# DESCRIÇÃO

A tabela "etapas" armazena as etapas de produção reutilizáveis que compõem as rotas de fabricação do sistema ORBIS.

Cada etapa representa uma fase específica do processo produtivo (ex.: CORTE LASER, DESTAQUE, DOBRADEIRA, MONTAGEM), contendo seus centros de produção, centros de trabalho e parâmetros necessários associados.

# OBJETIVO

* Padronizar etapas de produção em toda a empresa
* Centralizar configuração de centros de trabalho por etapa
* Definir parâmetros necessários para cada fase do processo
* Permitir reutilização de etapas em múltiplas etapas
* Facilitar manutenção e evolução dos processos produtivos
* Integrar com sistema de cálculo de TPE do widget ORBIS

# LÓGICA TÉCNICA DE AQUISIÇÃO DE DADOS

**Fontes**

* Interface administrativa do widget (criação/edição de parâmetros);
* Chamadas de API diretamente (criação/edição de parâmetros).

**Script para criar tabela**

CREATE TABLE etapas (

id INT IDENTITY(1,1) PRIMARY KEY,

nome NVARCHAR(100) NOT NULL UNIQUE,

descricao NVARCHAR(MAX),

centros NVARCHAR(200) NOT NULL,

centros\_trabalho NVARCHAR(MAX) NOT NULL,

parametros\_necessarios NVARCHAR(MAX) NOT NULL,

ativa BIT DEFAULT 1,

data\_criacao DATETIME2 DEFAULT GETDATE(),

data\_modificacao DATETIME2 DEFAULT GETDATE()

);

-- Índices para performance

CREATE INDEX idx\_etapas\_nome ON etapas(nome);

CREATE INDEX idx\_etapas\_ativa ON etapas(ativa);

CREATE INDEX idx\_etapas\_centros ON etapas(centros);

-- Constraint para validar JSON no campo centros\_trabalho

ALTER TABLE etapas

ADD CONSTRAINT chk\_etapas\_centros\_trabalho\_json

CHECK (centros\_trabalho IS NULL OR ISJSON(centros\_trabalho) = 1);

-- Constraint para validar JSON no campo parametros\_necessarios

ALTER TABLE etapas

ADD CONSTRAINT chk\_etapas\_parametros\_json

CHECK (parametros\_necessarios IS NULL OR ISJSON(parametros\_necessarios) = 1);

# FILTROS

Recomendações de filtros no consumo (API/relatórios/SQL):

* **Por nome**
* **Por descrição**
* **Por ativa (ativo/inativo)**
* **Por centro de produção**

# RECORRÊNCIA DE ATUALIZAÇÃO

* **Widget/API** → atualização **em tempo real** (transacional).

# CAMPO CHAVE PARA ATUALIZAÇÃO

**Chave primária**: `id` (INT IDENTITY)

**Chave natural (UNIQUE):** `nome` - garante que não existam etapas com nomes duplicados

**Estratégia de operações:**

**CREATE (POST)**

- Valida se `nome` já existe (UNIQUE constraint)

- Insere novo registro com ID auto-gerado

**UPDATE (PUT)**

- Identificação por `id` enviado na URL (`/orbis/*etapas*/:id`)

- Atualiza campos enviados no body

- Valida unicidade de `nome` se este campo for alterado

**DELETE (DELETE)**

- Quando usuário excluir via widget: DELETE FROM etapas WHERE id = @id

- Remove permanentemente o registro do banco

- Usado quando a etapa foi criada por engano ou não é mais necessária

**DESATIVAR (PATCH)**

- Quando usuário desativar via widget: UPDATE etapas SET ativa = 0 WHERE id = @id

- Mantém o registro no banco com ativa = 0

- Usado quando a etapa não deve mais aparecer, mas precisa ser mantida para histórico/auditoria

