**Documentação Técnica e Funcional: Módulo de Importação de Leads**

**Versão:** 1.0 **Data:** 19 de Agosto de 2025 **Autor:** Gemini

**1. Visão Geral Funcional**

Este documento detalha o sistema de importação de dados de contingência para a plataforma de acompanhamento de lançamentos de cursos online. O objetivo do sistema é permitir que o administrador importe em massa dados de **Inscrições** e **Check-ins** a partir de ficheiros CSV, garantindo a integridade e o processamento correto dos dados para análise de performance.

O fluxo de dados de um lead no sistema é o seguinte:

1. **Inscrição (Lead):** Um potencial cliente inscreve-se no evento de lançamento. Os seus dados (email, nome, telefone, UTMs) são capturados. Este processo cria um registo na tabela leads.
2. **Check-in (Pesquisa de Perfil):** O lead preenche uma pesquisa. As suas respostas são usadas para calcular dois scores principais:
   * **Score:** Mede o potencial de compra geral.
   * **MQL Score (Perfil):** Mede o alinhamento do perfil do lead com o público-alvo.
3. **Venda (Comprador):** A venda ocorre numa plataforma externa. Os dados dos compradores são importados posteriormente para marcar os leads correspondentes (is\_buyer = true) e calcular um score específico de comprador.

**2. Arquitetura Técnica**

O sistema é construído sobre uma arquitetura moderna utilizando os seguintes componentes:

* **Frontend:** React (Next.js) para a interface do utilizador, onde os uploads são iniciados.
* **Backend & Base de Dados:** Supabase (PostgreSQL) para armazenamento de dados. As tabelas principais envolvidas são leads, lancamentos, perguntas, e respostas\_leads.
* **Storage:** Supabase Storage é usado para armazenar temporariamente os ficheiros CSV importados num bucket chamado importacoes-csv.
* **Processamento Assíncrono:** Supabase Edge Functions (Deno/TypeScript) são usadas para processar os ficheiros CSV em segundo plano, garantindo que a interface do utilizador não fica bloqueada durante operações longas.

**3. Lógica de Importação de Inscrições**

Este processo é responsável por criar ou atualizar os registos de leads a partir de uma planilha de inscrições.

**Fluxo do Processo**

1. O utilizador seleciona um lançamento e um ficheiro CSV na página de "Importação".
2. O frontend faz o upload do ficheiro para o Supabase Storage.
3. O frontend invoca a Edge Function processar-inscricao-csv, passando o caminho do ficheiro e o ID do lançamento.
4. A Edge Function lê o CSV, agrupa os dados por email e envia um lote de dados para a função SQL processar\_lote\_inscricoes.
5. A função SQL itera sobre cada lead:
   * Se o lead **não existe** para aquele lançamento, um novo registo é criado na tabela leads.
   * Se o lead **já existe**, o seu registo é atualizado com as informações da planilha (ex: UTMs).
6. A função retorna um resumo da operação (total recebido, criados, atualizados).

**Códigos Finais**

**Edge Function: processar-inscricao-csv**

// NOME DA FUNÇÃO: processar-inscricao-csv

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2';

import Papa from 'https://esm.sh/papaparse@5.4.1';

const corsHeaders = {

'Access-Control-Allow-Origin': '\*',

'Access-control-allow-headers': 'authorization, x-client-info, apikey, content-type'

};

// Mapeador de colunas flexível

const columnMapper = (key) => {

if (!key) return null;

const lowerKey = key.toLowerCase().trim();

if (lowerKey.includes('email')) return 'email';

if (lowerKey.includes('nome')) return 'nome';

if (lowerKey.includes('phone') || lowerKey.includes('telefone')) return 'telefone';

if (lowerKey.includes('utm\_source')) return 'utm\_source';

if (lowerKey.includes('utm\_medium')) return 'utm\_medium';

if (lowerKey.includes('utm\_campaign')) return 'utm\_campaign';

if (lowerKey.includes('utm\_content')) return 'utm\_content';

if (lowerKey.includes('utm\_term')) return 'utm\_term';

return null;

};

Deno.serve(async (req) => {

if (req.method === 'OPTIONS') {

return new Response('ok', { headers: corsHeaders });

}

try {

const supabaseAdmin = createClient(Deno.env.get('SUPABASE\_URL') ?? '', Deno.env.get('SUPABASE\_SERVICE\_ROLE\_KEY') ?? '');

const { launch\_id, file\_path } = await req.json();

if (!launch\_id || !file\_path) {

throw new Error("Parâmetros 'launch\_id' e 'file\_path' são obrigatórios.");

