Linguagem de Programação III - Entrega 1 - Gabriel Michel Braucks

diretório.arquivo : src.entidades.interesse

```
import { BaseEntity, Column, CreateDateColumn, Entity, ManyToOne, PrimaryGeneratedColumn }
from
    "typeorm";

import Locatário from "./locatário";
import Residência from "./residência";

@Entity()
export default class Interesse extends BaseEntity {
    @PrimaryGeneratedColumn()
    id: number;
    @Column()
    valor_proposto: boolean;
    @Column()
    justificativa: string;
    @CreateDateColumn()
    data_manifestação: Date;
    @ManyToOne(() => Residência, (residência) => residência.interesses, { onDelete: "CASCADE" })
    residência: Residência;
    @ManyToOne(() => Locatário, (locatário) => locatário.interesses, { onDelete: "CASCADE" })
    locatário: Locatário;
}
```

diretório.arquivo : src.entidades.locador

```
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne, PrimaryGeneratedColumn
} from
    "typeorm";

import Usuário from "./usuário";
import Residência from "./residência";

@Entity()
export default class Locador extends BaseEntity {
    @PrimaryGeneratedColumn()
    id: number;
    @Column()
    anos_experiência: number;
    @Column()
    número_imóveis: number;
    @OneToMany(() => Residência, (residência) => residência.locador)
    residências: Residência[];
    @OneToOne(() => Usuário, (usuário) => usuário.locador, { onDelete: "CASCADE" })
    @JoinColumn()
    usuário: Usuário;
}
```

diretório.arquivo : src.entidades.locatário

```
import { BaseEntity, Column, Entity, JoinColumn, OneToMany, OneToOne, PrimaryGeneratedColumn
}
   from "typeorm";

import Usuário from "./usuário";
import Interesse from "./interesse";

@Entity()
export default class Locatário extends BaseEntity {
   @PrimaryGeneratedColumn()
   id: number;
   @Column()
   renda_mensal: number;
   @Column()
   telefone: string;
   @OneToMany(() => Interesse, (interesse) => interesse.locatário)
   interesses: Interesse[];
   @OneToOne(() => Usuário, usuário => usuário.locatário, { onDelete: "CASCADE" })
   @JoinColumn()
   usuário: Usuário;
}
```

diretório.arquivo : src.entidades.residência

```
import { BaseEntity, Column, Entity, ManyToOne, OneToMany, PrimaryGeneratedColumn } from
"typeorm";

import Locador from "./locador";
import Interesse from "./interesse";

export enum Categoria { APARTAMENTO = "Apartamento", CASA = "Casa", KITNET = "Kitnet",
LOFT = "Loft" };

@Entity()
export default class Proposta extends BaseEntity {
    @PrimaryGeneratedColumn()
    id: number;
    @Column()
    título: string;
    @Column({ type: "enum", enum: Categoria })
    categoria: Categoria;
    @Column()
    localização: string;
    @Column()
    valor_aluguel: number;
    @Column({ type: "date" })
    data_disponibilidade: Date;
    @Column()
    descrição: string;
    @Column()
    mobiliado: boolean;
    @ManyToOne(() => Locador, (locador) => locador.residências, { onDelete: "CASCADE" })
    locador: Locador;
    @OneToMany(() => Interesse, (interesse) => interesse.residência)
    interesses: Interesse[];
}
```

diretório.arquivo : src.entidades.usuário

```
import { BaseEntity, Column, CreateDateColumn, Entity, OneToOne, PrimaryColumn } from
"typeorm";

import Locador from "./locador";
import Locatário from "./locatário";

export enum Perfil { LOCATÁRIO = "locatário", LOCADOR = "locador" };
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(() => Locador, (locador) => locador.usuário)
   locador: Locador;
   @OneToOne(() => Locatário, (locatário) => locatário.usuário)
   locatário: Locatário;
   @CreateDateColumn()
```

```
    data_criação: Date;
}
```

diretório.arquivo : src.middlewares.verificar-perfil-locador

```
import { Perfil } from "../entidades/usuário";

export default function verificarPerfilLocador(request, response, next) {
    if (request.perfil === Perfil.LOCADOR) return next();
    else return response.status(401).json({ erro: "Acesso não autorizado." });
};
```

diretório.arquivo : 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." });
    }
};
```

diretório.arquivo : src.rotas.rotas-locador

```
import { Router } from "express";
import verificarToken from "../middlewares/verificar-token";
import verificarPerfilLocador from "../middlewares/verificar-perfil-locador";
import ServiçosLocador from "../serviços/serviços-locador";

const RotasProfessor = Router();

export default RotasProfessor;

RotasProfessor.post("/", ServiçosLocador.cadastrarLocador);
RotasProfessor.get("/:cpf", verificarToken, verificarPerfilLocador, ServiçosLocador.buscarLocador);
```

diretório.arquivo : src.rotas.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);
```

diretório.arquivo : src.serviços.serviços-locador

```
import md5 from "md5";
import { getManager } from "typeorm";
import Usuário, { Status } from "../entidades/usuário";

import Locador from "../entidades/locador";
import ServiçosUsuário from "./serviços-usuário";

export default class ServiçosLocador {
   constructor() { }
   static async cadastrarLocador(request, response) {
      try {
         const { usuário_info, anos_experiência } = request.body;
         const número_imóveis = 0;
         const { usuário, token } = await ServiçosUsuário.cadastrarUsuário(usuário_info);
         const entityManager = getManager();
         await entityManager.transaction(async (transactionManager) => {
            await transactionManager.save(usuário);
            const locador = Locador.create({ usuário, anos_experiência, número_imóveis });
            await transactionManager.save(locador);
            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 buscarLocador(request, response) {
      try {
         const cpf_encriptado = md5(request.params.cpf);
         const locador = await Locador.findOne({
            where: { usuário: cpf_encriptado },
            relations: ["usuário"]
         });
         if (!locador) return response.status(404).json({ erro: "Professor não encontrado." });
         return response.json({
            nome: locador.usuário.nome, email: locador.usuário.email,
            anos_experiência: locador.anos_experiência,
            número_imóveis: locador.número_imóveis
         });
      } catch (error) { return response.status(500).json({ erro: "Erro BD : buscarProfessor" }); }
   };
```

};

diretório.arquivo : 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 Locador from "../entidades/locador";
import Locatário from "../entidades/locatário";

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.LOCADOR:
            const locador = await Locador.findOne({
               where: { usuário: usuário.cpf },
               relations: ["usuário"]
            });
            if (!locador) return false;
            return true;
         case Perfil.LOCATÁRIO:
            const locatário = await Locatário.findOne({
               where: { usuário: usuário.cpf },
               relations: ["usuário"]
```

```
            });
            if (!locatário) 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");
        };
    };
};
```

diretório.arquivo : src.servidor

```
import cors from "cors";
import express from "express";
import "reflect-metadata";
import { createConnection } from "typeorm";
import RotasUsuário from "./rotas/rotas-usuário";
import RotasLocador from "./rotas/rotas-locador";

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("/locadores", RotasLocador);
app.listen(PORT || 3333);

const conexão = createConnection();

export default conexão;
```

diretório.arquivo : .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 MS, 18/03/2025 —

gov.br
Documento assinado digitalmente
GABRIEL MICHEL BRAUCKS
Data: 18/03/2025 12:49:25-0300
Verifique em https://validar.iti.gov.br