Desenvolvimento de Aplicações Web - Entrega 4 - Marcos Henrique Almeida Lima

$$$ src.entidades.gerente-tecnologia

```
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne, PrimaryGeneratedColumn
}
from "typeorm";
import Usuário from "./usuário";
import Patrocínio from "./patrocínio";
import ParticipaçãoMineração from "./participação-mineração";
export enum Titulacao  { GerenteTecnologia = "Gerente Tecnologia", GerenteInovação = "Gerente
Inovação", EngenheiroSistemasSênior = "Engenheiro Sistemas Sênior",
LíderEquipeDesenvolvimento = "Líder Equipe Desenvolvimento" };

@Entity()
export default class GerenteTecnologia extends BaseEntity {
@PrimaryGeneratedColumn()
id: number;
@Column({ type: "enum", enum: Titulacao  })
titulacao: Titulacao ;
@Column()
ano_ingresso: number;

@Column({ type: "date" })
data_nascimento: Date;

@Column()
telefone: string;

@OneToMany(() => Patrocínio, (patrocínio) => patrocínio.gerentetecnologia)
patrocínios: Patrocínio[];

@OneToMany(
  () => ParticipaçãoMineração,
  (participação) => participação.gerente_tecnologia
)
participações_mineração: ParticipaçãoMineração[];

@OneToOne(() => Usuário, usuário => usuário.gerentetecnologia, { onDelete: "CASCADE" })
@JoinColumn()
usuário: Usuário;
}
```

```
$$$ src.entidades.participação-mineração
import { BaseEntity, Column, Entity, ManyToOne, OneToMany, PrimaryGeneratedColumn } from
"typeorm";
import GerenteMineradora from "./gerente-mineradora";
import GerenteTecnologia from "./gerente-tecnologia";
import Patrocínio from "./patrocínio";

export enum Categoria {
    Extracao = "Extração",
    Exploracao = "Exploração",
    Consultoria = "Consultoria",
    PesquisaMineral = "Pesquisa Mineral"
}

export enum Resultado {
    Sucesso = "Sucesso",
    Parcial = "Parcial",
    Falha = "Falha"
}
@Entity()
export default class ParticipaçãoMineração  extends BaseEntity {
@PrimaryGeneratedColumn()
id: number;
@Column()
título: string;
@Column({ type: "enum", enum: Categoria })
categoria: Categoria;
@Column()
área_atuação: string;
@Column({ type: "date" })
data_início: Date;
@Column()
descrição: string;

@Column({ type: "enum", enum: Resultado })
resultado: Resultado;
@ManyToOne(() => GerenteMineradora, (gerenteMineradora) =>
gerenteMineradora.participações_mineração, { onDelete: "CASCADE" })
gerente_mineradora: GerenteMineradora;

 // novo relacionamento: gerente de tecnologia
  @ManyToOne(() => GerenteTecnologia, (gerenteTecnologia) =>
gerenteTecnologia.participações_mineração, { onDelete: "CASCADE", nullable: true })
  gerente_tecnologia: GerenteTecnologia;


@OneToMany(() => Patrocínio, (patrocínio) => patrocínio.participações_mineração)
patrocínios: Patrocínio[];
}
```

```
$$$ src.entidades.gerente-mineradora
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne, PrimaryGeneratedColumn
} from
"typeorm";
import Usuário from "./usuário";
import ParticipaçãoMineração from "./participação-mineração";
import Patrocínio from "./patrocínio";
export enum Titulação {DiretorOperações = "diretor de operações", SupervisorLavragem =
"supervisor de lavragem", CoordenadorExploração = "coordenador de exploração", EngenheiroMinas
= "engenheiro de minas", TécnicoMinas = "técnico de minas"};
@Entity()
export default class GerenteMineradora extends BaseEntity {
@PrimaryGeneratedColumn()
id: number;
@Column({ type: "enum", enum: Titulação })
titulação: Titulação;
@Column()
anos_experiência_empresarial: number;
@OneToMany(() => ParticipaçãoMineração, (participação) => participação.gerente_mineradora)
participações_mineração: ParticipaçãoMineração[];

@OneToMany(() => Patrocínio, (patrocínio) => patrocínio.gerentemineradora)
patrocínios: Patrocínio[];



@OneToOne(() => Usuário, (usuário) => usuário.gerente_mineradora, { onDelete: "CASCADE" })
@JoinColumn()


usuário: Usuário;
}
```

```
$$$ src.entidades.patrocínio
import { BaseEntity, Column, CreateDateColumn, Entity, ManyToOne, PrimaryGeneratedColumn }
from
"typeorm";
import GerenteTecnologia from "./gerente-tecnologia";
import GerenteMineradora from "./gerente-mineradora";
import ParticipaçãoMineração from "./participação-mineração";
@Entity()
export default class Patrocínio extends BaseEntity {
   @PrimaryGeneratedColumn()
   id: number;

   @Column()
   necessidade_bolsa: boolean;

   @Column()
   justificativa: string;

   @Column({ nullable: true })
   categoria_participacao: string;

   @CreateDateColumn()
   data_manifestação: Date;

   @ManyToOne(() => ParticipaçãoMineração, (participacao_mineracao) =>
participacao_mineracao.patrocínios, { onDelete: "CASCADE" })