}

const { data: fileData, error: downloadError } = await supabaseAdmin.storage.from('importacoes-csv').download(file\_path);

if (downloadError) throw downloadError;

const csvText = await fileData.text();

const { data: parsedData } = Papa.parse(csvText, { header: true, skipEmptyLines: true });

const leadsParaEnviar = parsedData.map(row => {

const lead: { [key: string]: any } = {};

for (const key in row) {

const standardKey = columnMapper(key);

if (standardKey) {

lead[standardKey] = row[key]?.trim() || null;

}

}

if (!lead.email) return null;

lead.email = lead.email.toLowerCase();

return lead;

}).filter(Boolean);

const { data: result, error: rpcError } = await supabaseAdmin.rpc('processar\_lote\_inscricoes', {

p\_launch\_id: launch\_id,

p\_leads\_json: leadsParaEnviar

});

if (rpcError) throw rpcError;

return new Response(JSON.stringify(result), {

headers: { ...corsHeaders, 'Content-Type': 'application/json' },

status: 200

});

} catch (error) {

console.error("ERRO NA IMPORTAÇÃO DE INSCRIÇÕES:", error);

return new Response(JSON.stringify({ message: error.message }), {

headers: { ...corsHeaders, 'Content-Type': 'application/json' },

status: 500

});

}

});

**Função SQL: processar\_lote\_inscricoes**

CREATE OR REPLACE FUNCTION public.processar\_lote\_inscricoes(

p\_launch\_id UUID,

p\_leads\_json JSONB

)

RETURNS JSONB

LANGUAGE plpgsql

AS $$

DECLARE

rec JSONB;

v\_lead\_id UUID;

v\_created\_count INT := 0;

v\_updated\_count INT := 0;

v\_total\_records INT := 0;

BEGIN

v\_total\_records := jsonb\_array\_length(p\_leads\_json);

FOR rec IN SELECT \* FROM jsonb\_to\_recordset(p\_leads\_json) AS x(

email TEXT, nome TEXT, telefone TEXT, utm\_source TEXT, utm\_medium TEXT, utm\_campaign TEXT, utm\_content TEXT, utm\_term TEXT

)

LOOP

SELECT id INTO v\_lead\_id

FROM public.leads

WHERE email = rec.email AND launch\_id = p\_launch\_id

LIMIT 1;

IF v\_lead\_id IS NULL THEN

INSERT INTO public.leads (launch\_id, email, nome, telefone, utm\_source, utm\_medium, utm\_campaign, utm\_content, utm\_term)

VALUES (p\_launch\_id, rec.email, rec.nome, rec.telefone, rec.utm\_source, rec.utm\_medium, rec.utm\_campaign, rec.utm\_content, rec.utm\_term);

v\_created\_count := v\_created\_count + 1;

ELSE

UPDATE public.leads

SET

nome = COALESCE(rec.nome, nome),

telefone = COALESCE(rec.telefone, telefone),

utm\_source = COALESCE(rec.utm\_source, utm\_source),

utm\_medium = COALESCE(rec.utm\_medium, utm\_medium),

utm\_campaign = COALESCE(rec.utm\_campaign, utm\_campaign),

utm\_content = COALESCE(rec.utm\_content, utm\_content),

utm\_term = COALESCE(rec.utm\_term, utm\_term)

WHERE id = v\_lead\_id;

v\_updated\_count := v\_updated\_count + 1;

END IF;

END LOOP;

RETURN jsonb\_build\_object(

'status', 'SUCCESS',

'message', 'Inscrições processadas.',

'total\_recebido', v\_total\_records,

'criados', v\_created\_count,

'atualizados', v\_updated\_count

);

EXCEPTION

WHEN OTHERS THEN

RETURN jsonb\_build\_object('status', 'ERROR', 'message', SQLERRM);

END;

$$;

**4. Lógica de Importação de Check-ins**

Este processo é responsável por gravar as respostas da pesquisa de perfil e calcular os scores (score e mql\_score).

