Como voce é um expert em desenvolvimento de soluções para WEB, preciso que me ajude a arrumar os erros encontrados em paginas Dashboards de demonstração de resultados de

Como o meu conhecimento em escrita de codificação de script é mínima, preciso que sua orientação e desenvolvimento seja efetuada com o envio da codificação de 100% do codigo.

2. Arquitetura do Sistema

O sistema é composto por três partes principais:

Fonte de Dados: 2 Planilha do Googles Sheets, que recebe as informações formulários (Inscrição e check-in) via api do N8N, contendo diversas informações.

Backend e Base de Dados: Um projeto Supabase (PostgreSQL) com as tabelas principais: leads, perguntas, respostas\_leads.

Frontend: Um painel de administração (Dashboard) construído em Next.js e hospedado na Vercel, que consome os dados do Supabase para exibir as análises (Dashboards).

3. Desafios e Solução Definida

O desenvolvimento passou por vários desafios, incluindo problemas de conexão, lógica de separação de respostas e, principalmente, a forma de guardar e ler os dados. A solução final e definitiva, definida pelo utilizador, é a seguinte:

Papel de Cada Componente na Nova Arquitetura

Google Apps Script (O Tradutor Inteligente):

Função: Atua como um pré-processador. Ele lê a planilha "bagunçada".

Processo: Para cada resposta, ele encontra a pergunta correspondente na tabela perguntas do Supabase para obter o seu ID e a sua classe ('Score' ou 'Perfil').

Resultado: Ele monta dois pacotes JSON separados: um para as respostas de Score e outro para as de Perfil, ambos usando o ID da pergunta como chave. Em seguida, envia estes pacotes para uma função no Supabase, que fará a gravação nas tabelas do Supabase.

Essa Função: Atua como o "cérebro" do processamento.

Processo: Recebe os dois pacotes JSON já separados. Guarda cada um na sua coluna respetiva (respostas e respostas\_perfil). Em seguida, recalcula o score total do lead do zero, lendo apenas as respostas da coluna respostas.

Resultado: Garante que a pontuação das respostas nunca seja inflados e que os dados de perfil e score sejam corretos.

**Título:** Especificação Técnica do Módulo de Importação em 2 Passos

**Visão Geral:** O módulo permite a importação de dados em massa a partir de dois arquivos CSV distintos. Ele é projetado para ser robusto, não travar a interface do usuário e fornecer logs detalhados sobre o processo.

**Passo 1: Importação de Inscrições**

* **Entrada:** Um arquivo CSV delimitado por:
* **Processo:**
  1. O script lê o arquivo linha por linha (Papa.parse com a opção step).
  2. Os cabeçalhos são normalizados para minúsculas.
  3. Para cada linha, o e-mail é validado (presença e formato). Linhas inválidas são ignoradas e um aviso é gerado no log.
  4. E-mails válidos são convertidos para minúsculas para evitar duplicidade e limpos para evitar sujeiras.
  5. Os dados são agrupados em um Map para garantir a unicidade do e-mail.
  6. Ao final da leitura, os dados são inseridos/atualizados na tabela leads em lotes de 400 para não sobrecarregar o banco.

**Passo 2: Importação de Check-ins e Respostas**

* **Entrada:** Um arquivo CSV de check-ins delimitado por:
* **Processo (Dentro do Web Worker):**
  1. **Processamento do CSV:** Lê o arquivo CSV linha por linha. Para cada linha:
     + Normaliza o e-mail e os cabeçalhos.
     + Encontra o lead correspondente na lista de leads carregada. Se não encontrar, gera um aviso no log.
     + Cria um objeto de dados para o lead encontrado.
     + **Cálculo do Score:** Inicia o score do lead. Itera sobre as colunas da linha do CSV; para cada coluna que corresponde a uma pergunta, valida a resposta e soma o peso correspondente ao score. Gera avisos detalhados para respostas inválidas, mostrando as opções válidas.
     + **Formato de Respostas:** Agrega todas as respostas do lead em um único objeto JSON.
     + **Data de Check-in:** Lê a coluna data\_checkin ou Submitted at no formato D/M/AAAA H:mm:ss ou DD/MM/AAAA hh:mm:ss ou AAAA/MM/DD hh:mm:ss
  2. **Gravação no Banco (em Lotes):**
     + Após processar todo o arquivo, o Worker atualiza a tabela leads em lotes, sobrescrevendo os campos score e check\_in\_at.
     + Em seguida, ele insere/atualiza os dados na tabela respostas\_leads, salvando uma única linha por lead com as respostas no campo jsonb.
  3. **Finalização:** Envia uma mensagem de conclusão para a página principal, que exibe o alerta de sucesso e reabilita os botões.