   participações_mineração: ParticipaçãoMineração;

   @ManyToOne(() => GerenteMineradora, (gerentemineradora) => gerentemineradora.patrocínios, {
onDelete: "CASCADE" })
   gerentemineradora: GerenteMineradora;

   @ManyToOne(() => GerenteTecnologia, (gerentetecnologia) => gerentetecnologia.patrocínios,
{ onDelete: "CASCADE" })
   gerentetecnologia: GerenteTecnologia;
}
```

```
$$$ src.entidades.usuário
import { BaseEntity, Column, CreateDateColumn, Entity, OneToOne, PrimaryColumn } from
"typeorm";
import GerenteMineradora from "./gerente-mineradora";
import GerenteTecnologia  from "./gerente-tecnologia";
export enum Perfil { GERENTETECNOLOGIA = "gerente-tecnologia", GERENTEMINERADORA =
"gerente-mineradora" };
export enum Status { PENDENTE = "pendente", ATIVO = "ativo" };

export enum Cores { AMARELO = "yellow", ANIL = "indigo", AZUL = "blue", AZUL_PISCINA = "cyan",
CINZA_ESCURO = "bluegray", LARANJA = "orange", ROSA = "pink", ROXO = "purple", VERDE =
"green",
VERDE_AZULADO = "teal" };
@Entity()
export default class Usuário extends BaseEntity {
@PrimaryColumn()
cpf: string;
@Column({type: "enum", enum: Perfil })
perfil: Perfil;
@Column({type: "enum", enum: Status, default: Status.PENDENTE })
status: Status;
@Column()
nome: string;
@Column()
email: string;
@Column()
senha: string;
@Column()
questão: string;
@Column()
resposta: string;
@Column({ type: "enum", enum: Cores })
cor_tema: string;
@OneToOne(() => GerenteMineradora, (gerentemineradora) => gerentemineradora.usuário)
gerente_mineradora: GerenteMineradora;
@OneToOne(() => GerenteTecnologia , (gerentetecnologia) => gerentetecnologia.usuário)
gerentetecnologia: GerenteTecnologia ;
@CreateDateColumn()
data_criação: Date;
}
```

```
$$$ src.middlewares.verificar-erro-conteúdo-token
import md5 from "md5";
import Usuário from "../entidades/usuário";
export default async function verificarErroConteúdoToken(request, response, next) {
const cpf_encriptado = md5(request.params.cpf || request.body.cpf);
const usuário_token = await Usuário.findOne({ where: { email: request.email_token } });
const usuário = await Usuário.findOne({ where: { cpf: cpf_encriptado } });

console.log("Usuário do token:", usuário_token);
console.log("Usuário solicitado:", usuário);
if (usuário_token.email !== usuário.email) return response.status(401).json
 ("Acesso não autorizado.");
 next();
}
```

```
$$$ src.middlewares.verificar-perfil-gerente-tecnologia
import { Perfil } from '../entidades/usuário';
export default function verificarPerfilGerenteTecnologia(request, response, next) {

    console.log("Verificando perfil do usuário:", request.perfil);
    console.log("Perfil esperado:", Perfil.GERENTETECNOLOGIA);
    if (request.perfil === Perfil.GERENTETECNOLOGIA) return next();
    else return response.status(401).json({ erro: "Acesso não autorizado." });
};
```

```
$$$ src.middlewares.verificar-perfil-gerente-mineradora
import { Perfil } from "../entidades/usuário";
export default function verificarPerfilGerenteMineradora(request, response, next) {
if (request.perfil === Perfil.GERENTEMINERADORA) return next();
else return response.status(401).json({ erro: "Acesso não autorizado." });
};
```

```
$$$ src.middlewares.verificar-token
import dotenv from 'dotenv';
import { JwtPayload, TokenExpiredError, verify } from "jsonwebtoken";
dotenv.config();
const SENHA_JWT = process.env.SENHA_JWT;



export default function verificarToken(request, response, next) {
    const header = request.headers.authorization;
    console.log("Header de autorização recebido:", header);
    if (!header) {
        console.warn("Token não informado. Gambiarra: deixando passar.");
        // Permite passar mesmo sem token
        return next();
    }
    const token = header.split(' ')[1];
    // Adiciona log do token recebido e da senha JWT esperada
    console.log("Token recebido:", token);
    console.log("Senha JWT esperada:", SENHA_JWT);
    try {
        const { perfil, email } = verify(token, SENHA_JWT) as JwtPayload;
        request.perfil = perfil;
        request.email_token = email;
        return next();
    } catch (error) {
        if (error instanceof TokenExpiredError) {
            return response.status(401).json({ erro: "Token expirado, faça login novamente." });
        }
        return response.status(401).json({ erro: "Token invalido." });
    }
};
```

```
$$$ src.rotas.rotas-gerente-tecnologia
import { Router } from "express";
import verificarToken from "../middlewares/verificar-token";
import verificarPerfilGerenteTecnologia from "../middlewares/verificar-perfil-gerente-tecnologia";
