", "Calheta"],

            ["MIGUEL GOMES", "Calheta"],

            ["PIZARRA", "Calheta"],

            ["RIBEIRA PILÃO BRANCO", "Calheta"],

            ["Roteiro IP CALHETA", "Calheta"],

            ["Veneza", "Calheta"],

            ["ADS - Santa Cruz", "Santa Cruz"],

            ["MICRO SC", "Santa Cruz"],

            ["ORGÃOS", "Santa Cruz"],

            ["Roteiro IP SANTA CRUZ", "Santa Cruz"],

            ["SANTA CRUZ GCI", "Santa Cruz"],

            ["SANTA CRUZ RP", "Santa Cruz"],

            ["SANTA CRUZ RURAL", "Santa Cruz"],

            ["SANTA CRUZ SA", "Santa Cruz"],

            ["VILA PEDRA BADEJO", "Santa Cruz"],

            ["IP MOSTEIROS", "Mosteiros"],

            ["MOSTEIROS GC", "Mosteiros"],

            ["MICRO MO", "Mosteiros"],

            ["Mosteiro-Rural", "Mosteiros"],

            ["Mosteiros", "Mosteiros"],

            ["CLIENTES BT", "São Filipe"],

            ["Clientes IP", "São Filipe"],

            ["CLIENTES MT", "São Filipe"],

            ["SAO FILIPE GC", "São Filipe"],

            ["MICRO SF", "São Filipe"],

            ["Barreiro-Rural", "Maio"],

            ["Calheta-Rural", "Maio"],

            ["Figueira da Horta/Seca-Rural", "Maio"],

            ["IP MAIO", "Maio"],

            ["MAIO GC", "Maio"],

            ["Morrinho-Rural", "Maio"],

            ["Morro-Rural", "Maio"],

            ["NORTE RURAL", "Maio"],

            ["Ribeira D.João-Rural", "Maio"],

            ["VILA P.Ingles", "Maio"],

            ["MICRO MA", "Maio"],

            ["BRAVA GC", "Brava"],

            ["IP BRAVA", "Brava"],

            ["MICRO BR", "Brava"],

            ["NSM Rural", "Brava"],

            ["SJB Rural", "Brava"],

            ["VN Sintra", "Brava"]

            ]

uc = pd.DataFrame(dados7, columns=['Roteiro', 'Unidade'])

# criar tabela de função

dados8 = [

        ["11", "Ativa"],

        ["12", "Ativa"],

        ["21", "Agua"],

        ["41", "Ativa"],

        ["49", "Reativa"],

        ["61", "Ativa"],

        ["69", "Reativa"],

        ["71", "Ponta"],

    ]

funcao = pd.DataFrame(dados8, columns=['Função', 'Descrição'])

# converter na função a coluna para formato inteiro

funcao['Função'] = funcao['Função'].astype(int)

# criar tabela de regiao

dados9 = [['Praia', 'SUL'],

         ['São Domingos', 'SUL'],

         ['Santa Catarina', 'SUL'],

         ['Tarrafal', 'SUL'],

         ['Calheta', 'SUL'],

         ['Santa Cruz', 'SUL'],

         ['Mosteiros', 'SUL'],

         ['São Filipe', 'SUL'],

         ['Maio', 'SUL'],

         ['Brava', 'SUL'],

         ['Mindelo', 'NORTE'],

         ['SAL', 'NORTE'],

         ['BOAVISTA', 'NORTE'],

         ['VILA DA RIBEIRA BRAVA', 'NORTE'],

         ['R.GRANDE - N.S.Rosário', 'NORTE'],

         ['PORTO NOVO - S.João Baptista', 'NORTE'],

         ['PAUL - S.António das Pombas', 'NORTE'],

         ['Tarrafal S.Nicolau', 'NORTE'],

         ['EDEC']

         ]

regiao = pd.DataFrame(dados9, columns=['Unidade', 'Regiao'])

regiao.set\_index('Unidade', inplace=True)

# criar tabela de estado de contrato

dados10 = [['0', 'Sem Contrato'],

           ['10', 'React. Pendente OS'],

           ['11', 'Pendente, Inspec, Não Aprovada'],

           ['12', 'Contrato em Vigor'],

           ['13', 'Reactivada de Inspecção'],

           ['14', 'Contrato Novo Ainda não Facturado'],

           ['15', 'Contrato Novo Pendente de OS'],

           ['17', 'Pend. Suspensão por Inspecção'],

           ['18', 'Suspenso por Inspecção'],

           ['19', 'Pend. Bai. Inspec não Aprovada'],

           ['21', 'Rescisão Voluntaria'],

           ['25', 'Baixa por Dívida'],

           ['27', 'Baixa Forçada'],

           ['28', 'Baixa por Inspec. não Aprovada']]

est\_contr = pd.DataFrame(dados10, columns=['EST CONTR', 'Estado Contrato'])

#est\_contr.set\_index('EST CONTR', inplace=True)

#criar cabeçalho Facturação

colunas = ['BOA IND', 'EMP ID', 'UC', 'Prod', 'DT\_PROC', 'DT\_FACT', 'NR\_FACT', 'CLI\_ID', 'CLI\_CONTA', 'CIL',

           'TP\_FACT', 'TP\_CLI', 'COD\_TARIFA', 'VAL\_TOT', 'CONCEITO', 'QTDE', 'VALOR']

#criar cabeçalho contros

colunas2 = [

    "EMP ID", "UC", "USR ID", "NIP", "PORTA", "CIL", "SIS ABAST", "CGV",

    "CLI CONTA", "TP\_CLI", "CAE ID", "TP USO", "NOME", "MORADA", "LOCALIDADE",

    "COD LOCAL", "Prod", "COD TARIFA", "SEQ CONTR", "EST CONTR",

    "DT CONTRATO", "DT INICIO", "DT BAIXA", "OBS", "REF", ""

]

#definir importação e tratamento do script facturação

def tratar\_factura():

    query = "SELECT \* FROM facturação"

    factura2 = pd.read\_sql(query, engine)

    #filtrar consumo

    factura2 = factura2.loc[factura2['CONCEITO'] == 'X30']

    #remover duplicado

    factura2 = factura2.drop\_duplicates(subset='NR\_FACT')

