admin_page.py

import streamlit as st import pandas as pd import db_utils import time

def render(): """Renderiza a página completa do painel de administração com tabela editável.""" st.title("Painel de Administração do Radar Pro") st.markdown("---")

```
# Seção 1: Métricas da Plataforma
st.subheader("Métricas Gerais")
stats = db_utils.get_platform_stats_admin()
if stats:
    col1, col2, col3 = st.columns(3)
    col1.metric("Total de Usuários", stats.get('total_users', 0))
    col2.metric("Total de Análises", stats.get('total_snapshots', 0))
    col3.metric("Total de Mercados", stats.get('total_markets', 0))
else:
    st.warning("Não foi possível carregar as estatísticas.")

st.markdown("---")

# Seção 2: Gerenciamento de Usuários com Edição
st.subheader("Gerenciamento de Usuários")
st.info("Clique diretamente nas células 'Créditos' ou 'Ativo?' para
editar e depois clique em 'Salvar Alterações'.")

users = db_utils.get_all_users_admin()
if not users:
    st.warning("Nenhum usuário encontrado.")
    return

df_users = pd.DataFrame(users)
df_users['last_sign_in_at'] =
pd.to_datetime(df_users['last_sign_in_at']).dt.tz_localize(None)
df_users.fillna(0, inplace=True) # Preenche todos os nulos com 0
df_users['is_active'] = df_users['is_active'].astype(bool)
df_users['analysis_credits'] = df_users['analysis_credits'].astype(int)

# Armazena o DataFrame original no estado da sessão para comparação
if 'original_df_users' not in st.session_state:
    st.session_state.original_df_users = df_users.copy()

# Configuração do data_editor
edited_df = st.data_editor(
    df_users,
    column_config={
        "id": st.column_config.TextColumn("User ID", disabled=True),
```

```
        "email": st.column_config.TextColumn("Email", disabled=True),
        "analysis_credits": st.column_config.NumberColumn("Créditos",
min_value=0, step=1),
        "is_active": st.column_config.CheckboxColumn("Ativo?"),
        "snapshot_count": st.column_config.NumberColumn("Nº Análises",
disabled=True),
        "market_count": st.column_config.NumberColumn("Nº Mercados",
disabled=True),
        "last_sign_in_at": st.column_config.DatetimeColumn("Último
Login", format="DD/MM/YYYY HH:mm", disabled=True),
    },
    use_container_width=True,
    hide_index=True,
    key="user_admin_editor"
)

if st.button("Salvar Alterações", type="primary",
use_container_width=True):
    # Encontra as diferenças entre o dataframe original e o editado
    diff = edited_df.compare(st.session_state.original_df_users)
    if diff.empty:
        st.toast("Nenhuma alteração detectada.", icon="🤷")
    else:
        with st.spinner("Salvando alterações..."):
            updates_made = 0
            for user_id in diff.index:
                update_data = {}
                # Pega as colunas alteradas para este usuário
                changed_cols =
diff.loc[user_id].dropna().to_dict().keys()
                for col, _ in changed_cols:
                    update_data[col] = edited_df.loc[user_id, col]

                if update_data:
                    if db_utils.update_user_profile_admin(user_id,
update_data):
                        updates_made += 1

            st.success(f"{updates_made} usuário(s) atualizado(s) com
sucesso!")
            # Atualiza o estado original para a próxima edição
            del st.session_state.original_df_users
            time.sleep(1)
            st.rerun()

st.markdown("---")
# A seção de supervisão de snapshots permanece a mesma
```

```
# api_calls.py
```

import streamlit as st import json import googlemaps import openai import time from tenacity import retry, stop_after_attempt, wait_exponential import db_utils

--- Configuração das APIs ---

try: gmaps = googlemaps.Client(key=st.secrets.google["maps_api_key"]) openai.api_key = st.secrets.openai["api_key"] except KeyError as e: st.error(f"Erro: A chave secreta {e} não foi encontrada no seu arquivo .streamlit/secrets.toml.") st.stop() except Exception as e: st.error(f"Ocorreu um erro inesperado ao configurar as APIs: {e}") st.stop()

--- Função de Chamada à IA ---

@retry(stop=stop_after_attempt(4), wait=wait_exponential(multiplier=2, min=4, max=30)) def call_chatgpt_with_retry(prompt: str): """Chama a API da OpenAI (ChatGPT) e retorna um objeto JSON.""" try: response = openai.chat.completions.create(model="gpt-3.5-turbo-1106", messages=[{"role": "system", "content": "Você é um consultor de negócios especialista. Responda APENAS com um objeto JSON válido, sem texto ou formatação adicional."}, {"role": "user", "content": prompt}], response_format={"type": "json_object"}) json_string = response.choices[0].message.content return json.loads(json_string) except Exception as e: print(f"ERRO DETALHADO NA CHAMADA DA OPENAI: {e}") raise e