**Fluxo do Processo**

1. O utilizador seleciona um lançamento e um ficheiro CSV de check-in. O cabeçalho deste ficheiro deve conter o **texto exato das perguntas**.
2. O frontend faz o upload do ficheiro para o Supabase Storage.
3. O frontend invoca a Edge Function processar-checkin-csv.
4. A Edge Function primeiro busca na base de dados um "mapa de tradução" para converter o texto das perguntas (do cabeçalho do CSV) nos seus respetivos IDs (UUIDs).
5. Ela lê o CSV, usa o mapa para encontrar o ID de cada pergunta, e agrupa todas as respostas por email.
6. A Edge Function envia um lote de dados para a função SQL processar\_lote\_checkins.
7. A função SQL itera sobre cada lead:
   * Encontra o lead na base de dados.
   * Itera sobre as suas respostas, buscando na tabela perguntas a classe ('Score' ou 'Perfil') e o peso de cada opção.
   * Soma os totais de score e mql\_score.
   * Grava os totais na tabela leads e os textos das respostas nas colunas respostas e respostas\_perfil da tabela respostas\_leads.
8. A função retorna um resumo da operação.

**Códigos Finais**

**Edge Function: processar-checkin-csv**

// NOME DA FUNÇÃO: processar-checkin-csv

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2';

import Papa from 'https://esm.sh/papaparse@5.4.1';

const corsHeaders = {

'Access-Control-Allow-Origin': '\*',

'Access-control-allow-headers': 'authorization, x-client-info, apikey, content-type'

};

function parseDate(dateString) {

if (!dateString) return new Date();

const match = dateString.match(/(\d{2})\/(\d{2})\/(\d{4})\s(\d{2}):(\d{2})(?::\d{2})?/);

if (match) {

const day = match[1], month = match[2], year = match[3], hour = match[4], minute = match[5];

return new Date(Date.UTC(year, month - 1, day, hour, minute));

}

const directParse = new Date(dateString);

if (!isNaN(directParse.getTime())) return directParse;

return new Date();

}

Deno.serve(async (req) => {

if (req.method === 'OPTIONS') {

return new Response('ok', { headers: corsHeaders });

}

try {

const supabaseAdmin = createClient(Deno.env.get('SUPABASE\_URL') ?? '', Deno.env.get('SUPABASE\_SERVICE\_ROLE\_KEY') ?? '');

const { launch\_id, file\_path } = await req.json();

if (!launch\_id || !file\_path) {

throw new Error("Parâmetros 'launch\_id' e 'file\_path' são obrigatórios.");

}

const { data: perguntas, error: perguntasError } = await supabaseAdmin.from('perguntas').select('id, texto');

if (perguntasError) throw perguntasError;

const perguntaMap = perguntas.reduce((map, p) => {

if (p.texto) {

map[p.texto.toLowerCase().trim()] = p.id;

}

return map;

}, {});

const { data: fileData, error: downloadError } = await supabaseAdmin.storage.from('importacoes-csv').download(file\_path);

if (downloadError) throw downloadError;

const csvText = await fileData.text();

const { data: parsedData } = Papa.parse(csvText, { header: true, skipEmptyLines: true });

const checkinsAgrupados = parsedData.reduce((acc, row) => {

let email = null;

let dataOriginalValue = null;

const respostas = [];

for (const key in row) {

const value = row[key];

const lowerKey = key.toLowerCase().trim();

if (!value) continue;

if (lowerKey.includes('email') || lowerKey.includes('e-mail')) {

email = value.trim().toLowerCase();

} else if (lowerKey.includes('data') || lowerKey.includes('carimbo') || lowerKey.includes('timestamp')) {

dataOriginalValue = value;

} else {

const perguntaId = perguntaMap[lowerKey];

if (perguntaId) {

respostas.push({ pergunta: perguntaId, resposta: value.trim() || '' });

}

}

}

if (!email) return acc;

if (!acc[email]) {

acc[email] = {

email: email,

data\_original: parseDate(dataOriginalValue).toISOString(),

respostas: []

};

}

acc[email].respostas.push(...respostas);

return acc;

}, {});

const checkinsParaEnviar = Object.values(checkinsAgrupados);

const { data: result, error: rpcError } = await supabaseAdmin.rpc('processar\_lote\_checkins', {

p\_launch\_id: launch\_id,

p\_checkins\_json: checkinsParaEnviar

});

if (rpcError) throw rpcError;

return new Response(JSON.stringify(result), {

headers: { ...corsHeaders, 'Content-Type': 'application/json' },

status: 200

});

} catch (error) {

console.error("ERRO NA IMPORTAÇÃO DE CHECK-INS:", error);

return new Response(JSON.stringify({ message: error.message }), {

headers: { ...corsHeaders, 'Content-Type': 'application/json' },

status: 500

});