    #alterar formato para str

    factura2['CIL'] = factura2['CIL'].astype(str)

    factura2['CLI\_ID'] = factura2['CLI\_ID'].astype(str)

    factura2['CLI\_CONTA'] = factura2['CLI\_CONTA'].astype(str)

    factura2['DT\_PROC'] = factura2['DT\_PROC'].astype(str)

    factura2['DT\_PROC'] = pd.to\_datetime(factura2['DT\_PROC'])

    factura2['DT\_PROC'] = factura2['DT\_PROC'].dt.date

    factura2['DT\_FACT'] = factura2['DT\_FACT'].astype(str)

    factura2['DT\_FACT'] = pd.to\_datetime(factura2['DT\_FACT'])

    factura2['Ano'] = factura2['DT\_FACT'].dt.year

    factura2['Me'] = factura2['DT\_FACT'].dt.month

    factura2['Me'] = factura2['Me'].astype(str)

    factura2['Ano'] = factura2['Ano'].astype(str)

    factura2['DT\_FACT'] = factura2['DT\_FACT'].dt.date

    factura3 = pd.merge(factura2, refmes, on='Me', how='left')

    factura3 = factura3[['Ano', 'Regiao', 'Unidade', 'CIL', 'CLI\_ID', 'CLI\_CONTA', 'Tipo\_Cliente','Produto', 'Tipo\_Factura', 'NR\_FACT',  'Mês',

                            'DT\_PROC', 'DT\_FACT', 'Tarifa', 'VAL\_TOT', 'CONCEITO', 'QTDE', 'VALOR']]

    #alterar nome das colunas

    factura3 = factura3.rename(columns={'CLI\_ID': 'Cliente'})

    factura3 = factura3.rename(columns={'CLI\_CONTA': 'Cliente Conta'})

    factura3 = factura3.rename(columns={'Tipo\_Cliente': 'Tipo Cliente'})

    factura3 = factura3.rename(columns={'Tipo\_Factura': 'Tipo Factura'})

    factura3 = factura3.rename(columns={'NR\_FACT': 'Nº Factura'})

    factura3 = factura3.rename(columns={'DT\_PROC': 'Data Processamento'})

    factura3 = factura3.rename(columns={'DT\_FACT': 'Data Facturação'})

    factura3 = factura3.rename(columns={'VAL\_TOT': 'Valor Facturado'})

    factura3 = factura3.rename(columns={'QTDE': 'Kwh'})

    factura3 = factura3.rename(columns={'VALOR': 'Valor Cons (ECV)'})

    return factura3

#criar dados se para maturidade

def classificar\_maturidade(media):

    if media == 0:

        return "0"

    elif 1 <= media <= 20:

        return "1 a 20"

    elif 21 <= media <= 30:

        return "21 a 30"

    elif 31 <= media <= 40:

        return "31 a 40"

    elif 41 <= media <= 60:

        return "41 a 60"

    elif 61 <= media <= 100:

        return "61 a 100"

    elif 101 <= media <= 200:

        return "101 a 200"

    elif 201 <= media <= 300:

        return "201 a 300"

    elif 301 <= media <= 400:

        return "301 a 400"

    elif 401 <= media <= 500:

        return "401 a 500"

    elif 501 <= media <= 1000:

        return "501 a 1000"

    elif 1001 <= media <= 5000:

        return "1001 a 5000"

    elif 5001 <= media <= 20000:

        return "5001 a 20000"

    elif media > 20000:

        return "> 20000"

    else:

        return "Valor inválido"

#definir importação

def importar\_contratos():

    query = "SELECT \* FROM contratos"

    contratos2 = pd.read\_sql(query, engine)

    #converter data

    contratos2['DT CONTRATO'] = pd.to\_datetime(contratos2['DT CONTRATO'], format='%Y%m%d', errors='coerce').dt.strftime('%d-%m-%Y')

    contratos2['DT INICIO'] = pd.to\_datetime(contratos2['DT INICIO'], format='%Y%m%d', errors='coerce').dt.strftime('%d-%m-%Y')

    contratos2['DT BAIXA'] = pd.to\_datetime(contratos2['DT BAIXA'], format='%Y%m%d', errors='coerce').dt.strftime('%d-%m-%Y')

    return contratos2

#inserir dados de cadastro

def insert\_user(nome, telefone, movel, cni, localidade, morada, filename, file\_data):

    session = SessionLocal()

    user = User(nome=nome, telefone=telefone, movel=movel, cni=cni, localidade=localidade, morada=morada, filename=filename, file\_data=file\_data)

    session.add(user)

    session.commit()

    session.close()

#ler arquivo config.json

with open("config/config.json") as f:

    config = json.load(f)

#criar ligação a base de dados

server =   config["BD\_SERVER"]

database = config["BD\_NAME"]

user = config["BD\_USER"]

password = config["BD\_PASSWORD"]

driver = config["BD\_DRIVER"]

#para situação de espaços no driver

driver\_encoded = quote\_plus(driver)

# String de conexão correta para SQLAlchemy + pyodbc

engine = create\_engine(

    f"mssql+pyodbc://{user}:{password}@{server}/{database}?driver={driver\_encoded}",

    fast\_executemany=True

)

Base = declarative\_base()

Session = sessionmaker(bind=engine)

session = Session()

SessionLocal = sessionmaker(bind=engine)

# Configuração da página

if "page\_config\_set" not in st.session\_state:

    st.set\_page\_config(page\_title="Login", layout="centered")

    st.session\_state["page\_config\_set"] = True

# Criar um estado de sessão para login

if "logged\_in" not in st.session\_state:

    st.session\_state.logged\_in = False

# Função para verificar login e obter nível de acesso

def check\_login(username, password):

    query = f"""

        SELECT nivel FROM usuarios

        WHERE username = '{username}' AND password = '{password}'

    """

    df = pd.read\_sql(query, engine)

    if not df.empty:

        return df.iloc[0]["nivel"]

    return None

# Interface do login

if not st.session\_state.logged\_in:

    st.title("Tela de Login ao Sistema")

    username = st.text\_input("Usuário")

    password = st.text\_input("Senha", type="password")

    if st.button("Login"):

        nivel = check\_login(username, password)

        if nivel:

            st.session\_state.logged\_in = True

            st.session\_state.username = username

            st.session\_state.nivel = nivel

            st.session\_state["page\_config\_set"] = False

            st.rerun()

        else:

            st.error("Usuário ou senha incorretos!")