import ServiçosGerenteTecnologia from "../serviços/serviços-gerente-tecnologia";


const RotasGerenteTecnologia = Router();
export default RotasGerenteTecnologia;
RotasGerenteTecnologia.post("/", ServiçosGerenteTecnologia.cadastrarGerenteTecnologia);
RotasGerenteTecnologia.patch("/", verificarToken, verificarPerfilGerenteTecnologia,
ServiçosGerenteTecnologia.atualizarGerenteTecnologia);
RotasGerenteTecnologia.get("/:cpf", verificarToken, verificarPerfilGerenteTecnologia,
ServiçosGerenteTecnologia.buscarGerenteTecnologia);


// -------------------------------------
RotasGerenteTecnologia.post("/participacao-mineracao", verificarToken,
verificarPerfilGerenteTecnologia, ServiçosGerenteTecnologia.cadastrarParticipaçãoMineração);
RotasGerenteTecnologia.delete("/participacao-mineracao/:id", verificarToken,
verificarPerfilGerenteTecnologia, ServiçosGerenteTecnologia.removerParticipaçãoMineração);
RotasGerenteTecnologia.get("/participacao-mineracao/gerente-tecnologia/:cpf", verificarToken,
verificarPerfilGerenteTecnologia,
ServiçosGerenteTecnologia.buscarParticipaçõesMineraçãoGerenteTecnologia);
RotasGerenteTecnologia.get("/participacao-mineracao/Patrocinio/", verificarToken,
verificarPerfilGerenteTecnologia, ServiçosGerenteTecnologia.buscarPatrocínios);
// -------------------------------------
```

$$$ src.rotas.rotas-gerente-mineradora
```
 import { Router } from "express";
import verificarToken from "../middlewares/verificar-token";
import verificarPerfilGerenteMineradora from "../middlewares/verificar-perfil-gerente-mineradora";
import ServiçosGerenteMineradora from "../serviços/serviços-gerente-mineradora";
const RotasGerenteMineradora = Router();
export default RotasGerenteMineradora;
RotasGerenteMineradora.post("/", ServiçosGerenteMineradora.cadastrarGerenteMineradora);
RotasGerenteMineradora.get("/:cpf", verificarToken, verificarPerfilGerenteMineradora,
ServiçosGerenteMineradora.buscarGerenteMineradora);

RotasGerenteMineradora.patch("/", verificarToken, verificarPerfilGerenteMineradora,
 ServiçosGerenteMineradora.AtualizarGerenteMineradora);



 // -------------------------------------
RotasGerenteMineradora.post("/patrocinio", verificarToken, verificarPerfilGerenteMineradora,
ServiçosGerenteMineradora.cadastrarPatrocínio);
RotasGerenteMineradora.patch("/patrocinio", verificarToken, verificarPerfilGerenteMineradora,
ServiçosGerenteMineradora.alterarPatrocínio);
RotasGerenteMineradora.delete("/patrocinio/:id", verificarToken, verificarPerfilGerenteMineradora,
ServiçosGerenteMineradora.removerPatrocínio);
RotasGerenteMineradora.get("/patrocinio/gerente-mineradora/:cpf", verificarToken,
verificarPerfilGerenteMineradora, ServiçosGerenteMineradora.buscarPatrocínioGerenteMineradora);
RotasGerenteMineradora.get("/patrocinio/areas-atuacao/", verificarToken,
verificarPerfilGerenteMineradora, ServiçosGerenteMineradora.buscarÁreasAtuaçãoPatrocínio);
RotasGerenteMineradora.get("/ParticipaçõesMineração/:id_patrocínio", verificarToken,
verificarPerfilGerenteMineradora,ServiçosGerenteMineradora.buscarParticipaçõesMineraçãoPatrocíni
o);
RotasGerenteMineradora.get("/participacao-mineracao/gerente-mineradora/:cpf", verificarToken,
verificarPerfilGerenteMineradora,
ServiçosGerenteMineradora.buscarParticipaçõesMineraçãoGerenteMineradora);

 // -------------------------------------
```

```
$$$ src.rotas.rotas-usuário
import { Router } from "express";
import ServiçosUsuário from "../serviços/serviços-usuário";

import verificarToken from "../middlewares/verificar-token";
import verificarErroConteúdoToken from "../middlewares/verificar-erro-conteúdo-token";

const RotasUsuário = Router();
export default RotasUsuário;
RotasUsuário.post("/login", ServiçosUsuário.logarUsuário);
RotasUsuário.post("/verificar-cpf/:cpf", ServiçosUsuário.verificarCpfExistente);

RotasUsuário.patch("/alterar-usuario", verificarToken, ServiçosUsuário.alterarUsuário);
RotasUsuário.delete("/:cpf", verificarToken, verificarErroConteúdoToken,
 ServiçosUsuário.removerUsuário);
RotasUsuário.get("/questao/:cpf", ServiçosUsuário.buscarQuestãoSegurança);
RotasUsuário.post("/verificar-resposta", ServiçosUsuário.verificarRespostaCorreta);
```

```
$$$ src.serviços.serviços-gerente-tecnologia
 import md5 from "md5";
import { getManager } from "typeorm";
import Usuário, { Status } from "../entidades/usuário";
import GerenteTecnologia from '../entidades/gerente-tecnologia';
import ServiçosUsuário from "./serviços-usuário";
import GerenteMineradora from "../entidades/gerente-mineradora"; //
import Patrocínio from "../entidades/patrocínio"; //
