

Instruções de Configuração do Supabase - DELIVEREI

Este guia contém o passo a passo completo para configurar o banco de dados do projeto DELIVEREI no Supabase.

Pré-requisitos


- Conta no Supabase (<https://supabase.com>)
- Projeto criado no Supabase
- Credenciais do projeto (já configuradas no arquivo `.env`)

Configuração Realizada

Arquivo `.env` Criado

O arquivo `.env` foi criado com as seguintes configurações:

```
DATABASE_URL="postgresql://postgres.hmlxtjcgkbzcwsjvdl:T39PcxxRThL3Bd0v@aws-1-us-east-1.pooler.supabase.com:6543/postgres?pgbouncer=true"
SUPABASE_URL="https://hmlxtjcgkbzcwsjvdl.supabase.co"
SUPABASE_ANON_KEY="eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
JWT_SECRET=<gerado automaticamente>
JWT_REFRESH_SECRET=<gerado automaticamente>
```

 **IMPORTANTE:** O arquivo `.env` contém informações sensíveis e **NÃO deve ser commitado** no Git. Ele já está incluído no `.gitignore`.

Dependências Instaladas

Todas as dependências do projeto foram instaladas com sucesso:

- `@prisma/client@5.22.0`
- `prisma@5.22.0`
- E todas as outras dependências do NestJS

Passo a Passo: Criar Tabelas no Supabase

1. Acessar o SQL Editor

1. Acesse o dashboard do Supabase: <https://supabase.com/dashboard>
2. Selecione seu projeto: **hmlxtjcgkbzcwsjvdl**
3. No menu lateral, clique em **SQL Editor** (ícone de banco de dados)
4. Clique em **New query** para criar uma nova consulta

2. Executar o Script SQL

1. Abra o arquivo `supabase-init.sql` (localizado nesta pasta)
2. **Copie TODO o conteúdo** do arquivo
3. **Cole no SQL Editor** do Supabase

4. Clique no botão **Run** (ou pressione `Ctrl+Enter` / `Cmd+Enter`)

3. Verificar a Execução

Após executar o script, você deve ver:

- ☒ Mensagem de sucesso: "Success. No rows returned"
- ☒ Tempo de execução (geralmente < 1 segundo)

4. Verificar as Tabelas Criadas

1. No menu lateral, clique em **Table Editor**
2. Você deve ver as seguintes tabelas:
 - ☒ `empresas` - Empresas/restaurantes cadastrados
 - ☒ `usuarios` - Usuários do sistema (admins e clientes)
 - ☒ `produtos` - Produtos/itens do cardápio
 - ☒ `pedidos` - Pedidos realizados
 - ☒ `refresh_tokens` - Tokens de autenticação
3. Clique em cada tabela para verificar:
 - Colunas criadas corretamente
 - Tipos de dados
 - Índices
 - Relacionamentos (Foreign Keys)



Estrutura do Banco de Dados

Tabelas Criadas

1. empresas

Armazena informações das empresas/restaurantes

- `id` (UUID) - Identificador único
- `nome` - Nome da empresa
- `slug` - URL amigável (único)
- `subdominio` - Subdomínio para multi-tenant (único)
- `ativo` - Status da empresa
- `createdAt` , `updatedAt` - Timestamps

2. usuarios

Armazena todos os usuários do sistema

- `id` (UUID) - Identificador único
- `email` - Email (único)
- `senha` - Senha hash
- `nome` - Nome completo
- `role` - Papel (SUPER_ADMIN, ADMIN_EMPRESA, CLIENTE)
- `empresaId` - Referência à empresa (opcional)
- `ativo` - Status do usuário
- `createdAt` , `updatedAt` - Timestamps

3. produtos

Armazena produtos/itens do cardápio

- `id` (UUID) - Identificador único
- `nome` - Nome do produto

- `descricao` - Descrição detalhada
- `preco` - Preço (DECIMAL 10,2)
- `imagem` - URL da imagem
- `ativo` - Status do produto
- `empresaId` - Referência à empresa
- `estoque` - Quantidade em estoque
- `categoria` - Categoria do produto
- `createdAt`, `updatedAt` - Timestamps

4. pedidos

Armazena pedidos realizados

- `id` (UUID) - Identificador único
- `numero` - Número do pedido (único)
- `status` - Status do pedido (enum)
- `total` - Valor total (DECIMAL 10,2)
- `clienteId` - Referência ao cliente
- `empresaId` - Referência à empresa
- `observacoes` - Observações do pedido
- `createdAt`, `updatedAt` - Timestamps

5. refresh_tokens

Armazena tokens de refresh JWT

- `id` (UUID) - Identificador único
- `token` - Token de refresh (único)
- `usuarioId` - Referência ao usuário
- `expiresAt` - Data de expiração
- `createdAt` - Timestamp

Enums Criados

- **Role:** `SUPER_ADMIN`, `ADMIN_EMPRESA`, `CLIENTE`
- **StatusPedido:** `PENDENTE`, `CONFIRMADO`, `EM_PREPARO`, `SAIU_ENTREGA`, `ENTREGUE`, `CANCELADO`



Testar a Conexão

Após criar as tabelas, teste a conexão do backend:

```
# No diretório backend/  
npm run start:dev
```

O servidor deve iniciar sem erros de conexão com o banco de dados.



Verificação Final - Checklist

- [] Script SQL executado com sucesso no Supabase SQL Editor
- [] 5 tabelas criadas: `empresas`, `usuarios`, `produtos`, `pedidos`, `refresh_tokens`
- [] 2 enums criados: `Role`, `StatusPedido`
- [] Índices criados corretamente
- [] Foreign Keys (relacionamentos) configurados
- [] Arquivo `.env` configurado com as credenciais corretas

- [] Dependências instaladas (`node_modules/` presente)
- [] Backend inicia sem erros de conexão

Solução de Problemas

Erro: “type already exists”

Se você já executou o script antes, os tipos (enums) já existem. Você pode:

1. Deletar as tabelas existentes no Table Editor
2. Executar o script novamente

Ou executar apenas as partes necessárias do script.

Erro: “relation already exists”

As tabelas já foram criadas. Verifique no Table Editor se todas as 5 tabelas estão presentes.

Erro de Conexão no Backend

Verifique se:

1. A `DATABASE_URL` no `.env` está correta
2. O projeto Supabase está ativo
3. As credenciais não expiraram

Próximos Passos

Após a configuração bem-sucedida:

1. **Criar dados de teste** (opcional):
 - Criar uma empresa de exemplo
 - Criar usuários de teste
 - Adicionar produtos ao cardápio
2. **Configurar autenticação:**
 - Testar registro de usuários
 - Testar login
 - Verificar geração de tokens JWT
3. **Desenvolver funcionalidades:**
 - CRUD de empresas
 - CRUD de produtos
 - Sistema de pedidos
 - Dashboard administrativo

Suporte

- Documentação Supabase: <https://supabase.com/docs>
 - Documentação Prisma: <https://www.prisma.io/docs>
 - Documentação NestJS: <https://docs.nestjs.com>
-

Data de criação: 08/10/2025

Versão do Prisma: 5.22.0

Projeto: DELIVEREI v1