# Interface após login

if st.session\_state.logged\_in:

    st.set\_page\_config(page\_title="Tratamento De Script's", layout="wide")

    # Sidebar com saudação

    st.sidebar.title(f"Bem-vindo, {st.session\_state.username}!")

    st.sidebar.markdown(f"\*\*Nível de acesso:\*\* {st.session\_state.nivel}")

    #'admin', 'gerente', 'factura', 'usuario', 'contagem', 'contrato'

    # Menu de opções conforme o nível de acesso

    if st.session\_state.nivel == "admin":

        menu\_opcoes = ["Home", "DAC - Lojas","Dep. Facturação", "Dep. Gestão Contagem", "Dep. Contratação","Administração"]

    elif st.session\_state.nivel == "gerente":

        menu\_opcoes = ["Home", "DAC - Lojas","Dep. Facturação", "Dep. Gestão Contagem", "Dep. Contratação", "Definição"]

    elif st.session\_state.nivel == "factura":

        menu\_opcoes = ["Home", "Dep. Facturação", "Definição"]

    elif st.session\_state.nivel == "contagem":

        menu\_opcoes = ["Home", "Dep. Gestão Contagem", "Definição"]

    elif st.session\_state.nivel == "contrato":

        menu\_opcoes = ["Home", "Dep. Contratação", "Definição"]

    elif st.session\_state.nivel == "usuario":

        menu\_opcoes = ["Home", "DAC - Lojas", "Definição"]

    else:

        menu\_opcoes = ["Home"]

    # Menu lateral

    with st.sidebar:

        selected = option\_menu(

            menu\_title="Menu Principal",

            options=menu\_opcoes,

            menu\_icon="cast",

            default\_index=0

        )

        # Páginas conforme menu

    if selected == "Home":

        st.title("Home")

        st.write("Bem-vindo à sua dashboard!")

    elif selected == "DAC - Lojas":

        st.title("DAC - Lojas")

        st.write("Área exclusiva para lojas.")

    elif selected == "Dep. Facturação":

        st.title("Dep. Facturação")

        #st.title("Facturação")

        menu = st.radio(

        "Seleção", ["Importação", "Dashboard", "Analise Maturidade"], horizontal=True

        )

        st.markdown("---")

        #campo de importação

        if menu == "Importação":

            st.title("Importação")

            #query importar facturação

            query = "SELECT \* FROM facturação"

            upload\_file = st.file\_uploader("Importar Facturação", type=["txt"])

            if upload\_file:

                st.markdown("---")

                content = upload\_file.read().decode("utf-8")

                factura = pd.read\_csv(io.StringIO(content), sep='\t', names=colunas)

                factuc = pd.merge(factura, unidade, on='UC', how='left')

                prodfact = pd.merge(factuc, produto, on='Prod', how='left')

                tpfact = pd.merge(prodfact, tp\_fact, on='TP\_FACT', how='left')

                tpfact['TP\_CLI'] = tpfact['TP\_CLI'].astype(str)

                clifact = pd.merge(tpfact, tip\_client, on='TP\_CLI', how='left')

                confact = pd.merge(clifact, tarifa, on='COD\_TARIFA', how='left')

                regfact = pd.merge(confact, regiao, on='Unidade', how='left')

                #organizar estrutura

                regfact = regfact[['Regiao', 'Unidade', 'CIL', 'CLI\_ID', 'CLI\_CONTA', 'Tipo\_Cliente','Produto', 'Tipo\_Factura', 'NR\_FACT', 'DT\_PROC', 'DT\_FACT', 'Tarifa',

                                    'VAL\_TOT', 'CONCEITO', 'QTDE', 'VALOR']]

                #função de carregar factura na base de dados

                if st.button("Guardar Facturação"):

                    try:

                        regfact.to\_sql("facturação", con=engine, if\_exists="append", index=False)

                        st.success("Dados inseridos com sucesso!")

                    except Exception as e:

                        st.error(f"Erro ao inserir dados: {e}")

        #campo de Dashboard

        if menu == "Dashboard":

            st.title("Dashboard")

            facturas\_tratadas = tratar\_factura()

            #dividir em colunas

            col1, col2, col3 = st.columns(3)

            #col1

            with col1:

                #filtrar região

                reg = st.multiselect(

                    "Definir Região: ",

                    options=facturas\_tratadas['Regiao'].unique(),

                )

                geral\_selection = facturas\_tratadas.query(

                    "`Regiao` == @reg"

                )

            #col2

            with col2:

                #filtar ano

                an = st.multiselect(

                    "Definir Ano: ",

                    options=geral\_selection['Ano'].unique(),

                )

                geral\_selection2 = geral\_selection.query(

                    "`Ano` == @an"

                )

            #col3

            with col3:

                #filtrar mês

                me = st.multiselect(

                    "Definir Mês",

                    options=geral\_selection2['Mês'].unique(),

                )

                geral\_selection3 = geral\_selection2.query(

                    "`Mês` == @me"

                )

            #criação de tabela dinamica por unidade

            tabdinamica = geral\_selection3.pivot\_table(

                index=['Unidade'],

                values=['Valor Facturado', 'Kwh'],

                aggfunc='sum',

                fill\_value=0

            )

            #criação de tabela dinamica por Tipo Cliente

            tabdinamica2 = geral\_selection3.pivot\_table(

                index=['Tipo Cliente'],

                values=['Valor Facturado', 'Kwh'],

                aggfunc='sum',

                fill\_value=0

            )

             #criação de tabela dinamica por Produto

            tabdinamica3 = geral\_selection3.pivot\_table(

                index=['Produto'],

                values=['Valor Facturado', 'Kwh'],

                aggfunc='sum',

                fill\_value=0

            )

            #criação de graficos por Unidade coemrcial

            #grafico UC Facturado

            fig\_ucval = px.bar(

                tabdinamica,

                x=tabdinamica.index,

                y=['Valor Facturado'],

                orientation="v",

                title="<b>Grafico Valor Facturado Por Unidade</b>",

                color\_discrete\_sequence=["#5F9EA0"],

                template="plotly\_white",

            )

            fig\_ucval.update\_layout(

                plot\_bgcolor="rgba(0,0,0,0)",

                xaxis=(dict(showgrid=False))

            )

            #grafico UC Kwh

            fig\_ucquant = px.bar(

                tabdinamica,

                x=tabdinamica.index,

                y=['Kwh'],

                orientation="v",

                title="<b>Grafico Consumo Por Unidade</b>",

                color\_discrete\_sequence=["#B22222"],

                template="plotly\_white",

            )

            fig\_ucquant.update\_layout(

                plot\_bgcolor="rgba(0,0,0,0)",

                xaxis=(dict(showgrid=False))

            )

            #criação de graficos por Tipo Cliente

            #grafico tipo cliente Facturado

            fig\_tipval = px.bar(

                tabdinamica2,

                x=tabdinamica2.index,

                y=['Valor Facturado'],

                orientation="v",

                title="<b>Grafico Valor Facturado Tipo Cliente</b>",

                color\_discrete\_sequence=["#5F9EA0"],

                template="plotly\_white",

            )

            fig\_tipval.update\_layout(

                plot\_bgcolor="rgba(0,0,0,0)",

                xaxis=(dict(showgrid=False))