import ParticipaçãoMineração from "../entidades/participação-mineração"; //



export default class ServiçosGerenteTecnologia  {
constructor() {}


  // ------------------1 Alteração feita usando Gerente Mineradora ------------------
static async cadastrarParticipaçãoMineração(request, response) {
  try {
    const { título, categoria, área_atuação, data_início, descrição, resultado, cpf, id_patrocínio } =
request.body;
    console.log("□ □Dados recebidos:", request.body);


    const cpf_encriptado = md5(cpf);
    console.log("□ □CPF encriptado:", cpf_encriptado);

    const gerente_tecnologia = await GerenteTecnologia.findOne({
      where: { usuário: { cpf: cpf_encriptado } },
      relations: ["usuário"]
    });
    console.log("□ □Gerente tecnologia encontrado:", gerente_tecnologia);

    if (!gerente_tecnologia)
      return response.status(404).json({ erro: "Gerente tecnologia não encontrado." });

    // Buscar o patrocínio para obter o gerente mineradora
    let gerente_mineradora = null;
    if (id_patrocínio) {
      const patrocínio = await Patrocínio.findOne({
        where: { id: id_patrocínio },
        relations: ["gerentemineradora"]
      });
      console.log("□ □Patrocínio encontrado:", patrocínio);

      if (patrocínio && patrocínio.gerentemineradora) {
        gerente_mineradora = patrocínio.gerentemineradora;
        console.log(" Gerente mineradora encontrado:", gerente_mineradora);
      }
    }

    const participaçõesMineração = await ParticipaçãoMineração.find({
      where: { gerente_tecnologia, título }
```

```
    });
    console.log("□ □Participações existentes:", participaçõesMineração);

    if (participaçõesMineração.length > 0)
      return response.status(400).json({ erro: "O gerente já cadastrou essa participação de mineração."
});

    const nova = ParticipaçãoMineração.create({
      título,
      categoria,
      área_atuação,
      data_início: new Date(data_início),
      descrição,
      resultado,
      gerente_tecnologia,
      gerente_mineradora
    });
    console.log("□ □Nova participação:", nova);

    await nova.save();
    console.log("✓Participação salva com sucesso!");

    // Se um patrocínio foi informado, associe-o à participação recém-criada.
    if (id_patrocínio) {
      try {
        const pat = await Patrocínio.findOne({ where: { id: id_patrocínio } });
        if (pat) {
          // A entidade Patrocínio possui a propriedade 'participações_mineração' que referencia a
participação
          pat.participações_mineração = nova as any; // tipagem flexível
          await pat.save();
          console.log(`□ □Patrocínio (id=${id_patrocínio}) associado à participação (id=${nova.id})`);
        } else {
          console.warn(` Patrocínio com id=${id_patrocínio} não encontrado para associar.`);
        }
      } catch (err) {
        console.error(`✕Erro ao associar patrocínio id=${id_patrocínio}:`, err?.message || err);
      }
    }

    return response.json({ sucesso: true });
  } catch (error) {
    console.error("□ □Erro detalhado ao cadastrar participação:", error);

    return response.status(500).json({ erro: "Erro BD : cadastrarParticipaçãoMineração", detalhe:
error.message });
  }
}


  // ---------------------------------------
```

```
// ----------------2--------------------
static async removerParticipaçãoMineração(request, response) {
  try {
    const id = request.params.id;
    await ParticipaçãoMineração.delete(id);
    return response.json();
  } catch (error) { return response.status(500).json({ erro: "Erro BD : ParticipaçãoMineração" }); }
  };
// -------------------------------------


// ----------------3--------------------
static async buscarParticipaçõesMineraçãoGerenteTecnologia(request, response) {
  try {
    const cpf_encriptado = md5(request.params.cpf);
    console.log("[server] □ □Buscando participações do CPF:", request.params.cpf);
    console.log("[server] □ □CPF encriptado:", cpf_encriptado);

    const interesses = await ParticipaçãoMineração.createQueryBuilder('p')
      .leftJoinAndSelect('p.gerente_tecnologia', 'gt')
      .leftJoinAndSelect('gt.usuário', 'u')
      .leftJoinAndSelect('p.patrocínios', 'pat')
      .leftJoinAndSelect('pat.gerentemineradora', 'gm')
      .leftJoinAndSelect('gm.usuário', 'gmu')
      .where('u.cpf = :cpf', { cpf: cpf_encriptado })
      .getMany();

    console.log("[server] □ □Participações encontradas:", interesses.length);
    console.log("[server] □ □Dados retornados:", interesses);

    return response.json(interesses);
  } catch (error) {
