1. No package.json

  "scripts": {

    "dev": "tsx watch src/server.ts",

    "seed": "npx prisma db seed"

  }

1. Criar src/server.ts

import express from "express";

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

const app = express();

const prisma = new PrismaClient();

app.use(express.json());

// adiciona aqui, um por um para testar

/\*\* 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}`));

/\*\* 1) Listar tipos \*/

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

  const tipos = await prisma.tipoDeficiencia.findMany({

    orderBy: { id: "asc" }

  });

  res.json(tipos);

});

/\*\* 2) Listar tipos com seus subtipos \*/

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

  const tipos = await prisma.tipoDeficiencia.findMany({

    orderBy: { id: "asc" },

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

  });

  res.json(tipos);

});

/\*\* 3) Listar barreiras \*/

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

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

  res.json(barreiras);

});

/\*\* 4) Listar acessibilidades \*/

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

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

  res.json(acess);

});

/\*\* 5) 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,

  });

});

/\*\* 6. Cria um tipo \*/

app.post("/tipos", async (req, res) => {

  const { nome } = req.body as { nome?: string };

  if (!nome || nome.trim() === "") {

    return res.status(400).json({ error: "O campo 'nome' é obrigatório" });

  }

  try {

    const tipo = await prisma.tipoDeficiencia.create({

      data: { nome: nome.trim() },

    });

    res.status(201).json(tipo);

  } catch (err) {

    res.status(500).json({ error: "Erro ao criar tipo", details: err });

  }

});

/\*\* 7. Cria um subtipo \*/

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

  const { nome, tipoId } = req.body as { nome?: string; tipoId?: number };

  if (!nome || !tipoId) {

    return res.status(400).json({ error: "Campos 'nome' e 'tipoId' são obrigatórios" });

  }

  try {

    // Verifica se o tipo existe

    const tipo = await prisma.tipoDeficiencia.findUnique({ where: { id: Number(tipoId) } });

    if (!tipo) {

      return res.status(404).json({ error: "Tipo não encontrado" });

    }

    const subtipo = await prisma.subtipoDeficiencia.create({

      data: { nome: nome.trim(), tipoId: Number(tipoId) },

    });

    res.status(201).json(subtipo);

  } catch (err) {

    res.status(500).json({ error: "Erro ao criar subtipo", details: err });

  }

});

/\*\* 8. Cria uma barreira \*/

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

  const { descricao } = req.body as { descricao?: string };

  if (!descricao || descricao.trim() === "") {

    return res.status(400).json({ error: "O campo 'descricao' é obrigatório" });

  }

  try {

    const barreira = await prisma.barreira.create({

      data: { descricao: descricao.trim() },

    });

    res.status(201).json(barreira);

  } catch (err) {

    res.status(500).json({ error: "Erro ao criar barreira", details: err });

  }

});

/\*\* 9. Cria uma acessibilidade \*/

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

  const { descricao } = req.body as { descricao?: string };

  if (!descricao || descricao.trim() === "") {

    return res.status(400).json({ error: "O campo 'descricao' é obrigatório" });

  }

  try {

    const acessibilidade = await prisma.acessibilidade.create({

      data: { descricao: descricao.trim() },

    });

    res.status(201).json(acessibilidade);

  } catch (err) {

    res.status(500).json({ error: "Erro ao criar acessibilidade", details: err });

  }

});

/\*\* 10) 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 });

});

/\*\* 11) 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 });

});