

Desenvolvimento de Aplicações Web - Entrega 2 - 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";
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[];

  @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 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;

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

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

```

$$$ src.entidades.usuario
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.usuario)
  gerente_mineradora: GerenteMineradora;
  @OneToOne(() => GerenteTecnologia, (gerentetecnologia) => gerentetecnologia.usuario)
  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_encryptado = 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_encryptado } });

  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 ServicosGerenteTecnologia from "../servicos/servicos-gerente-tecnologia";

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

```
$$$ 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);
```

```
$$$ 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";
export default class ServiçosGerenteTecnologia {
  constructor() {}

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

      await entityManager.transaction(async (transactionManager) => {
        await transactionManager.save(usuario);
        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_encryptado = md5(cpf);
    await GerenteTecnologia.update(
      { usuário: { cpf: cpf_encryptado } },
      { 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_encryptado = md5(request.params.cpf);
    const gerente = await GerenteTecnologia.findOne({
      where: { usuário: cpf_encryptado },
      relations: ["usuário"]
    });
    if (!gerente) return response.status(404).json({ erro: "Gerente de tecnologia não encontrado." });

    return response.json({
      nome: gerente.usuario.nome,
      email: gerente.usuario.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";

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 AtualizarGerenteMineradora(request, response) {
    console.log("AtualizarGerenteMineradora chamado");
    try {
      const { cpf, titulação, anos_experiência_empresarial } = request.body;
      const cpf_encryptado = md5(cpf);
      await GerenteMineradora.update({ usuário: { cpf: cpf_encryptado } },
        { 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_encryptado = md5(request.params.cpf);
      const gerenteMineradora = await GerenteMineradora.findOne({ where: { usuário: cpf_encryptado }, 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_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: loginUsuário" }); }
  };

  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");
};
};

```

// 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_encryptado = md5(cpf);
    let senha_encryptada: string, resposta_encryptada: string;
    let token: string;
    const usuário = await Usuário.findOne(cpf_encryptado);
    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_encryptada = await bcrypt.hash(senha, SALT);
      usuário.senha = senha_encryptada;
    }
    if (resposta) {
      resposta_encryptada = await bcrypt.hash(resposta, SALT);
      usuário.questão = questão;
      usuário.resposta = resposta_encryptada;
    }
    await Usuário.save(usuario);
    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(usuario_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_encryptado = md5(request.params.cpf);
    const entityManager = getManager();
    await entityManager.transaction(async (transactionManager) => {
      const usuário = await transactionManager.findOne(Usuário, cpf_encryptado);
      await transactionManager.remove(usuario);
      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_encryptado = md5(request.params.cpf);
    const usuário = await Usuário.findOne(cpf_encryptado);
    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_encryptado = md5(cpf);
    const usuário = await Usuário.findOne(cpf_encryptado);
    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
};

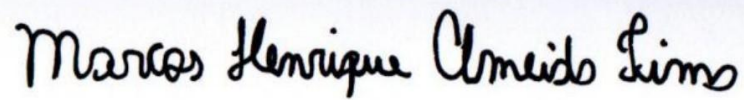
```

```
$$$ 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, 19/09/25

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