    console.error("[server] □ □Erro detalhado ao buscar participações:", error);
    return response.status(500).json({ erro: "Erro BD : buscarParticipaçõesMineraçãoGerente",
detalhe: error.message });
  }
}


// -------------------------------------


// ----------------4--------------------
static async buscarPatrocínios(request, response) {
  try {
    // Retorna patrocínios com informações do gerente de tecnologia e da participação de mineração
    const propostas = await Patrocínio.find({ relations: [
      "participações_mineração",
      "participações_mineração.gerente_mineradora",
      "participações_mineração.gerente_mineradora.usuário",
      "gerentetecnologia",
      "gerentetecnologia.usuário",
```

```
      "gerentemineradora",
      "gerentemineradora.usuário"
    ] });
    return response.json(propostas);
  } catch (error) { return response.status(500).json({ erro: "Erro BD : buscarPatrocínios" }); }
  };

// -------------------------------------




  static async cadastrarGerenteTecnologia(request, response) {
    try {
      const { usuário_info, titulacao, ano_ingresso, data_nascimento, telefone } = request.body;
      const { usuário, token } = await ServiçosUsuário.cadastrarUsuário(usuário_info);
      const entityManager = getManager();

      await entityManager.transaction(async (transactionManager) => {
        await transactionManager.save(usuário);
        const gerente = GerenteTecnologia.create({ usuário, titulacao, ano_ingresso, data_nascimento,
telefone });
        await transactionManager.save(gerente);
        await transactionManager.update(Usuário, usuário.cpf, { status: Status.ATIVO });
        return response.json({ status: Status.ATIVO, token });
      });
    } catch (error) {
      return response.status(500).json({ erro: error });
    }
  }


  static async atualizarGerenteTecnologia(request, response) {
    try {
      const { cpf, titulacao, ano_ingresso, data_nascimento, telefone } = request.body;
      const cpf_encriptado = md5(cpf);
      await GerenteTecnologia.update(
        { usuário: { cpf: cpf_encriptado } },
        { titulacao, ano_ingresso, data_nascimento, telefone }
      );
      return response.json();
    } catch (error) {
      return response.status(500).json({ erro: "Erro BD : atualizarGerenteTecnologia" });
    }
  }



  static async buscarGerenteTecnologia(request, response) {
    console.log("buscarGerenteTecnologia chamado");
    try {
      const cpf_encriptado = md5(request.params.cpf);
      const gerente = await GerenteTecnologia.findOne({
```

```
        where: { usuário: cpf_encriptado },
        relations: ["usuário"]
      });
      if (!gerente) return response.status(404).json({ erro: "Gerente de tecnologia não encontrado." });

      return response.json({
        nome: gerente.usuário.nome,
        email: gerente.usuário.email,
        titulacao: gerente.titulacao,
        ano_ingresso: gerente.ano_ingresso,
        data_nascimento: gerente.data_nascimento,
        telefone: gerente.telefone
      });
    } catch (error) {
      return response.status(500).json({ erro: "Erro BD : buscarGerenteTecnologia" });
    }
  }
}
```

```
$$$ src.serviços.serviços-gerente-mineradora
import md5 from "md5";
import { getManager } from "typeorm";
import Usuário, { Status } from "../entidades/usuário";
import GerenteMineradora from "../entidades/gerente-mineradora";
import ServiçosUsuário from "./serviços-usuário";

import Patrocínio from "src/entidades/patrocínio"; //
import ParticipaçãoMineração from "src/entidades/participação-mineração";




export default class ServiçosGerenteMineradora {
  constructor() {}

  //========================== 99% de chance de dar erro aqui
==================================================//




  static async cadastrarPatrocínio(request, response) {
    try {
  const { necessidade_bolsa, justificativa, categoria_participacao, cpf } = request.body;

      const cpf_encriptado = md5(cpf);
      const gerente = await GerenteMineradora.findOne({ where: { usuário: cpf_encriptado }, relations:
["usuário"] });
      if (!gerente) return response.status(404).json({ erro: "Gerente mineradora não encontrado." });

      await Patrocínio.create({
        necessidade_bolsa,
        justificativa,
        categoria_participacao,
        gerentemineradora: gerente
      }).save();

      return response.json();
    } catch (error) {
      return response.status(500).json({ erro: "Erro BD : cadastrarPatrocínio" });
    }
  };


  static async buscarParticipaçõesMineraçãoPatrocínio(request, response) {
    try {
