

Desenvolvimento de Aplicações Web - Entrega 2 - Abner Lucas Pereira Cardoso Vera

```
$$$ src.entidades.designer-gráfico
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne,
PrimaryGeneratedColumn }
  from "typeorm";
import Usuário from "../usuário";
import Candidatura from "../candidatura";
export enum Disponibilidade { FREELANCER = "freelancer", MEIOPERÍODO = "meio
período",
  TEMPOINTEGRAL = "tempo integral"
};
@Entity()
export default class DesignerGráfico extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ type: "enum", enum: Disponibilidade })
  disponibilidade: Disponibilidade;
  @Column()
  anos_experiência: number;
  @Column({ type: "date" })
  data_nascimento: Date;
  @Column()
  telefone: string;
  @OneToMany(() => Candidatura, (candidatura) => candidatura.designer_gráfico)
  candidaturas: Candidatura[];
  @OneToOne(() => Usuário, usuário => usuário.designer_gráfico, { onDelete:
"CASCADE" })
  @JoinColumn()
  usuário: Usuário;
}
```

```

$$$ src.entidades.candidatura
import { BaseEntity, Column, CreateDateColumn, Entity, ManyToOne,
PrimaryGeneratedColumn } from
"typeorm";
import DesignerGráfico from "../designer-gráfico";
import DesignLogo from "../design-logo";
@Entity()
export default class Candidatura extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  necessidade_contrato: boolean;
  @Column()
  justificativa: string;
  @CreateDateColumn()
  data_manifestação: Date;
  @ManyToOne(() => DesignLogo, (designLogo) => designLogo.candidaturas, { onDelete:
"CASCADE" })
  design_logo: DesignLogo;
  @ManyToOne(() => DesignerGráfico, (designerGráfico) => designerGráfico.candidaturas,
{ onDelete: "CASCADE" })
  designer_gráfico: DesignerGráfico;
}

```

```

$$$ src.entidades.empresário
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne,
PrimaryGeneratedColumn } from
"typeorm";
import Usuário from "../usuário";
import DesignLogo from "../design-logo";
export enum Classificação {PEQUENO = "pequeno", MÉDIO = "médio", GRANDE =
"grande"};
@Entity()
export default class Empresário extends BaseEntity {
@PrimaryGeneratedColumn()
id: number;
@Column({ type: "enum", enum: Classificação })
classificação: Classificação;
@Column()
anos_experiência_empresarial: number;
@OneToMany(() => DesignLogo, (designLogo) => designLogo.empresário)
designs_logos: DesignLogo[];
@OneToOne(() => Usuário, (usuário) => usuário.empresário, { onDelete: "CASCADE" })
@JoinColumn()
usuário: Usuário;
}

```

```

$$$ src.entidades.design-logo
import { BaseEntity, Column, Entity, ManyToOne, OneToMany, PrimaryGeneratedColumn }
from "typeorm";
import Empresário from "../empresário";
import Candidatura from "../candidatura";
export enum EstiloLogo {
    MINIMALISTA = "minimalista", TIPOGRÁFICO = "tipográfico",
    MASCOTE = "mascote", VINTAGE = "vintage", ABSTRATO = "abstrato",
    CORPORATIVO = "corporativo", FUTURISTA = "futurista", ORGÂNICO = "orgânico"
};
export enum StatusProjeto {
    RASCUNHO = "rascunho", EMANDAMENTO = "em andamento", REVISÃO = "revisão",
    APROVADO = "aprovado", CONCLUÍDO = "concluído", CANCELADO = "cancelado"};
@Entity()
export default class DesignLogo extends BaseEntity {
    @PrimaryGeneratedColumn()
    id: number;
    @Column()
    nome_empresa: string;
    @Column({ type: "enum", enum: EstiloLogo })
    estilo_logo: EstiloLogo;
    @Column()
    aplicação_prevista: string;
    @Column({ type: "date" })
    data_início: Date;
    @Column()
    descrição: string;
    @Column()
    concorrendo_contrato: boolean;
    @Column({ type: "enum", enum: StatusProjeto })
    status_projeto: StatusProjeto;
    @ManyToOne(() => Empresário, (empresário) => empresário.designs_logos, { onDelete:
"CASCADE" })
    empresário: Empresário;
    @OneToMany(() => Candidatura, (candidatura) => candidatura.design_logo)
    candidaturas: Candidatura[];
}