--- Lógica de Prompts Customizados ---

def get_prompt_for_business_type(tipo_negocio, termo, localizacao, competidores_texto, avg_rating): """Retorna um prompt customizado com base no tipo de negócio selecionado."""

```
# Prompt base com as informações essenciais que todos os relatórios
terão
prompt_base = f"""
Analise o mercado para '{termo}' em '{localizacao}'.
Dados coletados:
- Concorrentes encontrados: {competidores_texto}
- Nota média da concorrência: {avg_rating:.1f}

Gere um relatório em formato JSON com as seguintes chaves obrigatórias:
- "sumario_executivo": (string) Um parágrafo conciso com a visão geral
do mercado.
- "analise_sentimentos": (objeto) Um objeto com chaves "Positivo",
"Negativo", e "Neutro", com valores de 0 a 100.
- "plano_de_acao": (array de 5 a 7 strings) Passos práticos e
acionáveis.
```

```
- "analise_demografica": (objeto) com as chaves "resumo",
"faixa_etaria", e "interesses_principais" (array).
- "dossies_concorrentes": (array de objetos) para os 5 principais
concorrentes, cada um com "nome", "posicionamento_mercado",
"pontos_fortes", e "pontos_fracos".
"""

# Dicionário com adições específicas para cada tipo de negócio
prompts_especificos = {
    "Restaurante, Bar ou Lanchonete": prompt_base + """
    Adicione também as seguintes chaves ao JSON, com insights
específicos para alimentação:
    - "analise_cardapio": (string) "Sugestões de pratos, bebidas ou
tipos de culinária que estão em alta ou ausentes na região."
    - "estrategia_delivery": (string) "Dicas para otimizar a presença em
apps como iFood/Rappi e estratégias de entrega própria."
    """,
    "Loja de Varejo (Roupas, Eletrônicos, etc.)": prompt_base + """
    Adicione também as seguintes chaves ao JSON, com insights
específicos para varejo:
    - "analise_mix_produtos": (string) "Análise sobre o mix de produtos
ideal para a localidade, sugerindo marcas, estilos ou categorias em
alta."
    - "estrategia_visual_merchandising": (string) "Dicas para a vitrine
e layout interno da loja para maximizar a atração de clientes e as
vendas."
    """,
    "Salão de Beleza ou Barbearia": prompt_base + """
    Adicione também as seguintes chaves ao JSON, com insights
específicos para serviços de beleza:
    - "servicos_diferenciados": (string) "Sugestão de 2 a 3 serviços,
técnicas ou produtos exclusivos que podem diferenciar o estabelecimento
da concorrência local."
    - "estrategia_agendamento": (string) "Análise sobre a melhor forma
de gerenciar agendamentos (app próprio, WhatsApp Business, etc.) para
fidelizar o público da região."
    """
}

# Retorna o prompt específico ou o prompt base se o tipo for "Genérico /
Outros"
return prompts_especificos.get(tipo_negocio, prompt_base)
```

--- Função Principal de Orquestração ---

def run_full_analysis(termo: str, localizacao: str, user_id: str, market_id: int, maps_api_key: str, tipo_negocio: str): snapshot_data = {"termo_busca": termo, "localizacao_busca": localizacao, "tipo_negocio": tipo_negocio}

```
progress_bar.progress(10, text="Buscando concorrentes no Google
Maps...")
query = f"{termo} em {localizacao}"
places_result = gmaps.places(query=query).get('results', [])
competidores = [{'name': p.get('name'), 'address':
p.get('formatted_address'), 'rating': p.get('rating', 0),
'user_ratings_total': p.get('user_ratings_total', 0), 'latitude':
p.get('geometry', {}).get('location', {}).get('lat'), 'longitude':
p.get('geometry', {}).get('location', {}).get('lng')} for p in
places_result[:10]]
snapshot_data['competidores'] = competidores

geocode_result = gmaps.geocode(localizacao)
if geocode_result: snapshot_data['location_geocode'] = geocode_result[0]
['geometry']['location']

progress_bar.progress(40, text=f"Consultando IA para análise de
'{tipo_negocio}'...")
competidores_texto = "\n".join([f"- {c.get('name')} (Nota:
{c.get('rating')})" for c in competidores])
avg_rating_list = [c['rating'] for c in competidores if
c.get('rating')]; avg_rating = sum(avg_rating_list) /
len(avg_rating_list) if avg_rating_list else 0

prompt_final = get_prompt_for_business_type(tipo_negocio, termo,
localizacao, competidores_texto, avg_rating)

ai_analysis = call_chatgpt_with_retry(prompt_final)
snapshot_data.update(ai_analysis)

progress_bar.progress(90, text="Salvando análise no banco de dados...")
final_json_string = json.dumps(snapshot_data, default=str)
db_utils.add_snapshot(market_id=market_id, user_id=user_id,
dados_json=final_json_string)

progress_bar.progress(100, text="Análise concluída com sucesso!")
time.sleep(1)
```

Função para SWOT (sem alterações)

@retry(stop_after_attempt(3), wait=wait_exponential(multiplier=1, min=2, max=10)) def generate_swot_analysis(data: dict): termo = data.get('termo_busca', 'N/A') localizacao = data.get('localizacao_busca', 'N/A') sumario = data.get('sumario_executivo', 'Sem sumário.') prompt_swot = f"""Baseado na seguinte análise de mercado para '{termo}' em '{localizacao}': " {sumario}". Crie uma Análise SWOT. Sua resposta deve ser um objeto JSON com quatro chaves: "strengths", "weaknesses", "opportunities", e "threats". Cada chave deve conter um array de 2 a 3 strings.""" swot_analysis = call_chatgpt_with_retry(prompt_swot) return swot_analysis

```
# auth_utils.py
```

import streamlit as st from supabase_client import supabase_client import db_utils

def login_user(email, password): """ Realiza o login do usuário, verifica se a conta está ativa, se é um administrador, e carrega os dados da sessão. """ try: # Autentica o usuário com email e senha response = supabase_client.auth.sign_in_with_password({ "email": email, "password": password }) user_id = response.user.id

```
    # Etapa crucial: Busca o perfil do usuário para verificar status e
créditos
    profile = db_utils.get_user_profile(user_id)

    # Verifica se o perfil existe e se a conta está ativa
    if not profile or not profile.get('is_active', False):
        supabase_client.auth.sign_out() # Garante o logout se a conta
estiver inativa
        return None, "Usuário desativado ou não encontrado. Contate o
suporte."

    # Se tudo estiver OK, armazena os dados na sessão do Streamlit
    st.session_state.user = response.user.dict()
    st.session_state.user_session = response.session
    st.session_state.is_admin = db_utils.is_user_admin(user_id)
    st.session_state.credits = profile.get('analysis_credits', 0)

    return st.session_state.user, None

except Exception as e:
    # Trata erros comuns de login
    if "invalid login credentials" in str(e).lower():
        return None, "Email ou senha inválidos."
    return None, f"Ocorreu um erro: {e}"
```

def signup_user(email, password): """Realiza o cadastro de um novo usuário.""" try: response = supabase_client.auth.sign_up({ "email": email, "password": password }) # O perfil do usuário (com créditos) é criado automaticamente por um Trigger no Supabase. return response.user.dict(), None

except Exception as e: if 'User already registered' in str(e): return None, "Este email já está cadastrado." return None, f"Erro no cadastro: {e}"

def logout_user(): """Realiza o logout e limpa completamente o estado da sessão.""" if 'user' in st.session_state and st.session_state.user: supabase_client.auth.sign_out()

```
# Limpa todas as chaves da sessão para garantir um estado limpo
keys_to_clear = list(st.session_state.keys())
for key in keys_to_clear:
    del st.session_state[key]

    # db_utils.py
```

import streamlit as st from supabase import create_client, Client from supabase_client import supabase_client from datetime import datetime import json