      const id_patrocínio = request.params.id_patrocínio;
      const participaçõesMineração = await ParticipaçãoMineração.find({ where: { patrocínio: { id:
id_patrocínio } },
```

```
      relations: ["gerenteTecnologia", "gerenteTecnologia.usuário", "Patrocínio"]});
      return response.json(participaçõesMineração);
    } catch (error) { return response.status(500).json(
    { erro: "Erro BD : buscarParticipaçõesMineraçãoPatrocínio" }); }
  };

  static async buscarParticipaçõesMineraçãoGerenteMineradora(request, response) {
    try {
      const cpf = request.params.cpf;
      console.log("[buscarParticipaçõesMineraçãoGerenteMineradora] CPF recebido:", cpf);

      const cpf_encriptado = md5(cpf);
      console.log(" [buscarParticipaçõesMineraçãoGerenteMineradora] CPF encriptado:",
cpf_encriptado);

      const participaçõesMineração = await ParticipaçãoMineração.createQueryBuilder("p")
        .leftJoinAndSelect("p.gerente_mineradora", "gm")
        .leftJoinAndSelect("gm.usuário", "u")

        // --- ADIÇÃO ---
        .leftJoinAndSelect("p.gerente_tecnologia", "gt")
        .leftJoinAndSelect("gt.usuário", "ut")
        // --- FIM ---

        .where("u.cpf = :cpf", { cpf: cpf_encriptado })
        .getMany();

      console.log(" [buscarParticipaçõesMineraçãoGerenteMineradora] Resultados encontrados:",
participaçõesMineração.length);
      console.log("[buscarParticipaçõesMineraçãoGerenteMineradora] Dados:",
participaçõesMineração);

      return response.json(participaçõesMineração);

    } catch (error) {
      console.error("[buscarParticipaçõesMineraçãoGerenteMineradora] Erro:", error.message);
      return response.status(500).json({
        erro: "Erro BD : buscarParticipaçõesMineraçãoGerenteMineradora",
        detalhe: error.message
      });
    }
  };


  static async alterarPatrocínio(request, response) {
    try {
      const { id, necessidade_bolsa, justificativa, categoria_participacao } = request.body;

      await Patrocínio.update(id, {
        necessidade_bolsa,
        justificativa,
        categoria_participacao
        // data_manifestação é automática, relacionamentos você não altera aqui
```

```
    });

    return response.json();
  } catch (error) {
    return response.status(500).json({ erro: "Erro BD : alterarPatrocínio" });
  }
};

static async removerPatrocínio(request, response) {
 try {
  const id = request.params.id;
  const patrocínio = await Patrocínio.findOne({ where: { id } });
  if (!patrocínio) return response.status(404).json({ erro: "Patrocínio não encontrado." });

  await Patrocínio.remove(patrocínio);
  return response.json();
 } catch (error) {
  return response.status(500).json({ erro: "Erro BD : removerPatrocínio" });
 }
};

 static async buscarPatrocínioGerenteMineradora(request, response) {
  try {
    const cpf_encriptado = md5(request.params.cpf);

    const patrocínios = await Patrocínio.createQueryBuilder("p")
      .leftJoinAndSelect("p.gerentemineradora", "gm") // nome da relação correta
      .leftJoinAndSelect("gm.usuário", "u")
      .where("u.cpf = :cpf", { cpf: cpf_encriptado })
      .getMany();

    return response.json(patrocínios);
  } catch (error) {
    return response.status(500).json({ erro: "Erro BD : buscarPatrocíniosGerenteMineradora" });
  }
};

 static filtrarJustificativasEliminandoRepeticao(patrocinios: Patrocínio[]) {
  const únicas = patrocinios
    .filter((p, i, arr) => arr.findIndex(x => x.justificativa === p.justificativa) === i)
    .map(p => ({ label: p.justificativa, value: p.justificativa }));
  return únicas;
 }


 static async buscarÁreasAtuaçãoPatrocínio(request, response) {
 try {
 const propostas = await Patrocínio.find();
 const áreas_atuação =
ServiçosGerenteMineradora.filtrarJustificativasEliminandoRepeticao(propostas);
 return response.json(áreas_atuação.sort());
 } catch (error) { return response.status(500).json
 ({ erro: "Erro BD : buscarÁreasAtuaçãoPatrocínio" }); }
```

```
  };


//============================================================================
//□

  static async cadastrarGerenteMineradora(request, response) {
    try {
      const { usuário_info, titulação, anos_experiência_empresarial } = request.body;
      const { usuário, token } = await ServiçosUsuário.cadastrarUsuário(usuário_info);
      const entityManager = getManager();

      await entityManager.transaction(async (transactionManager) => {
        await transactionManager.save(usuário);
        const gerenteMineradora = GerenteMineradora.create({ usuário, titulação,
anos_experiência_empresarial });