            )

            #grafico tipo cliente Kwh

            fig\_tipquant = px.bar(

                tabdinamica2,

                x=tabdinamica2.index,

                y=['Kwh'],

                orientation="v",

                title="<b>Grafico Consumo Tipo Cliente</b>",

                color\_discrete\_sequence=["#B22222"],

                template="plotly\_white",

            )

            fig\_tipquant.update\_layout(

                plot\_bgcolor="rgba(0,0,0,0)",

                xaxis=(dict(showgrid=False))

            )

            #criação de graficos por Produto

            #grafico produto Facturado

            fig\_proval = px.bar(

                tabdinamica3,

                x=tabdinamica3.index,

                y=['Valor Facturado'],

                orientation="v",

                title="<b>Grafico Valor Facturado Produto</b>",

                color\_discrete\_sequence=["#5F9EA0"],

                template="plotly\_white",

            )

            fig\_proval.update\_layout(

                plot\_bgcolor="rgba(0,0,0,0)",

                xaxis=(dict(showgrid=False))

            )

            #grafico tipo cliente Kwh

            fig\_proquant = px.bar(

                tabdinamica3,

                x=tabdinamica3.index,

                y=['Kwh'],

                orientation="v",

                title="<b>Grafico Consumo Produto</b>",

                color\_discrete\_sequence=["#B22222"],

                template="plotly\_white",

            )

            fig\_proquant.update\_layout(

                plot\_bgcolor="rgba(0,0,0,0)",

                xaxis=(dict(showgrid=False))

            )

            #apresentar tabela dinamica na frame

            col1, col2, col3 = st.columns(3)

            with col1:

                st.header("Facturação Por Unidade")

                st.dataframe(tabdinamica, column\_config={"Unidade": st.column\_config.TextColumn("Unidade", width="medium"),

                                                         "Kwh": st.column\_config.NumberColumn("Kwh", width="small"),

                                                          "Valor Facturado": st.column\_config.NumberColumn("Valor Facturado", width="medium")})

            with col2:

                st.header("Facturação Por Tipo Cliente")

                st.dataframe(tabdinamica2, column\_config={"Tipo Cliente": st.column\_config.TextColumn("Tipo Cliente", width="medium"),

                                                          "Kwh": st.column\_config.NumberColumn("Kwh", width="small"),

                                                          "Valor Facturado": st.column\_config.NumberColumn("Valor Facturado", width="medium")})

            with col3:

                st.header("Facturação Por Produto")

                st.dataframe(tabdinamica3, column\_config={"Produto": st.column\_config.TextColumn("Produto", width="medium"),

                                                             "Kwh": st.column\_config.NumberColumn("Kwh", width="small"),

                                                            "Valor Facturado": st.column\_config.NumberColumn("Valor Facturado", width="medium")})

             #graficos po unidade comercial

            st.header("Grafico Unidade Comercial")

            #apresentar os graficos em colunas

            left\_column, right\_column = st.columns(2)

            left\_column.plotly\_chart(fig\_ucval, use\_container\_width=True)

            right\_column.plotly\_chart(fig\_ucquant, use\_container\_width=True)

            st.markdown("---")

            #graficos por tipo cliente

            st.header("Grafico Tipo de Cliente")

            #apresentar os graficos em colunas

            left\_column, right\_column = st.columns(2)

            left\_column.plotly\_chart(fig\_tipval, use\_container\_width=True)

            right\_column.plotly\_chart(fig\_tipquant, use\_container\_width=True)

            st.markdown("---")

            #graficos por Produto

            st.header("Grafico Produto")

            #apresentar os graficos em colunas

            left\_column, right\_column = st.columns(2)

            left\_column.plotly\_chart(fig\_proval, use\_container\_width=True)

            right\_column.plotly\_chart(fig\_proquant, use\_container\_width=True)

            st.markdown("---")

            st.header("Quadro Facturação")

            #organizar

            geral\_selection3 = geral\_selection3[['Unidade', 'CIL', 'Cliente', 'Cliente Conta', 'Tipo Cliente','Produto', 'Tipo Factura', 'Nº Factura',

                                                 'Data Processamento', 'Data Facturação', 'Tarifa', 'Valor Facturado', 'CONCEITO', 'Kwh', 'Valor Cons (ECV)']]

            st.dataframe(geral\_selection3, use\_container\_width=True, hide\_index=True)

            #opção de download dos dados em excel

            @st.cache\_data

            def convert\_df(df):

                #conversão do dado

                return df.to\_csv(sep=';', decimal=',', index=False).encode('utf-8-sig')

            csv = convert\_df(geral\_selection3)

            st.download\_button(

                label="Download Facturação",

                data=csv,

                file\_name='Script Facturação Tratado.csv',

                mime='text/csv'

            )

            #st.dataframe(facturas\_tratadas, use\_container\_width=True, hide\_index=True)

        #campo de Analise Matiridade

        if menu == "Analise Maturidade":

            st.title("Analise Maturidade")

            facturas\_maturidade = tratar\_factura()

            contratos\_geral = importar\_contratos()

            col1, col2, col3 = st.columns(3)

            #col1

            with col1:

                #filtrar região

                reg = st.multiselect(

                    "Definir Região: ",

                    options=facturas\_maturidade['Regiao'].unique(),

                )

                geral\_mat = facturas\_maturidade.query(

                    "`Regiao` == @reg"

                )

            #col2

            with col2:

                #filtar ano

                an = st.multiselect(

                    "Definir Ano: ",

                    options=geral\_mat['Ano'].unique(),

                )

                geral\_mat2 = geral\_mat.query(

                    "`Ano` == @an"

                )

            tab\_di = pd.pivot\_table(

                geral\_mat2,

                index=['CIL', 'Cliente Conta'],

                columns='Mês',

                values='Kwh',

                aggfunc='sum',

                fill\_value=0

            )

            #coluna de soma total na tabela dinamica

            tab\_di['Media Consumo Geral'] = tab\_di.sum(axis=1)

            #função se tendo em consideração a coluna de soma total

            tab\_di["Maturidade de consumo"] = tab\_di["Media Consumo Geral"].apply(classificar\_maturidade)

            tab\_di = tab\_di.reset\_index()

            #concatenação cil e conta

            tab\_di['CIL Conta'] = tab\_di['CIL'] + ' - ' + tab\_di['Cliente Conta']

            contratos\_geral['CIL Conta'] = contratos\_geral['CIL'] + ' - ' + contratos\_geral['CLI CONTA']

            contratos\_geral = contratos\_geral[["CIL Conta", "Regiao","Unidade", "NIP", "PORTA", "CGV",

