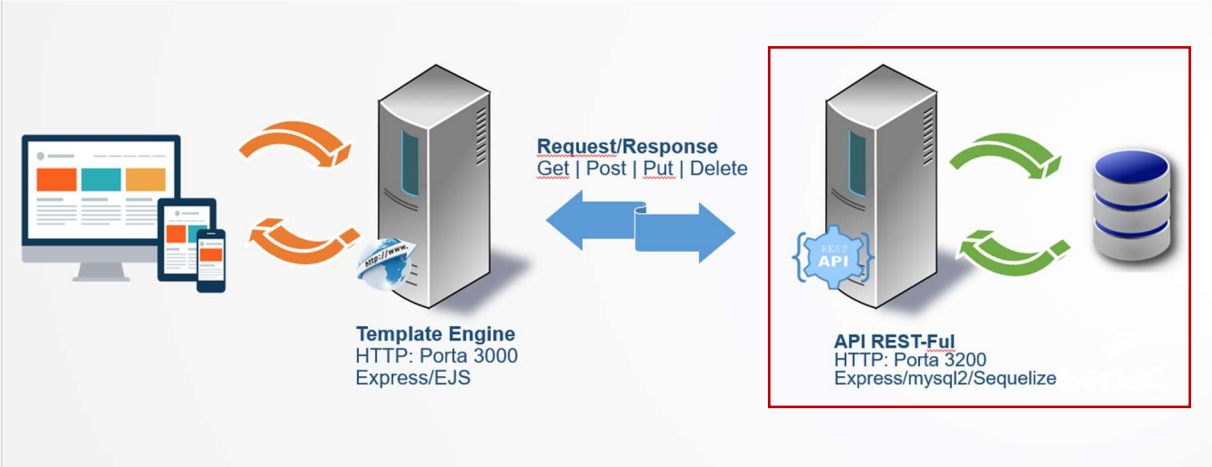
CAPÍTULO 6 – Manipulação de dados utilizando ORM

A proposta para esta aplicação é desenvolver dois servidores, conforme a imagem a seguir:



Chegou a hora de desenvolver uma versão funcional da API-Rest que fornecerá serviços para a aplicação web.

Neste capítulo você vai ver:

- Manipular tabelas do banco de dados utilizando models do ORM Sequelize;
- Criar relacionamento entre tabelas(models);
- Configuração de CORS;
- Utilização de Middleware;
- Organização do código em módulos no diretório “controllers”;
- Utilização do módulo “consign” para organizar o código.

Atividade 1 – Introdução e instalação do ORM - Sequelize

A sigla ORM significa **Object Relational Mapping**, tem como função principal aproximar o paradigma de orientação a objetos ao paradigma de **bancos de dados relacionais** (MySQL, PostigreSQL, SQL Server, Oracle e outros). Basicamente mapeia as entidades (tabelas) através de models(classes), a fim de abstrair complexidades do SGBD, tais como instruções SQL e relacionamentos entre tabelas.

Essa atividade tem como objetivo:

- Instalar o módulo Sequelize;

- Criar o módulo de conexão com o banco;

- Criar um model;

- Utilizar os principais métodos de manipulação de dados.

Comandos:

- **npm init** -y | **npm install** express mysql2 sequelize

Exemplo de aplicação CRUD utilizando Express e Sequelize

Vamos ver como criar uma conexão com o banco de dados, um model e utilizar os principais métodos do Sequelize.

1. Para iniciar a atividade, crie um diretório **Cap06**, dentro dele crie um subdiretório com o nome "**at01-Sequelize**" e abra no VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:
   - ✓ npm init -y
   - ✓ npm install express mysql2 sequelize

   Insira também o **"start"**: "nodemon index.js" (arquivo pakete.json vide exemplos anteriores)
3. Crie no MySql um novo banco(Schema) com o título de "teste".
4. Crie um módulo chamado "bdConexao.js" e insira o seguinte código:

```
//Desestruturação da classe Sequelize
const {Sequelize} = require('sequelize')
//Instancia de um objeto Sequelize
const sequelize = new Sequelize(
    //Lista de parâmetros: banco, usuário e senha
    'teste', 'nodejs', '1234', {
        host:'localhost', //nome DNS ou IP do servidor
        dialect:'mysql', //SGBD utilizado
        charset: 'utf8', //tabela de caracteres
        collate: 'utf8_general_ci', //colação
        timezone:"-03:00" //fuso horário
    })
try {
    //metodo de autenticação que conecta ao banco
    sequelize.authenticate()
    console.log('Conectado ao banco')
} catch (erro) {
    console.log('Não foi possível conectar: ', erro )
}
module.exports = sequelize
```