**Prompt de Continuidade: Especificação Técnica e Funcional da Ferramenta de Importação**

**1. Objetivo Principal**

O objetivo deste projeto foi desenvolver e refinar uma ferramenta de importação de dados de leads (Inscrições e Check-ins) para uma base de dados Supabase, através de uma página de gestão construída em Next.js (ImportacaoPage). A ferramenta deveria ser capaz de processar diferentes formatos de ficheiros CSV, incluindo lançamentos antigos e novos, calcular score de vendas e mql\_score de perfil, e lidar com as particularidades de formatação dos ficheiros exportados.

**2. Arquitetura Final do Sistema de Importação**

O fluxo de importação de ficheiros de Inscrição ou Check-in segue a seguinte arquitetura:

1. **Frontend (Página de Importação em Next.js):**
   * O utilizador seleciona um lançamento de destino e um ficheiro CSV.
   * **Importante:** A página **não** faz mais o processamento pesado. A sua única responsabilidade é ler o conteúdo do ficheiro CSV como **texto bruto** e enviá-lo, juntamente com o launch\_id, para uma Supabase Edge Function.
   * Esta abordagem demonstrou ser a mais robusta, evitando problemas de cache do navegador, inconsistências de bibliotecas de frontend e erros de processamento no lado do cliente.
2. **Backend (Supabase Edge Function import-checkin-csv):**
   * Atua como o "cérebro" da lógica de negócio. É escrita em TypeScript (Deno).
   * Recebe o texto bruto do CSV.
   * Usa a biblioteca PapaParse para ler e interpretar o ficheiro, lidando com delimitadores (;) e cabeçalhos.
   * Contém uma lógica robusta para identificar as colunas de e-mail e data, independentemente do nome exato do cabeçalho (email vs. e-mail, Submitted at vs. data\_checkin), garantindo **retrocompatibilidade**.
   * Lida com formatos de ficheiro complexos, onde uma única pergunta de múltipla escolha é representada por várias colunas (formato "pergunta (opção)"). A função sabe identificar estas colunas e extrair a resposta correta se o valor for VERDADEIRO.
   * Trata corretamente as **datas**, convertendo-as de formatos comuns (ex: DD/MM/YYYY HH:mm:ss) para o formato padrão ISO, ajustando o fuso horário para UTC-3 (São Paulo) para garantir que os relatórios por dia sejam precisos.
   * Após processar todas as linhas e respostas, ela monta um payload e chama a função de base de dados bulk\_process\_survey\_recalc para efetuar a escrita no banco de dados.
3. **Banco de Dados (Supabase PostgreSQL - Função bulk\_process\_survey\_recalc):**
   * Atua como a "mão" que escreve os dados de forma segura. É escrita em plpgsql.
   * A função é **idempotente**, o que significa que reprocessar o mesmo ficheiro não causa erros nem duplica dados.
   * Recebe o payload da Edge Function, que já contém as respostas separadas por score\_answers e profile\_answers.
   * **Lógica de Escrita:**
     + Cria os leads na tabela public.leads se eles não existirem (INSERT ... ON CONFLICT).
     + Calcula o score de vendas (baseado nas score\_answers) e o mql\_score de perfil (baseado nas profile\_answers).
     + Salva as respostas nas colunas corretas (respostas e respostas\_perfil) na tabela public.respostas\_leads.
     + Atualiza a linha do lead na tabela public.leads com os scores calculados e a data do check-in.

**3. Desafios e Particularidades Tratadas**

Durante o desenvolvimento, vários desafios foram encontrados e resolvidos:

* **Formatos de CSV Inconsistentes:** A Edge Function foi construída para lidar com cabeçalhos que podem conter quebras de linha, aspas e espaços extras.
* **Perguntas de Múltipla Escolha (Multi-coluna):** A Edge Function agora sabe interpretar o formato Pergunta (Opção) -> VERDADEIRO/FALSO para consolidar múltiplas colunas numa única resposta.
* **Datas e Fusos Horários:** A Edge Function converte explicitamente as datas do CSV para UTC, considerando o fuso horário de origem (UTC-3), o que resolveu o problema de registos aparecerem no dia errado nos relatórios.
* **Scores Decimais vs. Inteiros:** Foi implementada a capacidade de usar scores decimais (ex: 0.5).
  + As colunas score e mql\_score na tabela leads foram alteradas para o tipo NUMERIC.
  + O formulário de edição de perguntas (QuestionForm.tsx) foi ajustado para aceitar tanto vírgula (padrão do navegador no Brasil) como ponto, e converter para o formato de ponto antes de salvar.
* **Dependências de "Views" no Banco de Dados:** Foi necessário apagar e recriar uma view (leads\_com\_tipo\_trafego) para permitir a alteração do tipo da coluna score de INTEGER para NUMERIC.