                                               "Tipo\_Cliente", "NOME", "MORADA", "LOCALIDADE", "Produto", "COD TARIFA", "SEQ CONTR",

                                               "Estado Contrato", "DT CONTRATO", "DT INICIO", "DT BAIXA"]]

            contdi = pd.merge(tab\_di, contratos\_geral, on='CIL Conta', how='left')

            contdi.set\_index('CIL Conta', inplace=True)

            #definir botão de download numa coluna

            col1, col2 = st.columns(2)

            with col1:

                #opção de download dos dados em excel

                @st.cache\_data

                def convert\_df(df):

                    #conversão do dado

                    return df.to\_csv(sep=';', decimal=',', index=False).encode('utf-8-sig')

                csv = convert\_df(contdi)

                st.download\_button(

                    label="Download Facturação",

                    data=csv,

                    file\_name='Script Facturação Tratado.csv',

                    mime='text/csv'

                )

            #tabela dinamica numa coluna

            left\_column, midle\_column = st.columns(2)

            with col1:

                #criação de tabela dinamica para maturidade

                #quantidade de locais

                tabdicont = pd.pivot\_table(

                    contdi,

                    index=['Maturidade de consumo'],

                    values='CIL',

                    aggfunc='count',

                    fill\_value=0

                )

                #valor total consumido

                tabdicont2 = pd.pivot\_table(

                    contdi,

                    index=['Maturidade de consumo'],

                    values='Media Consumo Geral',

                    aggfunc='sum',

                    fill\_value=0

                )

                #merge duas tabelas de forma apresentar as informações de quantidade de consumo

                tb\_ge = pd.merge(tabdicont, tabdicont2, on='Maturidade de consumo', how='left')

                #apresentar as duas tabelas

                st.dataframe(tb\_ge)

            with col2:

                #definir filro para as maturidades

                #filtrar região

                matu = st.multiselect(

                    "Definir Maturidade: ",

                    options=contdi['Maturidade de consumo'].unique(),

                )

                geral\_mat = contdi.query(

                    "`Maturidade de consumo` == @matu"

                )

                geral\_mat.set\_index('CIL', inplace=True)

            #tabela filtrada

            st.dataframe(geral\_mat)

            #opção de download dos dados em excel

            @st.cache\_data

            def convert\_df(df):

                #conversão do dado

                return df.to\_csv(sep=';', decimal=',', index=False).encode('utf-8-sig')

            csv = convert\_df(geral\_mat)

            st.download\_button(

                label="Download Analise Maturidade",

                data=csv,

                file\_name='Script Facturação Por Maturidade.csv',

                mime='text/csv'

            )

    elif selected == "Dep. Gestão Contagem":

        st.title("Dep. Gestão Contagens")

        menu = st.radio(

        "Seleção", ["Importação", "Tratamento Itinerarios", "Analise Leituras"], horizontal=True

        )

        st.markdown("---")

        #campo de importação

        if menu == "Importação":

            #consultar itinerarios

            query = "SELECT \* FROM itinerarios"

            itinerarios2 = pd.read\_sql(query, engine)

            left\_column, middle\_column,right\_column = st.columns(3)

            with left\_column:

                #definir estrutura de data

                min\_date = datetime(2020, 1, 1)

                max\_date = datetime(2070, 12, 31)

                default\_start = datetime(2020, 1, 1)

                default\_end = datetime(2070, 12, 31)

                #configurar data a se usada

                dados = st.date\_input("Definir Data: ",

                                    value=(default\_start),

                                    min\_value=min\_date,

                                    max\_value=max\_date)

                data = pd.DataFrame({"Datas": [dados]})

            st.markdown("---")

            #carregar itinerarios

            upload\_file = st.file\_uploader("Importar Itinerarios", type="xlsx")

            if upload\_file:

                st.markdown("---")

                itinerario = pd.read\_excel(upload\_file, engine="openpyxl")

                itdata = pd.concat([data, itinerario], axis=1)

                itdata["Datas"] = itdata["Datas"].ffill()

                unidata = pd.merge(itdata, uc, on='Roteiro', how='left')

                unidata = unidata.loc[:,['Datas', 'Unidade','Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona', 'Rua', 'Cliente', 'Ponto de Medida', 'CIL',

                                                'Número', 'Marca', 'Função', 'Anterior', 'Atual', 'Anomalia']]

            #função de carregar it na base de dados

            if st.button("Guardar Itinerarios"):

                try:

                    unidata.to\_sql("itinerarios", con=engine, if\_exists="append", index=False)

                    st.success("Dados inseridos com sucesso!")

                except Exception as e:

                    st.error(f"Erro ao inserir dados: {e}")

            #lista de contadores de leitura remota

            query = "SELECT \* FROM contador"

            contador = pd.read\_sql(query, engine)

            upload\_file = st.file\_uploader("Importar Contadores", type="xlsx")

            if upload\_file:

                st.markdown("---")

                contadores = pd.read\_excel(upload\_file, engine="openpyxl")

            #carregar na base de dados

            if st.button("Guardar Contadores"):

                try:

                    contadores.to\_sql("contador", con=engine, if\_exists='replace', index=False)

                    st.success("Informação carregada com sucesso!")

                except Exception as e:

                    st.error(f'Ocorreu um erro na importação')

            #periodo

            query = "SELECT \* FROM periodo"

            periodo = pd.read\_sql(query, engine)

            upload\_file = st.file\_uploader("Importar Periodo", type="xlsx")

            if upload\_file:

                st.markdown("---")

                periodos = pd.read\_excel(upload\_file, engine="openpyxl")

            #carregar na base de dados

            if st.button("Carregar Periodo"):

                try:

                    periodos.to\_sql("periodo", con=engine, if\_exists='replace', index=False)

                    st.success("Informação carregada com sucesso!")

                except Exception as e:

                    st.error(f'Ocorreu um erro na importação')

            #leitura remota

            query = "SELECT \* FROM imtt"

            remota = pd.read\_sql(query, engine)

            upload\_file = st.file\_uploader("Importar Leituras", type="xlsx")

            if upload\_file:

                st.markdown("---")

                leituras = pd.read\_excel(upload\_file, engine="openpyxl")

            #carregar na base de dados

            if st.button("Carregar Leitura"):

                try:

                    leituras.to\_sql('imtt', con=engine, if\_exists='append', index=False)

                    st.success("Informação carregada com sucesso!")