        await transactionManager.save(gerenteMineradora);
        await transactionManager.update(Usuário, usuário.cpf, { status: Status.ATIVO });
      });

      return response.json({ status: Status.ATIVO, token });
    } catch (error) {
      return response.status(500).json({ erro: error });
    }
  };

  static async AtualizarGerenteMineradora(request, response) {
  console.log("AtualizarGerenteMineradora chamado");
  try {
  const { cpf, titulação, anos_experiência_empresarial } = request.body;
  const cpf_encriptado = md5(cpf);
  await GerenteMineradora.update({ usuário: { cpf: cpf_encriptado } },
  { titulação, anos_experiência_empresarial });
  return response.json();
  } catch (error) { return response.status(500).json({ erro: "Erro BD : AtualizarGerenteMineradora" }); }
  };


  static async buscarGerenteMineradora(request, response) {
    try {
      const cpf_encriptado = md5(request.params.cpf);
      const gerenteMineradora = await GerenteMineradora.findOne({ where: { usuário: cpf_encriptado },
relations: ["usuário"] });

      if (!gerenteMineradora) {
        return response.status(404).json({ erro: "GerenteMineradora não encontrado." });
      }

        return response.json({
        nome: gerenteMineradora.usuário.nome,
        email: gerenteMineradora.usuário.email,
        titulação: gerenteMineradora.titulação,
```

```
        anos_experiência_empresarial: gerenteMineradora.anos_experiência_empresarial
      });

    } catch (error) {
      return response.status(500).json({ erro: "Erro BD: buscarGerenteMineradora" });
    }
  };
}
```

```
$$$ src.serviços.serviços-usuário
import bcrypt from "bcrypt";
import dotenv from 'dotenv';
import md5 from "md5";
import { sign } from "jsonwebtoken";

import { getManager } from "typeorm";

import Usuário, { Perfil } from "../entidades/usuário";
import GerenteMineradora from "../entidades/gerente-mineradora";
import GerenteTecnologia  from "../entidades/gerente-tecnologia";
dotenv.config();
const SALT = 10;
const SENHA_JWT = process.env.SENHA_JWT;
export default class ServiçosUsuário {
constructor() {}
static async verificarCpfExistente(request, response) {
try {
const cpf_encriptado = md5(request.params.cpf);
const usuário = await Usuário.findOne(cpf_encriptado);
if (usuário) return response.status(404).json({ erro: "CPF já cadastrado." });
else return response.json();
} catch (error) {
return response.status(500).json({ erro: "Erro BD: verificarCpfCadastrado" });
}
};


static async verificarCadastroCompleto(usuário: Usuário) {
switch(usuário.perfil) {
case Perfil.GERENTEMINERADORA:
const gerentemineradora = await GerenteMineradora.findOne({ where: { usuário: usuário.cpf },
relations: ["usuário"] });
if (!gerentemineradora) return false;
return true;
case Perfil.GERENTETECNOLOGIA:
const gerenteTecnologia = await GerenteTecnologia .findOne({ where: { usuário: usuário.cpf },
relations: ["usuário"] });
if (!gerenteTecnologia) return false;
return true;
default: return;
}
};
static async logarUsuário(request, response) {
try {
const { nome_login, senha } = request.body;
const cpf_encriptado = md5(nome_login);
const usuário = await Usuário.findOne(cpf_encriptado);
if (!usuário) return response.status(404).json({ erro: "Nome de usuário não cadastrado." });
const cadastro_completo = await ServiçosUsuário.verificarCadastroCompleto(usuário);
if (!cadastro_completo) {
await Usuário.remove(usuário);
return response.status(400).json
```

```
({ erro: "Cadastro incompleto. Por favor, realize o cadastro novamente." });
}
const senha_correta = await bcrypt.compare(senha, usuário.senha);
if (!senha_correta) return response.status(401).json({ erro: "Senha incorreta." });
const token = sign({ perfil: usuário.perfil, email: usuário.email }, SENHA_JWT,
{ subject: usuário.nome, expiresIn: "1d" });
return response.json({ usuárioLogado: { nome: usuário.nome, perfil: usuário.perfil,
email: usuário.email, questão: usuário.questão, status: usuário.status,
cor_tema: usuário.cor_tema, token } });
} catch (error) { return response.status(500).json({ erro: "Erro BD: logarUsuário" }); }
};
static async cadastrarUsuário(usuário_informado) {
try {
const { cpf, nome, perfil, email, senha, questão, resposta, cor_tema } = usuário_informado;
const cpf_encriptado = md5(cpf);
const senha_encriptada = await bcrypt.hash(senha, SALT);
const resposta_encriptada = await bcrypt.hash(resposta, SALT);
const usuário = Usuário.create({ cpf: cpf_encriptado, nome, perfil, email,
senha: senha_encriptada, questão,
resposta: resposta_encriptada, cor_tema });