**4. Funções e Código Fonte Relevantes**

* **Supabase Edge Function:** import-checkin-csv (versão 23)
* **Supabase Database Function:** bulk\_process\_survey\_recalc (versão 4)
* **Componente Frontend Principal:** ferramenta/importacao/page.tsx
* **Ferramentas de Teste:** Foram criadas funções SQL (clear\_leads\_from\_launch, reset\_checkin\_data\_for\_launch) e botões na ImportacaoPage para permitir a limpeza total ou parcial dos dados de um lançamento para facilitar os testes.

AQUI

Como voce é um expert em desenvolvimento de soluções para WEB, preciso que me ajude a arrumar os erros encontrados em páginas Dashboards de demonstração de resultados de uma solução desenvolvida para acompanhar os processos referente a Lançamentos Classicos de venda on line de cursos para professores.  
Esse processo é composto de inscrição de interessados capturados por banners, que chegam a uma pagina de inscrição onde viram LEADS, após o preenchimento de uma ficha. Após virarem Leads, recebem uma pesquisa(Check in) para descobrimos o perfil desse lead, que servirão para orientação de área de trafego e de pré-vendas, através de pontuação de suas respostas.  
As perguntas feitas na pesquisa, tem em seu cadastro (tabela perguntas) um campo que indica qual tipo de pergunta se enquadra (score ou perfil).

Esses valores, assim como as respostas já foram gravadas na tabela resposta\_leads, e agora precisamos tratar as respostas de classe perfil, já que as da classe score, já estão tratadas e possuem dashboards para sua exibição, exceto uma pagina que esta apresentando uma falha na não apresentação dos detalhes das respostas.  
Existe uma outra dashboard, que precisa ser alterada para mostrar os totais dos valores apuradas nas respostas de classe perfil.

Como o meu conhecimento em escrita de codificação de script é mínima, preciso que sua orientação e desenvolvimento seja efetuada com o envio da codificação de 100% do codigo.   
Esta documentação foi confeccionada pelo outro chat que estava desenvolvendo, mas por motivo do chat estar muito grande, começou a perder razão de continuar o desenvolvimento.

**Documentação Técnica e Funcional: Evolução dos Dashboards**

**Versão:** 1.1 (Corrigida) **Data:** 09 de Agosto de 2025

**1. Visão Geral e Objetivos do Projeto**

Este documento estabelece o plano de ação para a conclusão e evolução dos dashboards de análise de leads. O processo de importação de dados (backend) está agora estabilizado e funcional. O foco deste projeto é a camada de visualização (frontend), para garantir que os dados corretos sejam exibidos de forma clara e para desenvolver novas ferramentas de análise.

Os objetivos específicos são:

1. **Corrigir a Divergência:** Unificar a exibição de dados entre os dashboards de Score, garantindo que ambos mostrem as respostas detalhadas.
2. **Evoluir para MQL:** Implementar o cálculo e a exibição do MQL Score com base nas respostas de perfil.
3. **Criar uma Visão Macro:** Desenvolver um novo dashboard de resumo geral que consolide as métricas de todos os lançamentos.

**2. Fases do Projeto**

O projeto será executado em três fases sequenciais e bem definidas.