--- Funções de Usuário Padrão ---

def find_market_by_term_and_location(user_id: str, termo: str, localizacao: str): try: response = supabase_client.table('mercados_monitorados').select('id').eq('user_id', user_id).eq('termo', termo).eq('localizacao', localizacao).limit(1).single().execute() return response.data['id'] except Exception: return None

def get_user_markets(user_id: str): try: response = supabase_client.table('mercados_monitorados').select('*').eq('user_id', user_id).order('created_at', desc=True).execute() return response.data except Exception as e: raise e

def add_market(user_id: str, termo: str, localizacao: str, tipo_negocio: str): """Adiciona um novo mercado, incluindo o tipo de negócio, e retorna seu ID.""" try: # Verifica se já existe um mercado idêntico para evitar duplicatas existing_market = supabase_client.table('mercados_monitorados').select('id').eq('user_id', user_id).eq('termo', termo).eq('localizacao', localizacao).execute() if existing_market.data: # Se já existe, apenas retorna o ID existente return existing_market.data[0]['id']

```
    # Se não existe, insere um novo com o tipo de negócio
    response = supabase_client.table('mercados_monitorados').insert({
        'user_id': user_id,
        'termo': termo,
        'localizacao': localizacao,
        'tipo_negocio': tipo_negocio
    }).execute()
    return response.data[0]['id']
except Exception as e:
    raise e
```

def delete_market(market_id: int): try: supabase_client.table('mercados_monitorados').delete().eq('id', market_id).execute() except Exception as e: raise e

def add_snapshot(market_id: int, user_id: str, dados_json: str): try: supabase_client.table('snapshots_dados').insert({'mercado_id': market_id, 'user_id': user_id, 'dados_json': json.loads(dados_json)}).execute() except Exception as e: raise e

def get_latest_snapshot(market_id: int): try: response = supabase_client.table('snapshots_dados').select('*').eq('mercado_id', market_id).order('data_snapshot', desc=True).limit(1).single().execute() return response.data except Exception: return None

def get_all_snapshots(market_id: int): try: response = supabase_client.table('snapshots_dados').select('*').eq('mercado_id', market_id).order('data_snapshot', desc=False).execute() return response.data except Exception as e: st.error(f"Erro ao buscar histórico: {e}"); return []

def get_latest_snapshot_date(market_id: int): try: response = supabase_client.table('snapshots_dados').select('data_snapshot').eq('mercado_id', market_id).order('data_snapshot', desc=True).limit(1).single().execute() return datetime.fromisoformat(response.data['data_snapshot'].replace('Z', '+00:00')) except Exception: return None

def get_user_profile(user_id: str): try: response = supabase_client.table('profiles').select('*').eq('id', user_id).single().execute() return response.data except Exception: return None

def decrement_user_credits(user_id: str): try: current_profile = get_user_profile(user_id) if current_profile and current_profile['analysis_credits'] > 0: new_credits = current_profile['analysis_credits'] - 1 supabase_client.table('profiles').update({'analysis_credits': new_credits}).eq('id', user_id).execute() return new_credits return 0 except Exception as e: st.error(f"Erro ao atualizar créditos: {e}"); return None

--- Funções de Administrador ---

def _create_admin_client() -> Client | None: try: url = st.secrets["supabase"]["url"] service_key = st.secrets["supabase"]["service_key"] return create_client(url, service_key) except KeyError: st.error("A chave 'service_key' do Supabase não foi encontrada."); return None except Exception as e: st.error(f"Erro ao criar cliente de admin: {e}"); return None

def is_user_admin(user_id: str) -> bool: try: response = supabase_client.table('admins').select('user_id', count='exact').eq('user_id', user_id).execute() return response.count > 0 except Exception: return False

def get_all_users_admin(): admin_client = _create_admin_client() if not admin_client: return [] try: users_resp = admin_client.rpc('get_all_users_with_details').execute() return users_resp.data except Exception as e: st.error(f"Erro ao buscar usuários: {e}."); return []

def update_user_profile_admin(user_id: str, data: dict): admin_client = _create_admin_client() if not admin_client: return False try: admin_client.table("profiles").update(data).eq("id", user_id).execute() return True except Exception as e: st.error(f"Erro ao atualizar perfil: {e}"); return False

def get_platform_stats_admin(): admin_client = _create_admin_client() if not admin_client: return {} try: stats = {} users_resp = admin_client.rpc('count_total_users').execute() stats['total_users'] =

users_resp.data snapshots_resp = admin_client.table("snapshots_dados").select("id", count='exact').execute() stats['total_snapshots'] = snapshots_resp.count markets_resp = admin_client.table("mercados_monitorados").select("id", count='exact').execute() stats['total_markets'] = markets_resp.count return stats except Exception as e: st.error(f"Erro ao buscar estatísticas: {e}"); return {}

def get_all_snapshots_admin(limit=50): admin_client = _create_admin_client() if not admin_client: return [] try: response = admin_client.table('snapshots_dados').select('id, user_id, created_at, mercados_monitorados(termo, localizacao)').order('created_at', desc=True).limit(limit).execute() return response.data except Exception as e: st.error(f"Erro ao buscar snapshots de admin: {e}"); return []

def delete_snapshot_admin(snapshot_id: int): admin_client = _create_admin_client() if not admin_client: return False try: admin_client.table('snapshots_dados').delete().eq('id', snapshot_id).execute() return True except Exception as e: st.error(f"Erro ao deletar snapshot: {e}"); return False

```
# main.py
```

import streamlit as st import pandas as pd import json from datetime import datetime, timedelta import time import folium from streamlit_folium import st_folium

Módulos do projeto

import auth_utils import db_utils import api_calls import supabase_client import report_generator import admin_page

--- Configuração da Página e Estilos ---

st.set_page_config(page_title="Radar Pro - Inteligência de Mercado", page_icon="logo.png", layout="wide")

def load_css(file_name): try: with open(file_name) as f: st.markdown(f'', unsafe_allow_html=True) st.markdown("""""", unsafe_allow_html=True) except FileNotFoundError: st.warning(f"Arquivo de estilo '{file_name}' não encontrado.")

