1)

No package.json (scripts):

"prisma": {

    "seed": "tsx prisma/seed.ts"

  },

  "scripts": {

    "seed": "npx prisma db seed"

  }

2)

Schema.prisma

generator client {

  provider = "prisma-client-js"

}

datasource db {

  provider = "postgresql"

  url      = env("DATABASE\_URL")

}

model TipoDeficiencia {

  id        Int                   @id @default(autoincrement())

  nome      String                @unique

  subtipos  SubtipoDeficiencia[]

  createdAt DateTime              @default(now())

  updatedAt DateTime              @updatedAt

}

model SubtipoDeficiencia {

  id         Int                  @id @default(autoincrement())

  nome       String

  tipoId     Int

  tipo       TipoDeficiencia      @relation(fields: [tipoId], references: [id], onDelete: Cascade)

  barreiras  SubtipoBarreira[]

  createdAt  DateTime             @default(now())

  updatedAt  DateTime             @updatedAt

  @@unique([tipoId, nome]) // evita subtipo duplicado dentro do mesmo tipo

}

model Barreira {

  id               Int                      @id @default(autoincrement())

  descricao        String                   @unique

  subtipos         SubtipoBarreira[]

  acessibilidades  BarreiraAcessibilidade[]

  createdAt        DateTime                 @default(now())

  updatedAt        DateTime                 @updatedAt

}

model Acessibilidade {

  id               Int                      @id @default(autoincrement())

  descricao        String                   @unique

  barreiras        BarreiraAcessibilidade[]

  createdAt        DateTime                 @default(now())

  updatedAt        DateTime                 @updatedAt

}

// tabelas de junção explícitas (para fácil evolução)

model SubtipoBarreira {

  subtipoId  Int

  barreiraId Int

  subtipo    SubtipoDeficiencia @relation(fields: [subtipoId], references: [id], onDelete: Cascade)

  barreira   Barreira           @relation(fields: [barreiraId], references: [id], onDelete: Cascade)

  @@id([subtipoId, barreiraId])

  @@index([barreiraId])

}

model BarreiraAcessibilidade {

  barreiraId       Int

  acessibilidadeId Int

  barreira         Barreira        @relation(fields: [barreiraId], references: [id], onDelete: Cascade)

  acessibilidade   Acessibilidade  @relation(fields: [acessibilidadeId], references: [id], onDelete: Cascade)

  @@id([barreiraId, acessibilidadeId])

  @@index([acessibilidadeId])

}

npx prisma migrate dev --name init

npx prisma generate

3) Seed de dados

Seed.ts

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

const prisma = new PrismaClient();

async function main() {

  // limpa dados (apenas para desenvolvimento)

  await prisma.subtipoBarreira.deleteMany();

  await prisma.barreiraAcessibilidade.deleteMany();

  await prisma.acessibilidade.deleteMany();

  await prisma.barreira.deleteMany();

  await prisma.subtipoDeficiencia.deleteMany();

  await prisma.tipoDeficiencia.deleteMany();

  // Tipos

  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" },

  });

  // Subtipos

  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 },

  });

  // Barreiras

  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" } }),

    ]);

  // Subtipo ↔ Barreiras (N:N)

  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,

  });

  // Barreira ↔ Acessibilidade (N:N)

  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,

  });

  console.log("Seed concluído ✅");

}

main()

  .catch((e) => {

    console.error(e);

    process.exit(1);

  })

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

npm run seed

4) API Express — src/server.ts

import express from "express";

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

const app = express();

const prisma = new PrismaClient();

app.use(express.json());

/\*\* 1) Listar tipos com seus subtipos \*/

app.get("/tipos", async (\_req, res) => {

const tipos = await prisma.tipoDeficiencia.findMany({

orderBy: { id: "asc" },

include: { subtipos: { orderBy: { id: "asc" } } },

});

res.json(tipos);

});

/\*\* 2) Listar barreiras \*/

app.get("/barreiras", async (\_req, res) => {

const barreiras = await prisma.barreira.findMany({ orderBy: { id: "asc" } });

res.json(barreiras);