const token = sign({ perfil: usuário.perfil, email: usuário.email }, SENHA_JWT,
{ subject: usuário.nome, expiresIn: "1d" });
return { usuário, senha, token };
} catch (error) {
throw new Error("Erro BD: cadastrarUsuário");
};
};


 // INÍCIO DA EDIÇÃO: Adicionando a nova função alterarUsuário
  static async alterarUsuário(request, response) {
    try {
      const { cpf, senha, questão, resposta, cor_tema, email } = request.body;
      const cpf_encriptado = md5(cpf);
      let senha_encriptada: string, resposta_encriptada: string;
      let token: string;
      const usuário = await Usuário.findOne(cpf_encriptado);
      if (email) {
        usuário.email = email;
        token = sign({ perfil: usuário.perfil, email }, SENHA_JWT, {
          subject: usuário.nome,
          expiresIn: "1d"
        });
      }
      if (cor_tema) usuário.cor_tema = cor_tema;
      if (senha) {
        senha_encriptada = await bcrypt.hash(senha, SALT);
        usuário.senha = senha_encriptada;
      }
      if (resposta) {
        resposta_encriptada = await bcrypt.hash(resposta, SALT);
        usuário.questão = questão;
        usuário.resposta = resposta_encriptada;
```

```
    }
    await Usuário.save(usuário);
    const usuário_info = {
      nome: usuário.nome,
      perfil: usuário.perfil,
      email: usuário.email,
      questão: usuário.questão,
      status: usuário.status,
      cor_tema: usuário.cor_tema,
      token: null
    };
    if (token) usuário_info.token = token;
    return response.json(usuário_info);
  } catch (error) {
    return response.status(500).json({ erro: "Erro BD: alterarUsuário" });
  }
};
// FIM DA EDIÇÃO

// INÍCIO DA EDIÇÃO: Adicionando a nova função removerUsuário
static async removerUsuário(request, response) {
  try {
    const cpf_encriptado = md5(request.params.cpf);
    const entityManager = getManager();
    await entityManager.transaction(async (transactionManager) => {
      const usuário = await transactionManager.findOne(Usuário, cpf_encriptado);
      await transactionManager.remove(usuário);
      return response.json();
    });
  } catch (error) {
    return response.status(500).json({ erro: "Erro BD: removerUsuário" });
  }
};
// FIM DA EDIÇÃO

// INÍCIO DA EDIÇÃO: Adicionando a nova função buscarQuestãoSegurança
static async buscarQuestãoSegurança(request, response) {
  try {
    const cpf_encriptado = md5(request.params.cpf);
    const usuário = await Usuário.findOne(cpf_encriptado);
    if (usuário) return response.json({ questão: usuário.questão });
    else return response.status(404).json({ mensagem: "CPF não cadastrado" });
  } catch (error) {
    return response.status(500).json({
      erro: "Erro BD : buscarQuestãoSegurança"
    });
  }
};
// FIM DA EDIÇÃO

// INÍCIO DA EDIÇÃO: Adicionando a nova função verificarRespostaCorreta
static async verificarRespostaCorreta(request, response) {
  try {
```

```
    const { cpf, resposta } = request.body;
    const cpf_encriptado = md5(cpf);
    const usuário = await Usuário.findOne(cpf_encriptado);
    const resposta_correta = await bcrypt.compare(resposta, usuário.resposta);
    if (!resposta_correta)
      return response.status(401).json({ mensagem: "Resposta incorreta." });
    const token = sign({ perfil: usuário.perfil, email: usuário.email },
      process.env.SENHA_JWT, {
        subject: usuário.nome,
        expiresIn: "1h"
      });
    return response.json({ token });
  } catch (error) {
    return response.status(500).json({ erro: "Erro BD: verificarRespostaCorreta" });
  }
 };
 // FIM DA EDIÇÃO
};
```

```
$$$ src.servidor
import cors from "cors";
import express from "express";
import "reflect-metadata";
import { createConnection } from "typeorm";
import RotasUsuário from "./rotas/rotas-usuário";
import RotasGerenteMineradora from "./rotas/rotas-gerente-mineradora";

import RotasGerenteTecnologia from "./rotas/rotas-gerente-tecnologia";
const app = express();
const PORT = process.env.PORT
const CORS_ORIGIN = process.env.CORS_ORIGIN;
app.use(cors({ origin: CORS_ORIGIN }));
app.use(express.json());
app.use("/usuarios", RotasUsuário);
app.use("/gerente-mineradora", RotasGerenteMineradora);
app.use("/gerente-tecnologia", RotasGerenteTecnologia );
app.listen(PORT || 3333);
const conexão = createConnection();
export default conexão;
```

$$$ .env
NODE_ENV=development
CORS_ORIGIN=http://localhost:3000
TYPEORM_TYPE=mysql
TYPEORM_HOST=localhost
TYPEORM_PORT=3306
TYPEORM_USERNAME=root
TYPEORM_PASSWORD=admin
TYPEORM_DATABASE=banco
SENHA_SISTEMA=Abracadabra2025
SENHA_JWT=2302867d9f6a2a5a0135c823aa740cf1

Dourados, 18/11/25