**Fase 1: Correção do Dashboard "Perfil de Score por Respostas"**

* **Objetivo Funcional:** Fazer com que o dashboard "Perfil de Score por Respostas" exiba os gráficos de barras com as respostas detalhadas para cada categoria de score (Quente, Morno, etc.), tal como o dashboard "Análise de Respostas por Score" já faz. A funcionalidade deve ser idêntica em ambos.
* **Diagnóstico Técnico:** A importação de dados está a gravar corretamente os JSONs nas colunas respostas (Score) e respostas\_perfil (Perfil). A divergência entre os dashboards prova que o problema está no código **frontend** do dashboard "Perfil de Score por Respostas". A sua função de busca de dados (fetchBreakdownData) está a chamar uma função SQL (get\_score\_profile\_by\_answers) que, muito provavelmente, só está a ler e a processar a coluna respostas\_perfil, ignorando a coluna respostas.
* **Plano de Ação Técnico:**
  1. **Análise do Código:** Precisamos do código completo do ficheiro da página **dashboard-perfil-score/page.tsx**.
  2. **Correção da Lógica de Leitura:** Iremos analisar a função get\_score\_profile\_by\_answers (no Supabase) e o código da página que a chama. A correção passará por modificar a função SQL para que ela leia e processe **ambas** as colunas (respostas e respostas\_perfil) ou criar uma nova função dedicada para as respostas de Score.
  3. **Entrega:** Devolveremos o ficheiro page.tsx 100% corrigido e a função SQL necessária para unificar a visualização.

**Fase 2: Implementação do MQL Score**

* **Objetivo Funcional:** Calcular e exibir o **MQL Score** (cuja coluna mql\_score já existe na tabela leads). Este score será calculado com base em respostas específicas de 'Perfil' e deverá ser visível no dashboard "Análise de Respostas de Perfil".
* **Plano de Ação Técnico:**
  + **Etapa 1: Backend - O Cálculo do Score**
    1. **Marcar Perguntas MQL:** Adicionaremos uma nova coluna booleana chamada is\_mql à sua tabela **perguntas**. Esta coluna funcionará como uma "etiqueta" para identificar quais perguntas de perfil devem ser usadas no cálculo do MQL Score.
    2. **Criar a Função de Cálculo:** Desenvolveremos uma nova função SQL chamada calcular\_mql\_score\_lancamento(p\_launch\_id UUID). Esta função será distinta da processar\_linha\_csv (que calcula o score de vendas) e terá a seguinte missão:
       - Selecionar todos os leads do lançamento especificado.
       - Para cada lead, ler o seu JSON da coluna respostas\_perfil.
       - Dentro do JSON, procurar pelas respostas às perguntas que estão "etiquetadas" como is\_mql = true.
       - Somar os pesos dessas respostas.
       - Gravar o resultado na coluna mql\_score da tabela leads.
    3. **Entrega:** Entregaremos o código SQL completo para a alteração da tabela e para a criação da nova função de cálculo.
  + **Etapa 2: Frontend - A Exibição do Score**
    1. **Análise do Código:** Precisamos do código completo do ficheiro da página **dashboard-analise-perfil/page.tsx** (ou o nome correspondente).
    2. **Alteração da Interface:** Iremos modificar o código para adicionar uma nova secção ou um novo KPI card para exibir o MQL Score médio do lançamento e, possivelmente, um novo gráfico que mostre a distribuição de leads por faixas de MQL Score.
    3. **Entrega:** Devolveremos o ficheiro da página 100% atualizado com a nova visualização.

**Fase 3: Desenvolvimento do Dashboard de Resumo Geral**

* **Objetivo Funcional:** Criar uma nova página de alto nível chamada "Visão Geral do Projeto" que consolide os dados de todos os lançamentos, permitindo uma análise macro e a comparação de performance entre eles.
* **Plano de Ação Técnico:**
  + **Etapa 1: Backend - Funções de Agregação de Dados**
    1. Desenvolveremos um conjunto de novas funções SQL otimizadas para agregar dados de toda a base:
       - get\_resumo\_geral\_kpis(): Retorna os totais de inscrições, check-ins e a taxa de conversão média de todos os tempos.
       - get\_performance\_por\_lancamento(): Retorna uma lista de todos os lançamentos com as suas métricas principais (inscrições, check-ins, taxa de check-in).
       - get\_distribuicao\_score\_agregada(): Retorna a contagem total de leads em cada categoria de score (Quente, Morno, etc.) somando todos os lançamentos.
  + **Etapa 2: Frontend - Construção da Nova Página**
    1. Criaremos um novo ficheiro de página (ex: app/dashboard/visao-geral/page.tsx).
    2. Este ficheiro irá conter:
       - **Cards de KPI:** Para exibir os resultados da função get\_resumo\_geral\_kpis().
       - **Gráfico de Barras:** Usando uma biblioteca como a Recharts, para visualizar a performance por lançamento.
       - **Gráfico de Pizza (ou Barras):** Para mostrar a distribuição agregada de scores.
       - **Tabela Detalhada:** Uma tabela com os dados de cada lançamento, permitindo ordenação.
    3. **Entrega:** Entregaremos o **código 100% completo** para a nova página e as funções SQL de backend necessárias.