# API – DADOS DE ENTRADA

**POST /orbis/etapas – Criar nova etapa**

* **Descrição**: Cria uma nova etapa de produção no sistema.
* **Body (JSON):**

{

"nome": "CORTE LASER",

"descricao": "Etapa de corte laser",

"centros": "MP10",

"centros\_trabalho": ["MA303", "MA301", "DE103"],

"parametros\_necessarios": [

"espessura\_chapa",

"material",

"perimetro",

"peso\_peca",

"qtd\_recortes\_laserchapa"

],

"ativa": true

}

* **Retorno 201:**

{

"success": true,

"data": {

"id": 1,

"nome": "CORTE LASER",

"descricao": "Etapa de corte laser",

"centros": "MP10",

"centros\_trabalho": ["MA303", "MA301", "DE103"],

"parametros\_necessarios": [

"espessura\_chapa",

"material",

"perimetro",

"peso\_peca",

"qtd\_recortes\_laserchapa"

],

"ativa": true,

"data\_criacao": "2025-10-10T14:30:00Z",

"data\_modificacao": "2025-10-10T14:30:00Z"

}

}

* **Retorno** **409**: {"success": false, "error": "Já existe uma etapa com este nome"}

**PUT /orbis/etapas/:id - Atualizar etapa existente**

* **Descrição:** Atualiza dados de uma etapa existente identificada pelo ID
* **URL Parameter:** "id" (obrigatório)
* **Body (JSON):**

{

"nome": "CORTE LASER (Atualizada)",

"descricao": "Etapa de corte laser com novos centros",

"centros\_trabalho": ["MA303", "MA301", "DE103", "MA305"]

}

* **Retorno 200:**

{

"success": true,

"data": {

"id": 1,

"nome": "CORTE LASER (Atualizada)",

"descricao": "Etapa de corte laser com novos centros",

"centros": "MP10",

"centros\_trabalho": ["MA303", "MA301", "DE103", "MA305"],

"parametros\_necessarios": [

"espessura\_chapa",

"material",

"perimetro",

"peso\_peca",

"qtd\_recortes\_laserchapa"

],

"ativa": true,

"data\_criacao": "2025-10-10T14:30:00Z",

"data\_modificacao": "2025-10-10T15:45:00Z"

}

}

* **Retorno** **404**: {"success": false, "error": "Etapa não encontrada"}
* **Retorno** **409**: {"success": false, "error": "Já existe uma etapa com este nome"}

**PATCH /orbis/etapas/:id/desativar - Desativar etapa**

* **Descrição:** Soft delete - marca "ativo = 0" (mantém registro no banco)
* **URL Parameter:** "id" (obrigatório)
* **Retorno 200:**

{

"success": true,

"message": "Etapa desativada com sucesso",

"data": {

"id": 1,

"ativa": false

}

}

**PATCH /orbis/etapas/:id/ativar - Reativar etapa**

* **Descrição:** Marca "ativo = 1" (reativa parâmetro desativado)
* **URL Parameter:** "id" (obrigatório)
* **Retorno 200:**

{

"success": true,

"message": "Etapa ativada com sucesso",

"data": {

"id": 1,

"ativo": true

}

}

**DELETE /orbis/etapas/:id - Excluir etapa permanentemente**

* **Descrição:** DELETE físico - remove registro do banco definitivamente
* **URL Parameter:** "id" (obrigatório)
* **Retorno 200:**

{

"success": true,

"message": "Etapa excluída permanentemente"

}

* **Retorno 404:** “{“success": false, "error": "Etapa não encontrada”}"

# API – DADOS DE SAÍDA

**GET “/orbis/etapas” – Listar etapas**

* **Descrição:** Retorna lista de todas as etapas (com filtros opcionais de [codigo, nome, ativo, centro\_prod, descrição])
* **Parâmetros:** filtros opcionais
* **Retorno:** coleção de etapas.