--- Funções de Processamento ---

def run_analysis_with_progress(termo: str, localizacao: str, user_id: str, market_id: int, maps_api_key: str, tipo_negocio: str): progress_bar = st.progress(0, text="Iniciando análise...") try: api_calls.run_full_analysis(termo, localizacao, user_id, market_id, progress_bar, maps_api_key, tipo_negocio) new_credits = db_utils.decrement_user_credits(user_id) if new_credits is not None: st.session_state.credits = new_credits st.success("Análise concluída e salva com sucesso!") time.sleep(2) st.rerun() except Exception as e: st.error(f"Ocorreu um erro crítico durante a análise: {e}") finally: progress_bar.empty()

--- Views (Páginas) da Aplicação ---

def login_page(): with st.container(): col1, col2, col3 = st.columns([1, 1.5, 1]) with col2: st.image("logo.png", width=200) st.title("Radar Pro: Inteligência de Mercado") st.subheader("Faça login para acessar seu dashboard") login_tab, signup_tab = st.tabs(["Login", "Cadastro"]) with login_tab: with st.form("login_form"): email = st.text_input("Email", key="login_email") password = st.text_input("Senha", type="password", key="login_password") if st.form_submit_button("Entrar", use_container_width=True): with st.spinner('Verificando...'): user, error = auth_utils.login_user(email, password) if user: st.rerun() else: st.error(f"Erro no login: {error}") with signup_tab: with st.form("signup_form"): email = st.text_input("Email", key="signup_email") password = st.text_input("Senha", type="password", key="signup_password") if st.form_submit_button("Cadastrar", use_container_width=True): with st.spinner('Criando conta...'): user, error = auth_utils.signup_user(email, password) if user: st.success("Cadastro realizado! Você já pode fazer o login.") time.sleep(3) st.rerun() else: st.error(f"Erro no cadastro: {error}")

def dashboard_page(): st.title("Dashboard de Mercados") st.divider() try: maps_api_key = st.secrets.google["maps_api_key"] except KeyError: st.error("Chave de API do Google Maps não configurada."); return

```
with st.expander("➕ Adicionar e Analisar Novo Mercado"):
    with st.form("new_market_form"):
        tipos_negocio = ["Genérico / Outros", "Restaurante, Bar ou
Lanchonete", "Loja de Varejo (Roupas, Eletrônicos, etc.)", "Serviços
Locais (Eletricista, Encanador, etc.)", "Salão de Beleza ou Barbearia",
"Academia ou Estúdio Fitness"]
        tipo_negocio_selecionado = st.selectbox("Selecione o Tipo de
Negócio:", tipos_negocio)
        termo = st.text_input("Termo de Busca Específico",
placeholder="Ex: Barbearia Clássica, Restaurante Japonês")
        localizacao = st.text_input("Localização", placeholder="Ex:
Copacabana, Rio de Janeiro")
        submitted = st.form_submit_button("Analisar Mercado",
use_container_width=True)
        if submitted:
            with st.spinner("Processando solicitação..."):
                if st.session_state.get('credits', 0) <= 0:
                    st.error("Créditos insuficientes.")
                elif termo and localizacao:
                    user_id = st.session_state.user['id']
                    market_id = db_utils.add_market(user_id, termo,
localizacao, tipo_negocio_selecionado)
                    run_analysis_with_progress(termo, localizacao,
user_id, market_id, maps_api_key, tipo_negocio_selecionado)
                else:
                    st.warning("Preencha o termo e a localização.")

st.divider()
```

```
st.subheader("Meus Mercados Monitorados")
user_markets = db_utils.get_user_markets(st.session_state.user['id'])
if not user_markets:
    st.info("Você ainda não adicionou nenhum mercado.")
else:
    for market in user_markets:
        with st.container(border=True):
            cols = st.columns([4, 2, 2, 2])
            with cols[0]:
                st.markdown(f"#### {market.get('termo', 'N/A')}")
                st.caption(f"Em: {market.get('localizacao', 'N/A')} |
Tipo: {market.get('tipo_negocio', 'N/A')}")
            with cols[1]:
                last_date =
db_utils.get_latest_snapshot_date(market['id'])
                st.caption("Última análise:" if last_date else
"Status:")
                st.markdown(f"**{last_date.strftime('%d/%m/%Y')}**" if
last_date else "**Ainda não analisado**")

            if cols[2].button("Ver Detalhes",
key=f"details_{market['id']}", use_container_width=True):
                with st.spinner("Carregando detalhes..."):
                    st.session_state.selected_market = market
                    st.session_state.page = 'details'
                    st.rerun()

            disable_reanalyze = False
            tooltip_message = "Reanalisar (custo: 1 crédito)"
            if last_date and datetime.now(last_date.tzinfo) - last_date
< timedelta(days=1):
                disable_reanalyze = True
                tooltip_message = "Reanálise disponível 24h após a
última análise."

            if cols[3].button("Reanalisar",
key=f"reanalyze_{market['id']}", use_container_width=True,
type="primary", disabled=disable_reanalyze, help=tooltip_message):
                with st.spinner("Iniciando reanálise..."):
                    if st.session_state.get('credits', 0) > 0:
                        run_analysis_with_progress(market['termo'],
market['localizacao'], st.session_state.user['id'], market['id'],
maps_api_key, market.get('tipo_negocio'))
                    else:
                        st.error("Créditos insuficientes.")

            if disable_reanalyze:
                cols[3].caption("Aguarde 24h")
```