                except Exception as e:

                    st.error(f'Ocorreu um erro na importação')

        #menu para tratar os dados de itinerarios e lançamento de leituras remotas

        if menu == "Tratamento Itinerarios":

            #st.title("Tratamento Itinerarios")

            #importar itinerarios

            query = "SELECT \* FROM itinerarios"

            it = pd.read\_sql(query, engine)

            # alterar codigo de função para descrição do codigo

            itfun = pd.merge(it, funcao, on='Função', how='left')

            itfun['Datas'] = pd.to\_datetime(itfun['Datas'])

            itfun['Ano'] = itfun['Datas'].dt.year

            itfun['Me'] = itfun['Datas'].dt.month

            itfun['Me'] = itfun['Me'].astype(str)

            itfun['Ano'] = itfun['Ano'].astype(str)

            itfun['Datas'] = itfun['Datas'].dt.date

            itfun2 = pd.merge(itfun, refmes, on='Me', how='left')

            #alterar ordem de apresentação

            itfun2 = itfun2.loc[:,['Ano', 'Mês', 'Descrição','Unidade','Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona', 'Rua', 'Cliente', 'Ponto de Medida', 'CIL',

                                                'Número', 'Marca', 'Função', 'Anterior', 'Atual', 'Anomalia']]

            col1, col2, col3 = st.columns(3)

            with col1:

                st.subheader("Definir Periodo de Leitura")

                #filtrar Ano

                ano = st.multiselect(

                    "Definir Ano: ",

                    options=itfun2['Ano'].unique(),

                )

                geralit = itfun2.query(

                    "`Ano` == @ano"

                )

                #filtrar Mês

                mes = st.multiselect(

                    "Definir Mês: ",

                    options=geralit['Mês'].unique(),

                )

                geralit2 = geralit.query(

                    "`Mês` == @mes"

                )

            #leitura remota

            query = "SELECT \* FROM imtt"

            leitura = pd.read\_sql(query, engine)

            leitura['Data Time'] = pd.to\_datetime(leitura['Data Time'])

            leitura['Ano'] = leitura['Data Time'].dt.year

            leitura['Me'] = leitura['Data Time'].dt.month

            leitura['Me'] = leitura['Me'].astype(str)

            leitura['Ano'] = leitura['Ano'].astype(str)

            leitura['Data Time'] = leitura['Data Time'].dt.date

            leitura2 = pd.merge(leitura, refmes, on='Me', how='left')

            #alterar ordem de apresentação

            leitura2 = leitura2.loc[:,['Ano', 'Mês', 'Descrição', 'Customer Name', 'Metering Points No.', 'Meter No.', 'Power Utility', 'Activeenergy(+)total(kWh)']]

            #junção de contador + função + cil

            leitura2['CIL/Contador/Função/Ano/Mês'] = leitura2['Metering Points No.'].astype(str) + '-' + leitura2['Meter No.'].astype(str) + '-' + leitura2['Descrição'] + '-' + leitura2['Ano'] + '-' + leitura2['Mês']

            #alterar ordem de apresentação

            leitura2 = leitura2.loc[:,['CIL/Contador/Função/Ano/Mês', 'Activeenergy(+)total(kWh)']]

            #junção de contador + função + cil

            geralit2['CIL/Contador/Função/Ano/Mês'] = geralit2['CIL'].astype(str) + '-' + geralit2['Número'].astype(str) + '-' + geralit2['Descrição'] + '-' + geralit2['Ano'] + '-' + geralit2['Mês']

            #cruzar leitura remota e itinerario

            leitre = pd.merge(geralit2, leitura2, on='CIL/Contador/Função/Ano/Mês', how='left')

            #alterar ordem

            leitre = leitre.loc[:,['Ano', 'Mês', 'Descrição','Unidade','Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona', 'Rua', 'Cliente', 'Ponto de Medida', 'CIL',

                                                'Número', 'Marca', 'Função', 'Anterior', 'Activeenergy(+)total(kWh)', 'Anomalia']]

            #reomear colunas

            leitre = leitre.rename(columns={'Activeenergy(+)total(kWh)': 'Atual',

                                            'Zona':'Zona ',

                                            'Rua':' Rua ',

                                            'Cliente':' Cliente'})

            with col1:

                #carregar na base de dados

                if st.button("Carregar Leitura"):

                    try:

                        leitre.to\_sql('base\_remota', con=engine, if\_exists='replace', index=False)

                        st.success("Informação carregada com sucesso!")

                    except Exception as e:

                        st.error(f'Ocorreu um erro na importação')

            #definir coluna para filtragem

            with col2:

                st.subheader("Definir Unidade E Roteiro")

                #filtrar Roteiro e itinerario

                unidade = st.multiselect(

                    "Definir Unidade: ",

                    options=leitre['Unidade'].unique(),

                )

                un = leitre.query(

                    "`Unidade` == @unidade"

                )

                roteiro = st.multiselect(

                    "Definir Roteiro: ",

                    options=un['Roteiro'].unique(),

                )

                rot = un.query(

                    "`Roteiro` == @roteiro"

                )

                iti = st.multiselect(

                    "Definir Itinerario: ",

                    options=rot['Itinerário'].unique(),

                )

                gerit = rot.query(

                    "`Itinerário` == @iti"

                )

            st.markdown("---")

            #alterar ordem

            gerit = gerit.loc[:,['Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona ', ' Rua ', ' Cliente', 'Ponto de Medida', 'CIL',

                                                'Número', 'Marca', 'Função', 'Anterior', 'Atual', 'Anomalia']]

            gerit.set\_index('Nr. Roteiro', inplace=True)

            st.dataframe(gerit, use\_container\_width=True)

            #opção de download dos dados em excel

            @st.cache\_data

            def convert\_df(df):

                #conversão do dado

                return df.to\_csv(sep=';', decimal=',', index=False).encode('utf-8-sig')

            csv = convert\_df(gerit)

            st.download\_button(

                label="Download Itinerarios",

                data=csv,

                file\_name='Itinerario.csv',

                mime='text/csv'

            )

        if menu == "Analise Leituras":

            st.title("Analise")

            query = "SELECT \* FROM base\_remota"

            base = pd.read\_sql(query, engine)

            query = "SELECT \* FROM contador"

            contador2 = pd.read\_sql(query, engine)

            query = "SELECT \* FROM periodo"

            periodo2 = pd.read\_sql(query, engine)

            #Definir remota

            contador2.loc[contador2['Nº Contador'] >= 0, "Tipo Contador"] = "Leitura Remota"

