**Guia Didático — Entendendo o arquivo seed.ts (Prisma)**

Este guia explica, de forma didática, o que o arquivo seed.ts faz, como ele se relaciona ao schema do Prisma e por que cada passo é importante.  
O objetivo é permitir que você adapte o seed para outros projetos com segurança.

**1) O que é um seed e por que usar?**

O **seed** popula o banco com *dados iniciais* úteis para desenvolvimento, testes e demonstrações.  
Ele garante que as tabelas tenham registros mínimos (ex.: tipos, categorias, perfis) para que a aplicação rode sem erros.

**2) Pré-requisitos para o seed funcionar**

* **Schema migrado** para o banco (tabelas existem):
* npx prisma migrate dev
* **Client do Prisma gerado**:
* npx prisma generate

(o migrate já gera automaticamente)

* **Script de seed configurado** no package.json:
* {
* "prisma": { "seed": "tsx prisma/seed.ts" },
* "scripts": { "seed": "npx prisma db seed" }
* }

**3) Estrutura geral do seed.ts**

import { PrismaClient } from "@prisma/client";

const prisma = new PrismaClient();

async function main() {

// 3.1 Limpeza (opcional em dev)

// 3.2 Inserts em entidades-base

// 3.3 Inserts de relações N:N

console.log("Seed concluído ✅");

}

main()

.catch((e) => { console.error(e); process.exit(1); })

.finally(async () => prisma.$disconnect());

O padrão envolve:

* Criar o client.
* Executar uma função principal (main) com await.
* Tratar erros.
* Garantir prisma.$disconnect() no final.

**4) Limpeza: por que esta ordem importa?**

O seed começa removendo registros das tabelas **na ordem correta**, respeitando chaves estrangeiras:

await prisma.subtipoBarreira.deleteMany();

await prisma.barreiraAcessibilidade.deleteMany();

await prisma.acessibilidade.deleteMany();

await prisma.barreira.deleteMany();

await prisma.subtipoDeficiencia.deleteMany();

await prisma.tipoDeficiencia.deleteMany();

**Ordem:**

1. Tabelas de junção (N:N).
2. Tabelas dependentes.
3. Tabelas-mãe.

✔ Dica: use a limpeza total **apenas em desenvolvimento**.

**5) Criação das entidades base**

Após limpar, o seed cria registros fundamentais nas tabelas-mãe.  
Os registros criados são guardados em variáveis para capturar os respectivos **IDs**, usados depois:

const motora = await prisma.tipoDeficiencia.create({ data: { nome: "Deficiência Motora" } });

const auditiva = await prisma.tipoDeficiencia.create({ data: { nome: "Deficiência Auditiva" } });

const visual = await prisma.tipoDeficiencia.create({ data: { nome: "Deficiência Visual" } });

const sub\_motora1 = await prisma.subtipoDeficiencia.create({ data: { nome: "Amputação MIE com muleta", tipoId: motora.id } });

const sub\_auditiva1 = await prisma.subtipoDeficiencia.create({ data: { nome: "Usuário de Libras", tipoId: auditiva.id } });

const sub\_visual1 = await prisma.subtipoDeficiencia.create({ data: { nome: "Baixa visão", tipoId: visual.id } });

**6) Uso de transações para múltiplos inserts**

Para criar várias barreiras e acessibilidades de forma atômica (todas ou nenhuma), foi usada a transação:

const [escadas, degrausAltos, pisoIrregular, faltaInterprete, comunicacaoOral, faltaContraste, faltaSinalizacaoTatil] =

await prisma.$transaction([

prisma.barreira.create({ data: { descricao: "Escadas" } }),

prisma.barreira.create({ data: { descricao: "Degraus altos" } }),

prisma.barreira.create({ data: { descricao: "Piso irregular" } }),

prisma.barreira.create({ data: { descricao: "Ausência de intérprete de Libras" } }),

prisma.barreira.create({ data: { descricao: "Dificuldade de comunicação oral" } }),

prisma.barreira.create({ data: { descricao: "Falta de contraste visual" } }),

prisma.barreira.create({ data: { descricao: "Falta de sinalização tátil" } }),

]);

// Acessibilidades

  const [rampa, pisoAntid, elevador, interprete, chatInterno, altoContraste, pisoGuia] =

    await prisma.$transaction([

      prisma.acessibilidade.create({ data: { descricao: "Rampa com inclinação adequada" } }),

      prisma.acessibilidade.create({ data: { descricao: "Piso antiderrapante" } }),

      prisma.acessibilidade.create({ data: { descricao: "Elevador / acesso em nível" } }),

      prisma.acessibilidade.create({ data: { descricao: "Intérprete de Libras" } }),

      prisma.acessibilidade.create({ data: { descricao: "Comunicação por chat interno" } }),

      prisma.acessibilidade.create({ data: { descricao: "Sinalização de alto contraste" } }),

      prisma.acessibilidade.create({ data: { descricao: "Piso guia / sinalização tátil" } }),

    ]);

Se algo falhar, a operação é revertida.

**7) Relacionamentos N:N com createMany e skipDuplicates**

As tabelas de junção são preenchidas em lote:

await prisma.subtipoBarreira.createMany({

data: [

{ subtipoId: sub\_motora1.id, barreiraId: escadas.id },

      { subtipoId: sub\_motora1.id, barreiraId: degrausAltos.id },

      { subtipoId: sub\_motora1.id, barreiraId: pisoIrregular.id },

      { subtipoId: sub\_auditiva1.id, barreiraId: comunicacaoOral.id },

      { subtipoId: sub\_auditiva1.id, barreiraId: faltaInterprete.id },

      { subtipoId: sub\_visual1.id, barreiraId: pisoIrregular.id },

      { subtipoId: sub\_visual1.id, barreiraId: faltaContraste.id },

      { subtipoId: sub\_visual1.id, barreiraId: faltaSinalizacaoTatil.id },

// ...

],

skipDuplicates: true,

});

await prisma.barreiraAcessibilidade.createMany({

data: [

  // Motora

      { barreiraId: escadas.id, acessibilidadeId: rampa.id },

      { barreiraId: escadas.id, acessibilidadeId: elevador.id },

      { barreiraId: degrausAltos.id, acessibilidadeId: rampa.id },

      { barreiraId: degrausAltos.id, acessibilidadeId: elevador.id },

      { barreiraId: pisoIrregular.id, acessibilidadeId: pisoAntid.id },

      // Auditiva

      { barreiraId: faltaInterprete.id, acessibilidadeId: interprete.id },

      { barreiraId: comunicacaoOral.id, acessibilidadeId: chatInterno.id },

      // Visual

      { barreiraId: faltaContraste.id, acessibilidadeId: altoContraste.id },

      { barreiraId: faltaSinalizacaoTatil.id, acessibilidadeId: pisoGuia.id },

],

skipDuplicates: true,

});

* skipDuplicates: evita erro caso a execução seja repetida.

**8) Boas práticas para seeds**

* Limpeza total **somente** em ambiente de desenvolvimento.
* Agrupar inserções relacionadas em transações para manter atomicidade.
* Guardar IDs resultantes para criar relações.
* Usar createMany + skipDuplicates para seeds idempotentes.
* Tratar erros e sempre finalizar com prisma.$disconnect().

**9) Como rodar e verificar**

Rodar o seed:

npm run seed

# ou

npx prisma db seed

Verificar no Prisma Studio:

npx prisma studio