

\$\$\$ src.entidades.interessado

```
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne,
PrimaryGeneratedColumn }
from "typeorm";
import Usuário from "../usuário";
import Patrocínio from "../patrocínio";
export enum Titulacao { EC = "Engenharia de Computação", SI = "Sistemas de
Informação" };
@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[];

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

\$\$\$ src.entidades.interesse

```
import { BaseEntity, Column, Entity, ManyToOne, OneToMany,
PrimaryGeneratedColumn } from "typeorm";
import GerenteMineradora from "../gerente-mineradora";
import Patrocínio from "../patrocínio";

export enum Categoria {
  Extracao = "Extração",
```

```

    Exploracao = "Exploração",
    Consultoria = "Consultoria",
    PesquisaMineral = "PesquisaMineral"
}

export enum Resultado {
    Sucesso = "Sucesso",
    Parcial = "Parcial",
    Falha = "Falha"
}

@Entity()
export default class ParticipacaoMineracao 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.participacoes_mineracao, { onDelete: "CASCADE" })
    gerentemineradora: GerenteMineradora;
    @OneToMany(() => Patrocínio, (patrocinio) => patrocinio.participacao_mineracao)
    patrocinios: Patrocínio[];
}

```

\$\$\$ src.entidades.proponente

```

import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne,
PrimaryGeneratedColumn } from
"typeorm";
import Usuário from "../usuário";
import ParticipacaoMineração from "../participação-mineração";
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(() => ParticipacaoMineração, (participacao) =>
    participacao.gerentemineradora)
  participacoes_mineracao: ParticipacaoMineração[];
  @OneToOne(() => Usuário, (usuário) => usuário.gerentemineradora, { onDelete:
    "CASCADE" })
  @JoinColumn()

  usuário: Usuário;
}

```

\$\$\$ src.entidades.proposta

```

import { BaseEntity, Column, CreateDateColumn, Entity, ManyToOne,
  PrimaryGeneratedColumn } from
  "typeorm";
import GerenteTecnologia from "../gerente-tecnologia";
import ParticipacaoMineração from "../participação-mineração";
@Entity()
export default class Patrocínio extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  necessidade_bolsa: boolean;

  @Column()
  justificativa: string;

  @CreateDateColumn()
  data_manifestação: Date;

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

```

```

    @ManyToOne() => GerenteTecnologia, (gerentetecnologia) =>
gerentetecnologia.patrocinios, { 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 = "gerentetecnologia",
GERENTEMINERADORA = "gerente_Tecnologia" };
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)

```

```

gerentemineradora: GerenteMineradora;
@OneToOne() => GerenteTecnologia , (gerentetecnologia) =>
gerentetecnologia.usuario)
gerentetecnologia: GerenteTecnologia ;
@CreateDateColumn()
data_criação: Date;
}

```

\$\$\$ src.middlewares.verificar-perfil-proponente

```

import { Perfil } from "../entidades/usuario";
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;
  if (!header) return response.status(401).json({ erro: "Token nao informado." });
  const token = header.split(' ')[1];
  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.proponente

```

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);

```

\$\$\$ src.rotas.usuário

```

import { Router } from "express";
import ServiçosUsuário from "../serviços/serviços-usuário";
const RotasUsuário = Router();
export default RotasUsuário;
RotasUsuário.post("/login", ServiçosUsuário.logarUsuário);
RotasUsuário.post("/verificar-cpf/:cpf", ServiçosUsuário.verificarCpfExistente);

```

\$\$\$ src.serviços.proponente

```

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

export default class Serviçosgerentemineradora {
  constructor() {}

  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 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."
});
    }

    // O erro estava na linha abaixo, faltando uma propriedade ou tendo uma vírgula
    extra.
    // O código correto inclui a propriedade 'anos_experiência_empresarial' e fecha o
    objeto.
    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.usuário

```

import bcrypt from "bcrypt";
import dotenv from 'dotenv';
import md5 from "md5";
import { sign } from "jsonwebtoken";
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_encryptado = md5(request.params.cpf);
      const usuário = await Usuário.findOne(cpf_encryptado);
      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 loginUsuário(request, response) {
    try {
      const { nome_login, senha } = request.body;
      const cpf_encryptado = md5(nome_login);
      const usuário = await Usuário.findOne(cpf_encryptado);
      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_encryptado = md5(cpf);
const senha_encryptada = await bcrypt.hash(senha, SALT);
const resposta_encryptada = await bcrypt.hash(resposta, SALT);
const usuário = Usuário.create({ cpf: cpf_encryptado, nome, perfil, email,
senha: senha_encryptada, questão,
resposta: resposta_encryptada, 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");
};
};
};
};

```

\$\$\$ 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";
const app = express();
const PORT = process.env.PORT

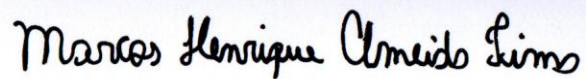
```

```
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.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, 12/09/25

A handwritten signature in black ink on a light blue background. The signature reads "Marcos Henrique Almeida Lima" in a cursive script.