            # definir coluna com condições

            base['Diferença'] = base['Atual'] - base['Anterior']

            base.loc[base['Diferença'] < 0, "Analise Leitura"] = "Volta de Contador"

            base.loc[base['Diferença'] == 0, "Analise Leitura"] = "Contador Parado"

            base.loc[base['Diferença'] > 0, "Analise Leitura"] = "Local Com Consumo"

            #junção de contador + função + cil

            base['CIL/Contador'] = base['CIL'].astype(str) + '-' + base['Número'].astype(str)

            contador2['CIL/Contador'] = contador2['CIL'].astype(str) + '-' + contador2['Nº Contador'].astype(str)

            #alterar ordem

            contador2 = contador2.loc[:,['CIL/Contador', 'Tipo Contador']]

            #definir nos it's os locais que são de leitura remota

            contador\_geral = pd.merge(base, contador2, on='CIL/Contador', how='left')

            #alterar ordem de apresentação

            contador\_geral = contador\_geral.loc[:,

                    ['Unidade', 'Analise Leitura', 'Tipo Contador','Diferença', 'Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona ', ' Rua ', ' Cliente',

                    'Ponto de Medida', 'CIL', 'Número', 'Marca', 'Função','Anterior', 'Atual', 'Anomalia']]

            cont\_geral = pd.merge(contador\_geral, periodo2, on='CIL', how='left')

            #alterar ordem de apresentação

            cont\_geral = cont\_geral.loc[:,

                    ['Tipo Contador','Diferença', 'periodo', 'Analise Leitura', 'Unidade', 'Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona ', ' Rua ', ' Cliente',

                    'Ponto de Medida', 'CIL', 'Número', 'Marca', 'Função','Anterior', 'Atual', 'Anomalia']]

            #definir campos de pesquisa para tipo de leitura

            st.sidebar.header("Definir Tipo de Leitura:")

            tp\_leit = st.sidebar.multiselect(

                "Definir Contador",

                options=cont\_geral['Tipo Contador'].unique(),

            )

            geral\_selection = cont\_geral.query(

            "`Tipo Contador` == @tp\_leit"

            )

            #definir campos de pesquisa para periodo de leitura

            st.sidebar.header("Definir Periodo:")

            pe\_leit = st.sidebar.multiselect(

                "Definir Formato de Leitura",

                options=cont\_geral['periodo'].unique(),

            )

            geral\_selection2 = geral\_selection.query(

            "`periodo` == @pe\_leit"

            )

            #definir campos de pesquisa para periodo de leitura

            st.sidebar.header("Definir Leitura:")

            dif\_leit = st.sidebar.multiselect(

                "Definir Formato de Leitura",

                options=cont\_geral['Analise Leitura'].unique(),

            )

            geral\_selection3 = geral\_selection2.query(

            "`Analise Leitura` == @dif\_leit"

            )

            #alterar ordem de apresentação

            geral\_selection3 = geral\_selection3.loc[:,

                        ['Unidade', 'Nr. Roteiro', 'Roteiro', 'Ciclo', 'Itinerário', 'Zona ', ' Rua ', ' Cliente',

                        'Ponto de Medida', 'CIL', 'Número', 'Marca', 'Função','Anterior', 'Atual', 'Anomalia']]

            # remover coluna de index

            geral\_selection3.set\_index('Unidade', inplace=True)

            st.dataframe(geral\_selection3)

            #opção de download dos dados em excel

            @st.cache\_data

            def convert\_df(df):

                #conversão do dado

                return df.to\_csv(sep=';', decimal=',', index=False).encode('utf-8-sig')

            csv = convert\_df(geral\_selection3)

            st.download\_button(

                label="Download Analise",

                data=csv,

                file\_name='Analise de Leitura.csv',

                mime='text/csv'

            )

    elif selected == "Dep. Contratação":

        st.title("Dep. Contratação")

        menu = st.radio(

        "Seleção", ["Importação", "Cadastro", "Contratos"], horizontal=True

        )

        #campo de importação

        if menu == "Importação":

            #query importar contratos

            query = "SELECT \* FROM contratos"

            arquivo = st.file\_uploader("Importar Contratos", type=["txt"])

            if arquivo is not None:   # <- mesma indentação que a linha acima

                df = pd.read\_csv(

                    arquivo,

                    sep="\t",

                    header=None,

                    names=colunas2,

                    dtype=str,

                    encoding="utf-8-sig",

                    engine="python",

                    on\_bad\_lines="warn"

        )

                df['UC'] = df['UC'].astype(int)

                st.success("Arquivo carregado com sucesso!")

                contest = pd.merge(df, est\_contr, on='EST CONTR', how='left')

                contrtp = pd.merge(contest, tip\_client, on='TP\_CLI', how='left')

                contruc = pd.merge(contrtp, unidade, on='UC', how='left')

                contreg = pd.merge(contruc, regiao, on='Unidade', how='left')

                contrpro = pd.merge(contreg, produto, on='Prod', how='left')

                contrpro = contrpro[["EMP ID", "Regiao","Unidade", "USR ID", "NIP", "PORTA", "CIL", "SIS ABAST", "CGV",

                                            "CLI CONTA", "Tipo\_Cliente", "CAE ID", "TP USO", "NOME", "MORADA", "LOCALIDADE",

                                            "COD LOCAL", "Produto", "COD TARIFA", "SEQ CONTR", "Estado Contrato",

                                            "DT CONTRATO", "DT INICIO", "DT BAIXA"]]

                #função de carregar contratos na base de dados

                if st.button("Guardar Contrato"):

                    try:

                        contrpro.to\_sql("contratos", con=engine, if\_exists="replace", index=False)

                        st.success("Dados inseridos com sucesso!")

                    except Exception as e:

                        st.error(f"Erro ao inserir dados: {e}")

        if menu == "Cadastro":

            class User(*Base*):

                \_\_tablename\_\_= "cadastro"

                id = Column(Integer, primary\_key=True, autoincrement=True)

                nome = Column(String(255), nullable=False)  # Nome

                telefone = Column(String(255), nullable=False)  # Telefone

                movel = Column(String(255), nullable=False)  # movel

                cni = Column(String(20), nullable=False)  # CNI

                localidade = Column(String(255), nullable=False)  # Localidade

                morada = Column(String(255), nullable=False)  # Morada

                filename = Column(String(255), nullable=False)  # Nome do arquivo

                file\_data = Column(LargeBinary, nullable=False)  # PDF armazenado

                uploaded\_at = Column(DATETIME, server\_default=func.current\_timestamp())  # Data de upload