}

});

**Função SQL: processar\_lote\_checkins**

CREATE OR REPLACE FUNCTION public.processar\_lote\_checkins(

p\_launch\_id UUID,

p\_checkins\_json JSONB

)

RETURNS JSONB

LANGUAGE plpgsql

AS $$

DECLARE

rec RECORD;

v\_lead\_id UUID;

v\_pergunta\_id UUID;

v\_resposta\_texto TEXT;

v\_pergunta\_classe TEXT;

v\_opcoes\_json JSONB;

v\_valor INT;

v\_score\_total INT;

v\_mql\_score\_total INT;

v\_respostas\_score\_obj JSONB;

v\_respostas\_perfil\_obj JSONB;

v\_updated\_count INT := 0;

v\_not\_found\_count INT := 0;

v\_total\_records INT := 0;

BEGIN

v\_total\_records := jsonb\_array\_length(p\_checkins\_json);

FOR rec IN SELECT \* FROM jsonb\_array\_elements(p\_checkins\_json)

LOOP

SELECT id INTO v\_lead\_id

FROM public.leads

WHERE email = rec.value->>'email' AND launch\_id = p\_launch\_id

LIMIT 1;

IF v\_lead\_id IS NULL THEN

v\_not\_found\_count := v\_not\_found\_count + 1;

CONTINUE;

END IF;

v\_score\_total := 0;

v\_mql\_score\_total := 0;

v\_respostas\_score\_obj := '{}'::jsonb;

v\_respostas\_perfil\_obj := '{}'::jsonb;

FOR v\_pergunta\_id, v\_resposta\_texto IN

SELECT (p.value->>'pergunta')::UUID, (p.value->>'resposta')::TEXT

FROM jsonb\_array\_elements(rec.value->'respostas') p

LOOP

SELECT opcoes, classe INTO v\_opcoes\_json, v\_pergunta\_classe

FROM public.perguntas

WHERE id = v\_pergunta\_id

LIMIT 1;

v\_valor := NULL;

IF FOUND AND v\_opcoes\_json IS NOT NULL THEN

SELECT (opt->>'valor')::INT INTO v\_valor FROM jsonb\_array\_elements(v\_opcoes\_json) AS opt WHERE TRIM(LOWER(opt->>'opcao')) = TRIM(LOWER(v\_resposta\_texto));

IF v\_valor IS NULL THEN

SELECT (opt->>'peso')::INT INTO v\_valor FROM jsonb\_array\_elements(v\_opcoes\_json) AS opt WHERE TRIM(LOWER(opt->>'texto')) = TRIM(LOWER(v\_resposta\_texto));

END IF;

IF v\_valor IS NOT NULL THEN

IF LOWER(v\_pergunta\_classe) = 'score' THEN

v\_score\_total := v\_score\_total + v\_valor;

v\_respostas\_score\_obj := v\_respostas\_score\_obj || jsonb\_build\_object(v\_pergunta\_id::text, v\_resposta\_texto);

ELSIF LOWER(v\_pergunta\_classe) = 'perfil' THEN

v\_mql\_score\_total := v\_mql\_score\_total + v\_valor;

v\_respostas\_perfil\_obj := v\_respostas\_perfil\_obj || jsonb\_build\_object(v\_pergunta\_id::text, v\_resposta\_texto);

END IF;

END IF;

END IF;

END LOOP;

UPDATE public.leads

SET

score = v\_score\_total,

mql\_score = v\_mql\_score\_total,

check\_in\_at = (rec.value->>'data\_original')::TIMESTAMPTZ

WHERE id = v\_lead\_id;

INSERT INTO public.respostas\_leads (lead\_id, respostas, respostas\_perfil)

VALUES (v\_lead\_id, v\_respostas\_score\_obj, v\_respostas\_perfil\_obj)

ON CONFLICT (lead\_id) DO UPDATE

SET

respostas = EXCLUDED.respostas,

respostas\_perfil = EXCLUDED.respostas\_perfil;

v\_updated\_count := v\_updated\_count + 1;

END LOOP;

RETURN jsonb\_build\_object(

'status', 'SUCCESS',

'message', 'Check-ins processados com sucesso.',

'total\_recebido', v\_total\_records,

'atualizados', v\_updated\_count,

'nao\_encontrados', v\_not\_found\_count

);

EXCEPTION

WHEN OTHERS THEN

RETURN jsonb\_build\_object('status', 'ERROR', 'message', SQLERRM);

END;

$$;