def details_page(): if 'selected_market' not in st.session_state or st.session_state.selected_market is None: st.warning("Nenhum mercado selecionado. Redirecionando..."); st.session_state.page = 'dashboard'; time.sleep(1); st.rerun(); return market = st.session_state.selected_market latest_snapshot = db_utils.get_latest_snapshot(market['id']) if not latest_snapshot: st.error("Dados de análise não encontrados. Por favor, reanalise."); if st.button("⬅ Voltar ao Dashboard"): st.session_state.selected_market = None; st.session_state.page = 'dashboard'; st.rerun() return data = latest_snapshot.get('dados_json', {}) col1, col2 = st.columns([3, 1]) with col1: st.title(f"Análise Detalhada: {data.get('termo_busca', 'N/A')}") st.subheader(f"Localização: {data.get('localizacao_busca', 'N/A')}") st.caption(f"Tipo de Negócio Analisado: {data.get('tipo_negocio', 'Genérico / Outros')}") with col2: st.write(""); pdf_bytes = report_generator.gerar_relatorio_pdf(data, st.secrets.google["maps_api_key"]) if pdf_bytes: st.download_button(label="📄 Baixar Relatório PDF", data=pdf_bytes, file_name=f"Relatorio_{data.get('termo_busca')}.pdf", mime="application/pdf", use_container_width=True) if st.button("⬅ Voltar ao Dashboard"): with st.spinner(): st.session_state.selected_market = None; st.session_state.page = 'dashboard'; st.rerun() st.divider()

```
# --- LÓGICA DE ABAS SIMPLIFICADA E CORRIGIDA ---
(tab_geral, tab_plano, tab_extra, tab_demografia, tab_dossies,
 tab_mapa, tab_swot, tab_evolucao) = st.tabs([
    "📊 Visão Geral", "📝 Plano de Ação", "⭐ Insights do Setor", "👥
Demografia",
    " Dossiês", "🗺 Mapa", "💡 Análise SWOT", "📈 Evolução"
])

with tab_geral:
    st.header("Sumário Executivo");
st.write(data.get('sumario_executivo', 'N/A'))
    st.header("Análise de Sentimentos");
    sentimentos = data.get('analise_sentimentos', {});
    if sentimentos: df_sentimentos =
pd.DataFrame(list(sentimentos.items()), columns=['Sentimento',
'Pontuação']); st.bar_chart(df_sentimentos.set_index('Sentimento'))
    else: st.info("Nenhuma análise de sentimentos foi gerada.")

with tab_plano:
    st.header("Plano de Ação Sugerido"); plano =
data.get('plano_de_acao', []);
    if plano:
        for i, passo in enumerate(plano): st.markdown(f"**{i+1}.**
{passo}")
    else: st.info("Nenhum plano de ação foi gerado.")

with tab_extra:
    st.header("Insights Específicos do Setor")
    has_extra_data = False
    if "analise_cardapio" in data:
        st.subheader("Análise de Cardápio");
st.write(data.get("analise_cardapio")); has_extra_data = True
```

```
    if "estrategia_delivery" in data:
        st.subheader("Estratégia de Delivery");
st.write(data.get("estrategia_delivery")); has_extra_data = True
    if "analise_mix_produtos" in data:
        st.subheader("Análise de Mix de Produtos");
st.write(data.get("analise_mix_produtos")); has_extra_data = True
    if "estrategia_visual_merchandising" in data:
        st.subheader("Estratégia de Visual Merchandising");
st.write(data.get("estrategia_visual_merchandising")); has_extra_data =
True
    if "servicos_diferenciados" in data:
        st.subheader("Serviços Diferenciados");
st.write(data.get("servicos_diferenciados")); has_extra_data = True
    if "estrategia_agendamento" in data:
        st.subheader("Estratégia de Agendamento");
st.write(data.get("estrategia_agendamento")); has_extra_data = True

    if not has_extra_data:
        st.info("Nenhum insight específico para este setor foi gerado. A
análise foi do tipo 'Genérico'.")

with tab_demografia:
    st.header("Análise Demográfica do Público-Alvo");
    demografia = data.get('analise_demografica', {});
    if demografia:
        st.subheader("Resumo do Perfil");
st.write(demografia.get('resumo', 'N/A')); st.subheader("Faixa Etária
Principal"); st.info(demografia.get('faixa_etaria', 'N/A'));
st.subheader("Principais Interesses")
        for interesse in demografia.get('interesses_principais', []):
st.markdown(f"- {interesse}")
    else: st.info("Nenhuma análise demográfica foi gerada.")

with tab_dossies:
    st.header("Dossiês dos Principais Concorrentes");
    dossies = data.get('dossies_concorrentes', []);
    if dossies:
        for concorrente in dossies:
            with st.container(border=True):
                st.subheader(concorrente.get('nome', 'N/A'));
st.markdown(f"**Posicionamento:** *
{concorrente.get('posicionamento_mercado', 'N/A')}*"); col1, col2 =
st.columns(2)
                with col1: st.success(f"**Pontos Fortes:**
{concorrente.get('pontos_fortes', 'N/A')}")
                with col2: st.warning(f"**Pontos Fracos:**
{concorrente.get('pontos_fracos', 'N/A')}")
    else: st.info("Nenhum dossiê de concorrente foi gerado.")
```

```
with tab_mapa:
    st.header("Mapa Interativo da Concorrência");
    competidores = data.get('competidores', []); competidores_com_coords
= [c for c in competidores if c.get('latitude') and c.get('longitude')]
    if competidores_com_coords:
        avg_lat = sum(c['latitude'] for c in competidores_com_coords) /
len(competidores_com_coords); avg_lon = sum(c['longitude'] for c in
competidores_com_coords) / len(competidores_com_coords)
        mapa = folium.Map(location=[avg_lat, avg_lon], zoom_start=14)
        for comp in competidores_com_coords: folium.Marker(location=
[comp['latitude'], comp['longitude']], popup=f"<b>{comp['name']}</b>",
tooltip=comp['name'], icon=folium.Icon(color='red', icon='info-
sign')).add_to(mapa)
        st_folium(mapa, use_container_width=True)
    else: st.warning("Nenhum concorrente com dados de localização foi
encontrado.")

with tab_swot:
    st.header("Análise SWOT"); st.info("Esta análise é gerada sob
demanda. (custo: 1 crédito).");
    st.session_state.setdefault('swot_analysis', None)
    if st.button("Gerar Análise SWOT com IA", type="primary"):
        if st.session_state.get('credits', 0) > 0:
            with st.spinner("A IA está elaborando a matriz
estratégica..."):
                try:
                    st.session_state.swot_analysis =
api_calls.generate_swot_analysis(data)
                    new_credits =
db_utils.decrement_user_credits(st.session_state.user['id'])
                    if new_credits is not None: st.session_state.credits
= new_credits
                except Exception as e: st.error(f"Erro ao gerar análise
SWOT: {e}"); st.session_state.swot_analysis = None
        else: st.error("Créditos insuficientes.")
    if st.session_state.swot_analysis:
        swot = st.session_state.swot_analysis; col1, col2 =
st.columns(2)
        with col1: st.subheader("👍 Forças");
        for item in swot.get("strengths", []): st.markdown(f"- {item}");
st.subheader("👎 Fraquezas");
        for item in swot.get("weaknesses", []): st.markdown(f"- {item}")
        with col2: st.subheader("✨ Oportunidades");
        for item in swot.get("opportunities", []): st.markdown(f"-
{item}"); st.subheader("❗ Ameaças");
        for item in swot.get("threats", []): st.markdown(f"- {item}")
```

```
with tab_evolucao:
    st.header("Evolução Histórica dos Indicadores (KPIs)");
    all_snapshots = db_utils.get_all_snapshots(market['id'])
    if len(all_snapshots) < 2: st.info("É necessário ter pelo menos duas
análises para comparar a evolução.")
    else:
        kpi_data = [];
        for s in all_snapshots:
            snap_data = s.get('dados_json', {}); snap_date =
datetime.fromisoformat(s['data_snapshot'].replace('Z', '+00:00'))
            ratings = [c.get('rating', 0) for c in
snap_data.get('competidores', [])]; avg_rating = sum(ratings) /
len(ratings) if ratings else 0
            kpi_data.append({"Data": snap_date, "Nº de Concorrentes":
len(snap_data.get('competidores', [])), "Nota Média": avg_rating,
"Sentimento Positivo": snap_data.get('analise_sentimentos',
{}).get('Positivo', 0)})
        df_kpis = pd.DataFrame(kpi_data).set_index("Data")
        st.subheader("Tendência Geral dos KPIs");
st.line_chart(df_kpis); st.divider()
        st.subheader("Comparativo Detalhado entre Períodos")
        snapshots_dict =
{datetime.fromisoformat(s['data_snapshot'].replace('Z',
'+00:00')).strftime("%d/%m/%Y %H:%M"): s.get('dados_json', {}) for s in
all_snapshots}
        date_options = list(snapshots_dict.keys()); col1, col2 =
st.columns(2)
        date_from_str = col1.selectbox("Comparar de:", date_options,
index=0); date_to_str = col2.selectbox("Para:", date_options,
index=len(date_options)-1)
        if date_from_str and date_to_str and date_from_str <
date_to_str:
            data_from = snapshots_dict[date_from_str]; data_to =
snapshots_dict[date_to_str]
            ratings_to = [c.get('rating', 0) for c in
data_to.get('competidores', [])]; avg_to = sum(ratings_to) /
len(ratings_to) if ratings_to else 0
            sent_to = data_to.get('analise_sentimentos',
{}).get('Positivo', 0); ratings_from = [c.get('rating', 0) for c in
data_from.get('competidores', [])]
            avg_from = sum(ratings_from) / len(ratings_from) if
ratings_from else 0; sent_from = data_from.get('analise_sentimentos',
{}).get('Positivo', 0)
            m_col1, m_col2, m_col3 = st.columns(3)
            m_col1.metric(label="Nº de Concorrentes",
value=len(data_to.get('competidores', [])),
delta=len(data_to.get('competidores', [])) -
len(data_from.get('competidores', [])))
```

```
            m_col2.metric(label="Nota Média Concorrência", value=f"
{avg_to:.2f}", delta=f"{avg_to - avg_from:.2f}")
            m_col3.metric(label="Sentimento Positivo", value=f"
{sent_to}%", delta=f"{sent_to - sent_from}%")
        else: st.warning("Por favor, selecione um período de início
anterior ao período de fim.")

st.divider();
with st.expander("ℹ Sobre esta Análise: Como os dados são gerados?"):
    st.markdown("""Esta análise combina dados públicos e inteligência
artificial. As informações de concorrentes vêm da Plataforma Google
Maps, enquanto os insights estratégicos são elaborados pela OpenAI
(tecnologia do ChatGPT).""")
```

