

## Linguagem de Programação III - Entrega 2 - 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";

export enum RendaMensal { ATE2 = "até_2_salarios", DE2A5 = "de_2_a_5_salarios",
ACIMADE5 = "acima_5_salarios" };

@Entity()
export default class Locatário extends BaseEntity {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ type: "enum", enum: RendaMensal })
  renda_mensal: RendaMensal;
  @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 Residência 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.usuario**

```
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.usuario)  
  locador: Locador;  
  @OneToOne(() => Locatário, (locatário) => locatário.usuario)  
  locatário: Locatário;  
  @CreateDateColumn()  
  data_criação: Date;  
}
```

**diretório.arquivo: 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();
}
```

**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-perfil-locatário**

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

export default function verificarPerfilLocatário(request, response, next) {
  if (request.perfil === Perfil.LOCATÁRIO) 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 RotasLocador = Router();

export default RotasLocador;

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

## **diretório.arquivo: src.rotas.rotas-locatário**

```
import { Router } from "express";
import verificarToken from "../middlewares/verificar-token";
import verificarPerfilLocatário from "../middlewares/verificar-perfil-locatário";
import ServiçosLocatário from "../serviços/serviços-locatário";

const RotasLocatário = Router();

export default RotasLocatário;

RotasLocatário.post("/", ServiçosLocatário.cadastrarLocatário);
RotasLocatário.patch("/", verificarToken, verificarPerfilLocatário,
ServiçosLocatário.atualizarLocatário);
RotasLocatário.get("/:cpf", verificarToken, verificarPerfilLocatário,
ServiçosLocatário.buscarLocatário);
```

## **diretório.arquivo: 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);
```

## **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(usuario_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: "Locador não
encontrado." });
      return response.json({
        nome: locador.usuario.nome, email: locador.usuario.email,
        anos_experiência: locador.anos_experiência,
        número_imóveis: locador.número_imóveis
      });
    } catch (error) { return response.status(500).json({ erro: "Erro BD :
buscarLocador" }); }
  };
}
```

```
static async atualizarLocador(request, response) {
  try {
    const { cpf, anos_experiência } = request.body;
    const cpf_encryptado = md5(cpf);
    await Locador.update({ usuário: { cpf: cpf_encryptado },
      { anos_experiência } });
    return response.json();
  } catch (error) { return response.status(500).json({ erro: "Erro BD :
atualizarLocador" }); }
};
};
```

## **diretório.arquivo: src.serviços.serviços-locatário**

```
import md5 from "md5";
import { getManager } from "typeorm";
import Usuário, { Status } from "../entidades/usuário";
import Locatário from "../entidades/locatário";
import ServiçosUsuário from "../serviços-usuário";

export default class ServiçosLocatário {
  constructor() {}
  static async cadastrarLocatário(request, response) {
    try {
      const { usuário_info, renda_mensal, telefone } = request.body;
      const { usuário, token } = await
ServiçosUsuário.cadastrarUsuário(usuário_info);
      const entityManager = getManager();
      await entityManager.transaction(async (transactionManager) => {
        await transactionManager.save(usuário);
        const locatário = Locatário.create({ usuário, renda_mensal, telefone });
        await transactionManager.save(locatário);
        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 atualizarLocatário(request, response) {
    try {
      const { cpf, renda_mensal, telefone } = request.body;
      const cpf_encryptado = md5(cpf);
      await Locatário.update({ usuário: { cpf: cpf_encryptado } }, {
        renda_mensal, telefone
      });
      return response.json();
    } catch (error) { return response.status(500).json({ erro: "Erro BD :
atualizarLocatário" }); }
  };
  static async buscarLocatário(request, response) {
    try {
      const cpf_encryptado = md5(request.params.cpf);
      const locatário = await Locatário.findOne({
        where: { usuário: cpf_encryptado },
        relations: ["usuário"]
      });
      if (!locatário) return response.status(404).json({ erro: "Locatário não
encontrado." });
      return response.json({
        nome: locatário.usuário.nome, email: locatário.usuário.email,
        renda_mensal: locatário.renda_mensal, telefone: locatário.telefone
      });
    }
  }
}
```

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



## **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/usuario";
import Locador from "../entidades/locador";
import Locatário from "../entidades/locatário";

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

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" }); }
};

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" });
    }
};

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" });
    }
};
};
};

```

## **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";
import RotasLocatário from "../rotas/rotas-locatário";

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.use("/locatarios", RotasLocatário);
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
```