});

/\*\* 3) Listar acessibilidades \*/

app.get("/acessibilidades", async (\_req, res) => {

const acess = await prisma.acessibilidade.findMany({ orderBy: { id: "asc" } });

res.json(acess);

});

/\*\* 4) Obter um subtipo com suas barreiras e as acessibilidades de cada barreira \*/

app.get("/subtipos/:id", async (req, res) => {

const id = Number(req.params.id);

const subtipo = await prisma.subtipoDeficiencia.findUnique({

where: { id },

include: {

tipo: true,

barreiras: {

include: {

barreira: {

include: {

acessibilidades: {

include: { acessibilidade: true },

orderBy: { acessibilidadeId: "asc" },

},

},

},

},

orderBy: { barreiraId: "asc" },

},

},

});

if (!subtipo) return res.status(404).json({ error: "Subtipo não encontrado" });

// opcional: “achatar” a resposta para facilitar a leitura no front

const barreiras = subtipo.barreiras.map((sb) => ({

id: sb.barreira.id,

descricao: sb.barreira.descricao,

acessibilidades: sb.barreira.acessibilidades.map((ba) => ({

id: ba.acessibilidade.id,

descricao: ba.acessibilidade.descricao,

})),

}));

res.json({

id: subtipo.id,

nome: subtipo.nome,

tipo: { id: subtipo.tipo.id, nome: subtipo.tipo.nome },

barreiras,

});

});

/\*\* 5) Vincular barreiras a um subtipo (N:N) \*/

app.post("/subtipos/:id/barreiras", async (req, res) => {

const subtipoId = Number(req.params.id);

const { barreiraIds } = req.body as { barreiraIds: number[] };

if (!Array.isArray(barreiraIds) || barreiraIds.length === 0) {

return res.status(400).json({ error: "barreiraIds deve ser um array com pelo menos 1 id" });

}

// valida existência

const subtipo = await prisma.subtipoDeficiencia.findUnique({ where: { id: subtipoId } });

if (!subtipo) return res.status(404).json({ error: "Subtipo não encontrado" });

await prisma.subtipoBarreira.createMany({

data: barreiraIds.map((barreiraId) => ({ subtipoId, barreiraId })),

skipDuplicates: true,

});

res.json({ ok: true });

});

/\*\* 6) Vincular acessibilidades a uma barreira (N:N) \*/

app.post("/barreiras/:id/acessibilidades", async (req, res) => {

const barreiraId = Number(req.params.id);

const { acessibilidadeIds } = req.body as { acessibilidadeIds: number[] };

if (!Array.isArray(acessibilidadeIds) || acessibilidadeIds.length === 0) {

return res.status(400).json({ error: "acessibilidadeIds deve ser um array com pelo menos 1 id" });

}

const barreira = await prisma.barreira.findUnique({ where: { id: barreiraId } });

if (!barreira) return res.status(404).json({ error: "Barreira não encontrada" });

await prisma.barreiraAcessibilidade.createMany({

data: acessibilidadeIds.map((acessibilidadeId) => ({ barreiraId, acessibilidadeId })),

skipDuplicates: true,

});

res.json({ ok: true });

});

/\*\* middleware básico de erro \*/

app.use((err: any, \_req: any, res: any, \_next: any) => {

console.error(err);

res.status(500).json({ error: "Erro interno" });

});

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => console.log(`API Etapa 1 rodando em [http://localhost:${PORT}`)](http://localhost:$%7bPORT%7d%60)));

npm run dev

**Endpoints para testar (exemplos)**

* GET /tipos → lista tipos e subtipos
* GET /barreiras
* GET /acessibilidades
* GET /subtipos/1 → retorna subtipo + barreiras + acessibilidades por barreira
* POST /subtipos/1/barreiras

{ "barreiraIds": [1,2,3] }

* POST /barreiras/1/acessibilidades

{ "acessibilidadeIds": [1,3] }