--- Roteador Principal ---

def main(): load_css("style.css"); st.session_state.setdefault('user', None); st.session_state.setdefault('selected_market', None); st.session_state.setdefault('is_admin', False); st.session_state.setdefault('page', 'dashboard'); st.session_state.setdefault('swot_analysis', None) if st.session_state.user: with st.sidebar: st.image("logo.png"); st.write(f"Bem-vindo, **{st.session_state.user.get('email')}**") st.write(f"Créditos: **{st.session_state.get('credits', 0)}**"); st.divider() if st.button("Meu Dashboard", use_container_width=True): with st.spinner(): st.session_state.page = 'dashboard'; st.session_state.selected_market = None; st.session_state.swot_analysis = None; st.rerun() if st.session_state.is_admin: if st.button("Painel de Administração", use_container_width=True, type="secondary"): with st.spinner(): st.session_state.page = 'admin'; st.rerun() st.divider(); if st.button("Sair", use_container_width=True): auth_utils.logout_user(); st.rerun() if st.session_state.page == 'details': details_page() elif st.session_state.page == 'admin' and st.session_state.is_admin: admin_page.render() else: dashboard_page() else: login_page()

if **name** == "**main**": main()

```
build-essential
```

libssl-dev libffi-dev python3-dev libpango-1.0-0

report_generator.py

import streamlit as st import base64 from io import BytesIO from xhtml2pdf import pisa import matplotlib.pyplot as plt import pandas as pd import requests from datetime import datetime from jinja2 import Environment, FileSystemLoader

def image_to_base64(path): """Converte uma imagem local para uma string Base64 para embutir no HTML.""" try: with open(path, "rb") as image_file: return base64.b64encode(image_file.read()).decode('utf-8') except FileNotFoundError: print(f"Arquivo de imagem não encontrado em: {path}") return None