            # Criando o formulário

            with st.form("formulario"):

                nome = st.text\_input("Nome")

                telefone = st.text\_input("Telefone")

                movel = st.text\_input("Movel")

                cni = st.text\_input("CNI")

                localidade = st.text\_input("Localidade")

                morada = st.text\_input("Morada")

                uploaded\_file = st.file\_uploader("Carregar PDF", type=["pdf"])

                # Botão de submit

                submit\_button = st.form\_submit\_button("Cadastrar")

                if submit\_button:

                    if not nome or not telefone or not movel or not cni or not localidade or not morada or not uploaded\_file:

                        st.warning("Preencha todos os campos!")

                    else:

                        file\_data = uploaded\_file.read()

                        insert\_user(nome, telefone, movel, cni, localidade, morada, uploaded\_file.name, file\_data)

                        st.success("Cadastro realizado com sucesso!")

        if menu == "Contratos":

            st.title("Painel Administrativo")

    elif selected == "Administração":

        st.title("Painel Administrativo")

        st.subheader("📋 Lista de Usuários")

        # Consulta todos os usuários

        df\_usuarios = pd.read\_sql("SELECT id, username, nivel FROM usuarios", engine)

        st.dataframe(df\_usuarios, hide\_index=True)

        st.markdown("---")

        st.subheader("➕ Criar Novo Usuário")

        with st.form("form\_criar\_usuario"):

            novo\_user = st.text\_input("Nome de usuário")

            nova\_senha = st.text\_input("Senha", type="password")

            novo\_nivel = st.selectbox("Nível de acesso", ["admin", "gerente", "usuario", "contrato", "factura", "contagem"])

            submitted = st.form\_submit\_button("Criar")

            if submitted:

                try:

                    query = text("""

                        INSERT INTO usuarios (username, password, nivel)

                        VALUES (:username, :password, :nivel)

                    """)

                    session = SessionLocal()

                    try:

                        session.execute(query, {"username": novo\_user, "password": nova\_senha, "nivel": novo\_nivel})

                        session.commit()

                        st.success(f"Usuário '{novo\_user}' criado com sucesso!")

                    except Exception as e:

                        session.rollback()

                        st.error(f"Erro ao criar usuário: {e}")

                    finally:

                        session.close()

                    st.success(f"Usuário '{novo\_user}' criado com sucesso!")

                    #st.experimental\_rerun()

                except Exception as e:

                    st.error(f"Erro ao criar usuário: {e}")

        st.markdown("---")

        st.subheader("✏️ Editar ou Excluir Usuário")

        # Selecionar usuário para edição

        user\_ids = df\_usuarios["id"].tolist()

        user\_map = {f"{row['username']} ({row['nivel']})": row["id"] for \_, row in df\_usuarios.iterrows()}

        selecionado = st.selectbox("Selecionar usuário", list(user\_map.keys()))

        id\_selecionado = user\_map[selecionado]

        # Buscar dados do usuário selecionado

        dados\_user = df\_usuarios[df\_usuarios["id"] == id\_selecionado].iloc[0]

        with st.form("form\_editar\_usuario"):

            novo\_username = st.text\_input("Novo nome de usuário", value=dados\_user["username"])

            nova\_senha\_edit = st.text\_input("Nova senha (deixe em branco para não alterar)", type="password")

            novo\_nivel\_edit = st.selectbox("Novo nível", ["admin", "gerente", "usuario", "contrato", "factura", "contagem"], index=["admin", "gerente", "usuario", "contrato", "factura", "contagem"].index(dados\_user["nivel"]))

            col1, col2 = st.columns(2)

            with col1:

                atualizar = st.form\_submit\_button("Atualizar")

            with col2:

                deletar = st.form\_submit\_button("Excluir", type="primary")

            if atualizar:

                try:

                    if nova\_senha\_edit:

                        update\_query = f"""

                            UPDATE usuarios

                            SET username = '{novo\_username}', password = '{nova\_senha\_edit}', nivel = '{novo\_nivel\_edit}'

                            WHERE id = {id\_selecionado}

                        """

                    else:

                        update\_query = f"""

                            UPDATE usuarios

                            SET username = '{novo\_username}', nivel = '{novo\_nivel\_edit}'

                            WHERE id = {id\_selecionado}

                        """

                    session = SessionLocal()

                    try:

                        session.execute(text(update\_query))

                        session.commit()

                        st.success("Usuário atualizado com sucesso!")

                    except Exception as e:

                        session.rollback()

                        st.error(f"Erro ao atualizar: {e}")

                    finally:

                        session.close()

                        st.success("Usuário atualizado com sucesso!")

                        #st.experimental\_rerun()

                except Exception as e:

                    st.error(f"Erro ao atualizar: {e}")

            if deletar:

                try:

                    session = SessionLocal()

                    try:

                        session.execute(text(f"DELETE FROM usuarios WHERE id = {id\_selecionado}"))

                        session.commit()

                        st.success("Usuário excluído com sucesso!")

                    except Exception as e:

                        session.rollback()

                        st.error(f"Erro ao excluir: {e}")

                    finally:

                        session.close()

                        st.success("Usuário excluído com sucesso!")

                        #st.experimental\_rerun()

                except Exception as e:

                    st.error(f"Erro ao excluir: {e}")

    elif selected == "Definição":

        st.title("Configuração de Conta")

        st.subheader("Alterar Senha da Conta")

        # Recuperar nome do usuário logado

        username\_logado = st.session\_state.username

        with st.form("alterar\_senha\_form"):

            senha\_atual = st.text\_input("Senha Atual", type="password")

            nova\_senha = st.text\_input("Nova Senha", type="password")

            confirmar\_senha = st.text\_input("Confirmar Nova Senha", type="password")

            submitted = st.form\_submit\_button("Alterar Senha")

            if submitted:

                # Verificar se senha atual está correta

                query = text("SELECT password FROM usuarios WHERE username = :username")

                session = SessionLocal()

                try:

                    result = session.execute(query, {"username": username\_logado}).fetchone()

                    if result and result[0] == senha\_atual:

                        if nova\_senha == confirmar\_senha:

                            update\_query = text("UPDATE usuarios SET password = :nova WHERE username = :username")

                            session.execute(update\_query, {"nova": nova\_senha, "username": username\_logado})

                            session.commit()

                            st.success("Senha alterada com sucesso!")

                        else:

                            st.error("A nova senha e a confirmação não coincidem.")

                    else:

                        st.error("Senha atual incorreta.")

                except Exception as e:

                    st.error(f"Erro ao alterar a senha: {e}")

                finally:

                    session.close()

    if st.sidebar.button("Logout"):

        st.session\_state.logged\_in = False

        st.rerun()