**GET “/orbis/etapas/:id” – Buscar etapa por ID**

* **Descrição:** Retorna uma etapa específica pelo ID
* **Retorno 200**: Objeto único da etapa
* **Retorno 404**: “{"success": false, "error": "Etapa não encontrada"}”

**GET “/orbis/etapas?ids=1,2,3” – Buscar múltiplas etapas por IDs**

* **Descrição:** Retorna múltiplas etapas de uma vez
* **Retorno 200**: Coleção de etapas.
* **Retorno 404**: “{"success": false, "error": "Etapas não encontrada"}”

# CATEGORIZAÇÃO DO DHIVE

# ÁREAS IMPACTADAS PELA INFORMAÇÃO

Engenharia de Processos

# VALOR AGREGADO PARA A COMPANHIA

* **Padronização**: Unifica definição de etapas produtivas em toda a empresa
* **Reutilização**: Mesma etapa pode ser usada em múltiplas rotas
* **Manutenção**: Alterar uma etapa afeta automaticamente todas as rotas
* **Flexibilidade**: Permite criar e modificar etapas sem alteração de código
* **Rastreabilidade**: Mantém histórico completo de alterações
* **Integração**: Base para cálculos automatizados de TPE e planejamento
* **Qualidade**: Garante que processos sigam etapas padronizadas e aprovadas
* **Migração** **para** **3DEXPERIENCE**: Remove dependência de configurações hardcoded

# INTEGRAÇÃO COM SISTEMA DE AUDITORIA

Todas as operações de CREATE, UPDATE, DELETE, ACTIVATE e DEACTIVATE nesta tabela DEVEM registrar logs na tabela "logs\_auditoria".

1. RECEBER user\_id no Body

Todas as requisições POST, PUT, PATCH e DELETE devem receber o campo user\_id:

Body exemplo:

{

"nome": "...",

"descricao": "...",

"user\_id": "eduardo.silva" ← OBRIGATÓRIO

}

2. CHAMAR Função de Log

Após cada operação bem-sucedida, chamar:

registrarLog(

tabela: 'nome\_da\_tabela',

registro\_id: id\_do\_registro,

operacao: 'CREATE' | 'UPDATE' | 'DELETE' | 'ACTIVATE' | 'DEACTIVATE',

valores\_anteriores: objeto\_com\_valores\_antigos (NULL para CREATE),

valores\_novos: objeto\_com\_valores\_novos (NULL para DELETE),

user\_id: user\_id\_recebido\_no\_body

);

3. EXEMPLOS POR OPERAÇÃO

**CREATE** (POST):

* Capturar user\_id do body
* Executar INSERT na tabela
* Chamar: registrarLog('tabela', novo\_id, 'CREATE', null, dados\_inseridos, user\_id)

**UPDATE** (PUT):

* Capturar user\_id do body
* Buscar valores anteriores: SELECT \* FROM tabela WHERE id = @id
* Executar UPDATE na tabela
* Buscar valores novos: SELECT \* FROM tabela WHERE id = @id
* Chamar: registrarLog('tabela', id, 'UPDATE', valores\_anteriores, valores\_novos, user\_id)

**DELETE** (DELETE):

* Capturar user\_id do body
* Buscar valores anteriores: SELECT \* FROM tabela WHERE id = @id
* Executar DELETE na tabela
* Chamar: registrarLog('tabela', id, 'DELETE', valores\_anteriores, null, user\_id)

**ACTIVATE** (PATCH /:id/ativar):

* Capturar user\_id do body
* Buscar valor anterior: SELECT ativo FROM tabela WHERE id = @id
* Executar: UPDATE tabela SET ativo = 1 WHERE id = @id
* Chamar: registrarLog('tabela', id, 'ACTIVATE', {ativo: 0}, {ativo: 1}, user\_id)

**DEACTIVATE (PATCH /:id/desativar):**

* Capturar user\_id do body
* Buscar valor anterior: SELECT ativo FROM tabela WHERE id = @id
* Executar: UPDATE tabela SET ativo = 0 WHERE id = @id
* Chamar: registrarLog('tabela', id, 'DEACTIVATE', {ativo: 1}, {ativo: 0}, user\_id)