def get_static_map_url(competidores, api_key): """Gera a URL para um mapa estático com marcadores para os concorrentes.""" if not api_key or not competidores: return ""

```
base_url = "https://maps.googleapis.com/maps/api/staticmap?"
params = {"size": "600x400", "maptype": "roadmap", "key": api_key}

markers = []
# Limita o número de marcadores para não exceder o limite de URL da API
for i, comp in enumerate(competidores[:18]):
    lat = comp.get('latitude')
    lon = comp.get('longitude')
    if lat and lon:
        # Usando números para os labels para economizar espaço
        markers.append(f"color:red|label:{i+1}|{lat},{lon}")

if markers:
    params["markers"] = markers

try:
    request = requests.Request('GET', base_url, params=params).prepare()
    # Retorna a URL pronta para ser usada na tag <img>
    return request.url
except Exception as e:
    print(f"Erro ao criar URL do mapa estático: {e}")
    return ""
```

def generate_sentiment_chart_base64(sentimentos): # ... (código desta função permanece o mesmo) if not sentimentos or not isinstance(sentimentos, dict): return "" try: df = pd.DataFrame(list(sentimentos.items()), columns=['Sentimento', 'Pontuação']) plt.style.use('seaborn-v0_8-whitegrid') fig, ax = plt.subplots(figsize=(6, 4)) colors = {'Positivo': '#2ca02c', 'Negativo': '#d62728', 'Neutro': '#ffaa00'} bar_colors = [colors.get(s, '#7f7f7f') for s in df['Sentimento']] bars = ax.bar(df['Sentimento'], df['Pontuação'], color=bar_colors) ax.set_title('Análise de Sentimentos', fontsize=16, weight='bold', color='#333') ax.set_ylabel('Pontuação (0-100)', fontsize=12); ax.set_ylim(0, 105) ax.spines['top'].set_visible(False); ax.spines['right'].set_visible(False) for bar in bars: yval = bar.get_height() ax.text(bar.get_x() + bar.get_width()/2.0, yval + 1, int(yval), va='bottom', ha='center') buf = BytesIO() plt.savefig(buf, format='png', bbox_inches='tight', transparent=True); plt.close(fig) buf.seek(0) return base64.b64encode(buf.getvalue()).decode('utf-8') except Exception as e: print(f"Erro ao gerar gráfico de sentimentos: {e}") return ""

def gerar_relatorio_pdf(data: dict, maps_api_key: str): """Gera o relatório PDF completo, agora incluindo o logo e o mapa estático.""" try: env = Environment(loader=FileSystemLoader('.')) template = env.get_template("template.html")

```
    # Prepara o contexto com todos os dados necessários para o template
    context = {
        'logo_base64': image_to_base64("logo.png"), # Converte o logo
        'termo_busca': data.get('termo_busca', 'N/A'),
```

```
        'localizacao_busca': data.get('localizacao_busca', 'N/A'),
        'sumario': data.get('sumario_executivo', 'N/A'),
        'plano_acao': data.get('plano_de_acao', []),
        'demografia': data.get('analise_demografica', {}),
        'dossies': data.get('dossies_concorrentes', []),
        'data_geracao': datetime.now().strftime('%d/%m/%Y'),
        'sentiment_chart_b64':
generate_sentiment_chart_base64(data.get('analise_sentimentos', {})),
        'static_map_url': get_static_map_url(data.get('competidores',
[]), maps_api_key),
        'competidores_lista': data.get('competidores', []) # Lista para
a legenda do mapa
    }

    html_out = template.render(context)

    pdf_bytes = BytesIO()
    pisa_status = pisa.CreatePDF(BytesIO(html_out.encode('UTF-8')),
dest=pdf_bytes)

    if pisa_status.err:
        st.error("Ocorreu um erro ao renderizar o PDF.")
        print(f"Erro PDF: {pisa_status.err}")
        return None

    pdf_bytes.seek(0)
    return pdf_bytes.getvalue()
except Exception as e:
    st.error(f"Erro ao preparar o relatório PDF: {e}")
    return None

    aiohappyeyeballs==2.6.1
```

aiohttp==3.12.13 aiosignal==1.3.2 altair==5.5.0 annotated-types==0.7.0 anyio==4.9.0 arabic-reshaper==3.0.0 asn1crypto==1.5.1 attrs==25.3.0 blinker==1.9.0 branca==0.8.1 cachetools==5.5.2 certifi==2025.6.15 cffi==1.17.1 charset-normalizer==3.4.2 click==8.2.1 colorama==0.4.6 contourpy==1.3.2 cryptography==45.0.4 cssselect2==0.8.0 cycler==0.12.1 deprecation==2.1.0 distro==1.9.0 folium==0.20.0 fonttools==4.58.4 frozenlist==1.7.0 gitdb==4.0.12 GitPython==3.1.44 google-ai-generativelanguage==0.6.15 google-api-core==2.25.1 google-api-python-client==2.173.0 google-auth==2.40.3 google-auth-httplib2==0.2.0 google-generativeai==0.8.5 googleapis-common-protos==1.70.0 googlemaps==4.10.0 gotrue==2.12.0 grpcio==1.73.0 grpcio-status==1.71.0 h11==0.16.0 h2==4.2.0 hpack==4.1.0 html5lib==1.1 httpcore==1.0.9 httplib2==0.22.0 httpx==0.28.1 hyperframe==6.1.0 idna==3.10 iniconfig==2.1.0 Jinja2==3.1.6 jiter==0.10.0 jsonschema==4.24.0 jsonschema-specifications==2025.4.1 kiwisolver==1.4.8 lxml==5.4.0 MarkupSafe==3.0.2 matplotlib==3.10.3 multidict==6.5.0 narwhals==1.43.1 numpy==2.3.1 openai==1.93.0 oscrypto==1.3.0 packaging==25.0 pandas==2.3.0 pillow==11.2.1 pluggy==1.6.0 postgrest==1.0.2 propcache==0.3.2 proto-plus==1.26.1 protobuf==5.29.5 pyarrow==20.0.0 pyasn1==0.6.1 pyasn1_modules==0.4.2 pycparser==2.22 pydantic==2.11.7 pydantic_core==2.33.2 pydeck==0.9.1 Pygments==2.19.2