```

```

$$$ src.entidades.usuario
import { BaseEntity, Column, CreateDateColumn, Entity, OneToOne, PrimaryColumn } from
"typeorm";
import Empresário from "../empresário";
import DesignerGráfico from "../designer-gráfico";
export enum Perfil { DESIGNERGRÁFICO = "designer-gráfico", EMPRESÁRIO =
"empresário" };
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(() => Empresário, (empresário) => empresário.usuario)
empresário: Empresário;
@OneToOne(() => DesignerGráfico, (designer_gráfico) => designer_gráfico.usuario)
designer_gráfico: DesignerGráfico;
@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 } });
  if (usuário_token.email !== usuário.email) return response.status(401).json
    ("Acesso não autorizado.");
  next();
}
```

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

```
$$$ src.middlewares.verificar-perfil-empresário
import { Perfil } from "../entidades/usuário";
export default function verificarPerfilEmpresário(request, response, next) {
  if (request.perfil === Perfil.EMPRESÁRIO) 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.rotas-designer-gráfico
import { Router } from "express";
import verificarToken from "../middlewares/verificar-token";
import verificarPerfilDesignerGráfico from "../middlewares/verificar-perfil-designer-gráfico";
import ServiçosDesignerGráfico from "../serviços/serviços-designer-gráfico";
const RotasDesignerGráfico = Router();
export default RotasDesignerGráfico;
RotasDesignerGráfico.post("/", ServiçosDesignerGráfico.cadastrarDesignerGráfico);
RotasDesignerGráfico.patch("/", verificarToken, verificarPerfilDesignerGráfico,
ServiçosDesignerGráfico.atualizarDesignerGráfico);
RotasDesignerGráfico.get("/:cpf", verificarToken, verificarPerfilDesignerGráfico,
ServiçosDesignerGráfico.buscarDesignerGráfico);
```

```
$$$ src.rotas.rotas-empresário
import { Router } from "express";
import verificarToken from "../middlewares/verificar-token";
import verificarPerfilEmpresário from "../middlewares/verificar-perfil-empresário";
import ServiçosEmpresário from "../serviços/serviços-empresário";
const RotasEmpresário = Router();
export default RotasEmpresário;
RotasEmpresário.post("/", ServiçosEmpresário.cadastrarEmpresário);
RotasEmpresário.get("/:cpf", verificarToken, verificarPerfilEmpresário,
  ServiçosEmpresário.buscarEmpresário);
RotasEmpresário.patch("/", verificarToken, verificarPerfilEmpresário,
  ServiçosEmpresário.atualizarEmpresário);
```

```
$$$ 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.get("/questao/:cpf", ServiçosUsuário.buscarQuestãoSegurança);
RotasUsuário.post("/verificar-resposta", ServiçosUsuário.verificarRespostaCorreta);
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);
```

```

$$$ src.serviços.serviços-designer-gráfico
import md5 from "md5";
import { getManager } from "typeorm";
import Usuário, { Status } from "../entidades/usuário";
import DesignerGráfico from '../entidades/designer-gráfico';
import ServiçosUsuário from "../serviços-usuário";
export default class ServiçosDesignerGráfico {
  constructor() {}
  static async cadastrarDesignerGráfico(request, response) {
    try {
      const { usuário_info, disponibilidade, anos_experiência, 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 designerGráfico = DesignerGráfico.create({ usuário, disponibilidade,
          anos_experiência, data_nascimento, telefone });
        await transactionManager.save(designerGráfico);
        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 atualizarDesignerGráfico(request, response) {
    try {
      const { cpf, disponibilidade, anos_experiência, data_nascimento, telefone } = request.body;
      const cpf_encryptado = md5(cpf);
      await DesignerGráfico.update({ usuário: { cpf: cpf_encryptado } }, { disponibilidade,
        anos_experiência,
        data_nascimento, telefone });
      return response.json();
    } catch (error) { return response.status(500).json({ erro: "Erro BD :
        atualizarDesignerGráfico" }); }
  };
  static async buscarDesignerGráfico(request, response) {
    try {
      const cpf_encryptado = md5(request.params.cpf);
      const designerGráfico = await DesignerGráfico.findOne({ where: { usuário: cpf_encryptado },
        relations: ["usuário"] });
      if (!designerGráfico) return response.status(404).json({ erro: "DesignerGráfico não
        encontrado." });
      return response.json({ nome: designerGráfico.usuário.nome, email:
        designerGráfico.usuário.email,
        disponibilidade: designerGráfico.disponibilidade, anos_experiência:
        designerGráfico.anos_experiência,
        data_nascimento: designerGráfico.data_nascimento, telefone: designerGráfico.telefone });
    }
  }
}

```

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

```

$$$ src.serviços.serviços-empresário
import md5 from "md5";
import { getManager } from "typeorm";
import Usuário, { Status } from "../entidades/usuário";
import Empresário from "../entidades/empresário";
import ServiçosUsuário from "../serviços-usuário";
export default class ServiçosEmpresário {
  constructor() {}
  static async cadastrarEmpresário(request, response) {
    try {
      const { usuário_info, classificação, 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 empresário = Empresário.create({ usuário, classificação,
          anos_experiência_empresarial });
        await transactionManager.save(empresário);
        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 buscarEmpresário(request, response) {
    try {
      const cpf_encriptado = md5(request.params.cpf);
      const empresário = await Empresário.findOne({ where: { usuário: cpf_encriptado },
        relations: ["usuário"] });
      if (!empresário) return response.status(404).json({ erro: "Empresário não encontrado." });
      return response.json({ nome: empresário.usuário.nome, email: empresário.usuário.email,
        classificação: empresário.classificação,
        anos_experiência_empresarial: empresário.anos_experiência_empresarial });
    } catch (error) { return response.status(500).json({ erro: "Erro BD : buscarEmpresário" }); }
  };
  static async atualizarEmpresário(request, response) {
    try {
      const { cpf, classificação, anos_experiência_empresarial } = request.body;
      const cpf_encriptado = md5(cpf);
      await Empresário.update({ usuário: { cpf: cpf_encriptado } },
        { classificação, anos_experiência_empresarial });
      return response.json();
    } catch (error) { return response.status(500).json({ erro: "Erro BD : atualizarEmpresário" }); }
  };
};

```

```

$$$ src.serviços.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 Empresário from "../entidades/empresário";
import DesignerGráfico from "../entidades/designer-gráfico";
import { getManager } from "typeorm";
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.EMPRESÁRIO:
        const empresário = await Empresário.findOne({ where: { usuário: usuário.cpf },
          relations: ["usuário"] });
        if (!empresário) return false;
        return true;
      case Perfil.DESIGNERGRÁFICO:
        const designerGráfico = await DesignerGráfico.findOne({ where: { usuário: usuário.cpf },
          relations: ["usuário"] });
        if (!designerGráfico) 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(usuario_informado) {
  try {
    const { cpf, nome, perfil, email, senha, questão, resposta, cor_tema } = usuario_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");
  }
};

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(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" }); }
};

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

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" }); }
};

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" });
}
};
};

```

```
$$$ src.servidor
import cors from "cors";
import express from "express";
import "reflect-metadata";
import { createConnection } from "typeorm";
import RotasUsuário from "../rotas/rotas-usuário";
import RotasEmpresário from "../rotas/rotas-empresário";
import RotasDesignerGráfico from "../rotas/rotas-designer-gráfico";

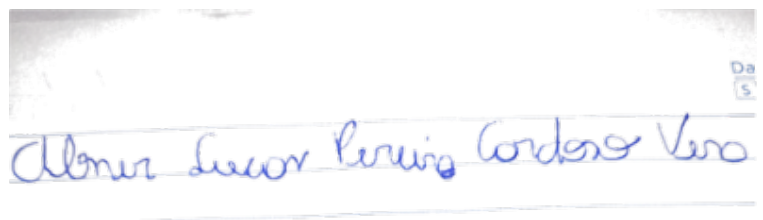
const app = express();
const PORT = process.env.PORT
const CORS_ORIGIN = process.env.CORS_ORIGIN;

app.use(cors({ origin: CORS_ORIGIN }));
app.use(express.json());
app.use("/designers-grafico", RotasDesignerGráfico);
app.use("/usuarios", RotasUsuário);
app.use("/empresarios", RotasEmpresário);
app.listen(PORT || 3333);
const conexão = createConnection();
export default conexão;

console.log(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, 22/09/2025 -- assinatura



Almir Luiz Pereira Cordes Viro