pyHanko==0.29.1 pyhanko-certvalidator==0.27.0 PyJWT==2.10.1 pyparsing==3.2.3 pypdf==5.6.1 pytest==8.4.1 pytest-mock==3.14.1 python-bidi==0.6.6 python-dateutil==2.9.0.post0 pytrends==4.9.2 pytz==2025.2 PyYAML==6.0.2 realtime==2.4.3 referencing==0.36.2 reportlab==4.4.2 requests==2.32.4 rpds-py==0.25.1 rsa==4.9.1 six==1.17.0 smmap==5.0.2 sniffio==1.3.1 storage3==0.11.3 streamlit==1.46.0 streamlit_folium==0.25.0 StrEnum==0.4.15 supabase==2.15.3 supafunc==0.9.4 svglib==1.5.1 tenacity==9.1.2 tinycss2==1.4.0 toml==0.10.2 tornado==6.5.1 tqdm==4.67.1 typing-inspection==0.4.1 typing_extensions==4.14.0 tzdata==2025.2 tzlocal==5.3.1 uritemplate==4.2.0 uritools==5.0.0 urllib3==2.5.0 watchdog==6.0.0 webencodings==0.5.1 websockets==14.2 xhtml2pdf==0.2.17 xyzservices==2025.4.0 yarl==1.20.1

/* style.css */

/* Importa uma fonte mais moderna do Google Fonts */ @import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;600;700&display=swap');

/* Aplica a nova fonte a todo o corpo da aplicação */ *html, body, [class*="st-"] { font-family: 'Inter', sans-serif; }

/* Customiza os botões principais */ .stButton > button { border-radius: 20px; border: 1px solid #005f73; */* Cor primária do tema */ color: #005f73; background-color: #ffffff; transition: all 0.2s ease-in-out; */* Efeito de transição suave */ }

/* Efeito hover para os botões principais */ .stButton > button:hover { border-color: #0a9396; color: #0a9396; transform: scale(1.02); */* Leve aumento no hover */ }

/* Customiza os botões secundários (vermelhos/primários do Streamlit) */ .stButton > button[kind="primary"] { background-color: #d62728; */* Cor de destaque/alerta */ color: white; border: none; }

.stButton > button[kind="primary"]:hover { background-color: #e53935; border: none; transform: scale(1.02); }

/* Customiza os botões da barra lateral (sidebar) */ [data-testid="stSidebar"] .stButton > button { background-color: transparent; border: 1px solid #ddd; color: #444; }

[data-testid="stSidebar"] .stButton > button:hover { background-color: #f0f2f6; color: #005f73; border-color: #005f73; }

/* Customiza o expander (caixa de "Adicionar Novo Mercado") */ .st-expander { border-radius: 10px; border: 1px solid #e5e7eb; }

/* Customiza o contêiner dos mercados monitorados */ [data-testid="stVerticalBlock"] [data-testid="stVerticalBlock"] [data-testid="stVerticalBlock"] { border-radius: 10px; }

/* Customiza as abas (tabs) */ .stTabs [data-baseweb="tab-list"] { gap: 24px; }

supabase_client.py

import streamlit as st from supabase import create_client, Client

Cliente padrão para usuários normais

supabase_client: Client = None

try: supabase_url = st.secrets["supabase"]["url"] supabase_key = st.secrets["supabase"]["key"] supabase_client = create_client(supabase_url, supabase_key)

```
# Lógica de restauração da sessão do usuário
if "user_session" in st.session_state and st.session_state.user_session:
    supabase_client.auth.set_session(
        st.session_state.user_session.access_token,
        st.session_state.user_session.refresh_token
    )
```

except Exception as e: st.error(f"Erro ao inicializar o Supabase: {e}") st.stop()

```
<!-- template.html -->
```

Relatório gerado por Radar Pro | {{ data_geracao }}

```
{% if logo_base64 %}
    <img src="{{ logo_base64 }}" class="logo">
{% endif %}

<h1>Análise de Mercado: {{ termo_busca }}</h1>
<p class="subtitle"><strong>Localização:</strong> {{ localizacao_busca
}}</p>

<div class="section">
    <h2>Sumário Executivo</h2>
    <p>{{ sumario }}</p>
</div>

{% if plano_acao %}
<div class="section">
    <h2>Plano de Ação Sugerido</h2>
    <ul>
        {% for passo in plano_acao %}
            <li>{{ passo }}</li>
        {% endfor %}
    </ul>
</div>
{% endif %}

{% if sentiment_chart_b64 %}
<div class="section center">
    <h2>Análise de Sentimentos</h2>
```

```
    <img src="{{ sentiment_chart_b64 }}"
style="width: 80%; max-width: 500px;">
</div>
{% endif %}

{% if demografia %}
<div class="section">
    <h2>Análise Demográfica</h2>
    <p>{{ demografia.resumo }}</p>
</div>
{% endif %}

{% if static_map_url %}
<div class="section center" style="page-break-before: always;">
    <h2>Mapa da Concorrência</h2>
    <img src="{{ static_map_url }}" style="width: 100%; max-width:
600px;">
    <div class="map-legend">
        <h4>Legenda</h4>
        {% for i in range(competidores_lista|length) %}
            <p><strong>{{ i + 1 }}:</strong> {{
competidores_lista[i].name }}</p>
        {% endfor %}
    </div>
</div>
{% endif %}

{% if dossies %}
<div class="section" style="page-break-before: always;">
    <h2>Dossiês dos Concorrentes</h2>
    {% for d in dossies %}
        <div class="dossie-card">
            <h4>{{ d.get('nome', 'N/A') }}</h4>
            <p><strong>Posicionamento:</strong> {{
d.get('posicionamento_mercado', 'N/A') }}</p>
            <p><strong>Pontos Fortes:</strong> {{ d.get('pontos_fortes',
'N/A') }}</p>
            <p><strong>Pontos Fracos:</strong> {{ d.get('pontos_fracos',
'N/A') }}</p>
        </div>
    {% endfor %}
</div>
{% endif %}
```