5. Vamos criar o model para representar a entidade(tabela) usuário. Para isso crie um subdiretório no projeto chamado “models” que irá abrigar os models. Crie dentro de “models” um arquivo chamado “usuario.js” e insira o seguinte código:

```
//Desestruturação das classes DataTypes e Model do módulo sequelize
const { DataTypes, Model } = require('sequelize')

//Importação do módulo de conexão
const sequelize = require('../bdConexao')

//Classe Usuario herdando a classe Model
class Usuario extends Model{}
Usuario.init({
    //Atributos da entidade usuário
    nome: {
        type: DataTypes.STRING,
        allowNull: false
      },
      cpf: {
        type: DataTypes.STRING,
        allowNull:false
      }
},{
    sequelize,
    modelName:'usuario'
  }
)

sequelize.sync() //Cria a tabela caso não exista
 module.exports = Usuario

 /*
   Consulte todos os tipos de dados da classe dataTypes em:
   https://sequelize.org/docs/v6/core-concepts/model-basics/#data-types
  */
```

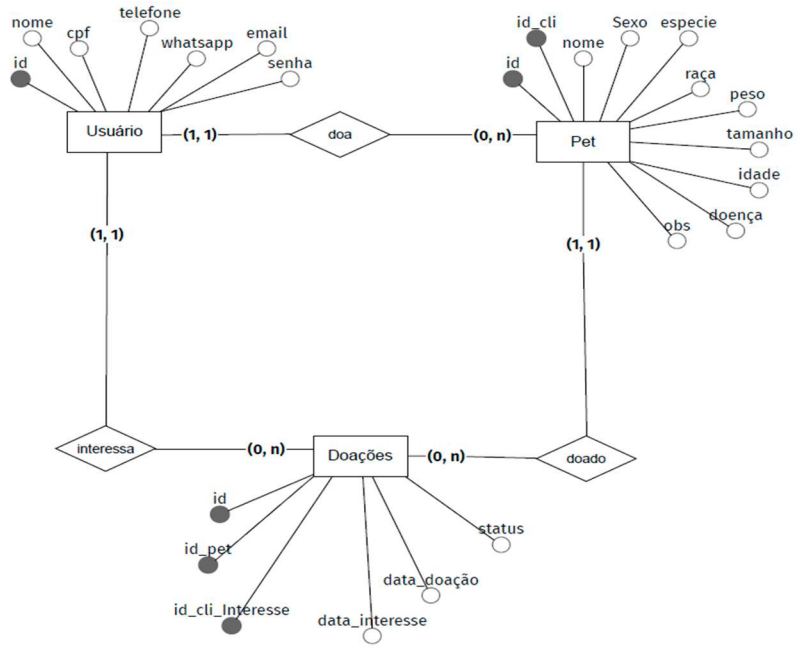
6. Saia do diretório “models” e no diretório raiz do projeto crie o módulo index.js e insira o seguinte código:

```
const express = require('express')
const app = express()

//Instância do objeto (Model)
const usuario = new require('./models/usuario')
var porta = '3200'

app.use(express.urlencoded({extended:false}))
app.use(express.json())

app.get('/', (req, res)=>res.send('API - Amigo do Pet'))

app.get('/consultar/usuarios/:id?', async (req, res)=>{
    //O método finOne busca um registro baseado em uma condição, por
exemplo o id
    // O método findAll obtém todos os registros da entidade
    let dados = req.params.id? await
usuario.findOne({where:{id:req.params.id}}) :
    await usuario.findAll()
    res.json(dados)
})
app.post('/cadastrar/usuarios', async (req, res)=>{
    let dados = req.body
    //Método create insele novo registro
    let respBd = await usuario.create(dados)
    res.json(respBd)
})
app.put('/atualizar/usuarios/:id', async (req, res) => {
    let id = req.params.id
    let dados = req.body
    //Método update atualiza o registro baseado no id
    let respBd = await usuario.update(dados, {where:{id:id}})
    res.json(respBd)
})
app.delete('/excluir/usuarios/:id', async (req, res) => {
    let id = req.params.id
    //Método destry exclui um registro baseado no id
    let respBd = await usuario.destroy({where:{id:id}})
    res.json(respBd)
})
app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

7. Rode o código com o comando **npm start**.
8. Execute o procedimento de testes nas rotas GET, POST, PUT e DELETE utilizando o Postmann conforme atividade 4 do capítulo 5.

Atividade 2 – Desenvolvimento de API – REST

Para esta atividade vamos considerar as entidades – Usuário – Pet – Doações os atributos e relacionamentos definidos no MER a seguir:



Essa atividade tem como objetivo:

- Criar models com definição de relacionamentos entre as entidades;

- Estruturar o código da aplicação em módulos no diretório controllers.

Comandos:

- **npm init** -y | **npm install** express mysql2 sequelize cors

Desenvolvimento da versão 1.0 da API do sistema "Amigo do Pet".

Nesta aplicação vamos definir as tabelas, campos e relacionamentos do banco de dados através de models do Sequelize. E estruturar o código da aplicação em módulos, utilizando a biblioteca "consign" e o diretório controllers.

1. Para iniciar a atividade, crie no diretório **Cap06** um subdiretório com o nome "**at02-APIRest_Amigo_do_Pet-V_1_0**" e abra no VSCode;
2. Realize o procedimento para inicializar o projeto e instalar os módulos necessários com os comandos no console:
   - ✓ npm init -y
   - ✓ npm install express mysql2 sequelize cors

Insira também o **"start"**: "nodemon index.js" (arquivo pakete.json vide exemplos anteriores)

3. Crie no MySql um novo banco(**Schema**) com o título de "**amigo_do_pet**" - Caso o banco de dados dos capítulos anteriores ainda existir, deve ser apagado e criado novamente.
4. Crie um módulo chamado "**bdConexao.js**" e insira o seguinte código:

```
const {Sequelize} = require('sequelize');
const sequelize = new Sequelize(
    'amigo_do_pet', 'nodejs', '1234', {
        host:'localhost',
        dialect:'mysql',
        charset: 'utf8',
        collate: 'utf8_general_ci',
        timezone:"-03:00"
    })
try {
    sequelize.authenticate()
    console.log('Conectado ao banco')
} catch (erro) {
    console.log('Não foi possível conectar: ', erro )
}
module.exports = sequelize
```

5. Crie no diretório raiz do projeto o módulo index.js e insira o seguinte código:

```
const express = require('express')
//Módulo consign facilita a organização de módulos e faz carga automático
de scripts
const consign=require('consign')
const app = express()
const cors = require('cors')
var porta = '3200'
app.use(cors()) //Configura política de segurança CORS
app.use(express.urlencoded({extended:false}))
app.use(express.json())
app.get('/', (req, res)=>res.send('API - Amigo do Pet'))

consign() //Execução do consign
    .include('./controllers/rotas') //Inclui  módulos contidos no diretório
especificado
    .into(app) //passa a instância do Express o para os módulos

app.listen(porta, ()=>console.log(`Servidor rodando em:
http://localhost:${porta}`))
```

6. Crie um subdiretório no projeto chamado "models" para os models. Crie dentro de "**models**" um arquivo chamado - "**usuario.js**" e insira o seguinte código:

```
const { DataTypes, Model } = require('sequelize')
const sequelize = require('../bdConexao')
class Usuario extends Model{}
Usuario.init({
    nome: {
        type: DataTypes.STRING(50),
        allowNull: false
      },
      cpf: {
        type: DataTypes.STRING(14),
        allowNull:false
      },
      telefone: {
        type: DataTypes.STRING(14),
        allowNull:true
      },
      whatsapp: {
        type: DataTypes.STRING(14),
        allowNull:true
      },
      email: {
        type: DataTypes.STRING(50),
        allowNull:false
      },
      senha: {
        type: DataTypes.STRING,
        allowNull:false
      }
},{
    sequelize,
    modelName:'usuario'
  }
)
sequelize.sync()
 module.exports = Usuario
```

7. Crie dentro de “**models**” outro arquivo chamado - “**pet.js**” e insira o seguinte código:

```
const { DataTypes, Model } = require('sequelize')
const sequelize = require('../bdConexao')
const usuario = new require('./usuario')
class Pet extends Model{}
Pet.init({
    nome: {
        type: DataTypes.STRING(50),
        allowNull: false
      },
      sexo: {
        type: DataTypes.STRING(1),
        allowNull:false
      },
      especie: {
        type: DataTypes.STRING(50),
        allowNull:false
      },
      raca: {
        type: DataTypes.STRING(50),
        allowNull:true
      },
      peso: {
        type: DataTypes.INTEGER,
        allowNull:true
      },
      tamanho: {
        type: DataTypes.STRING(10),
        allowNull:false
      },
      idade: {
        type: DataTypes.INTEGER,
        allowNull:true
      },
      doenca: {
        type: DataTypes.STRING,
        allowNull:true
      },
      obs: {
        type: DataTypes.STRING,
        allowNull:true
      }
},{
    sequelize,
    modelName:'pet'
  })
usuario.hasMany(Pet) // Usuario tem muitos Pets 1-p-M
Pet.belongsTo(usuario) //Pet pertence a Usuário 1-p-1
sequelize.sync()
 module.exports = Pet
```

8. Crie dentro de “**models**” um arquivo chamado - “**doacao.js**” e insira o seguinte código:

```
const { Sequelize, DataTypes, Model } = require('sequelize')
const sequelize = require('../bdConexao')
const usuario = new require('./usuario')
const pet = new require('./pet')
class Doacao extends Model{}
Doacao.init({
    data_interesse: {
        type:Sequelize.DATEONLY,
        defaultValue: DataTypes.NOW
      },
      data_doacao:{
        type:Sequelize.DATEONLY,
        allowNull: true
      },
      status: {
        type: DataTypes.STRING,
        defaultValue:"Cadastrada"
      }
    },{
      sequelize,
      modelName:'doacoes'
    }
  )
  pet.hasMany(Doacao) //Muitos Pets têm muitas Doações - M-p-M
  Doacao.belongsTo(pet)

  usuario.hasMany(Doacao) //Muitos Usuários têm muitas Doações - M-p-M
  Doacao.belongsTo(usuario)

  sequelize.sync()
   module.exports = Doacao
```

9. Crie um subdiretório no projeto chamado “**controllers”** e dentro dele outro subdiretório chamado “**rotas**”, conforme definido no método include do consign.
10. Dentro do subdiretório “rotas” vamos criar um módulo para implementarmos as rotas de cada entidade. Vamos começar criando em “rotas” um arquivo “**usuario.js**” e inserindo o seguinte código:

```
const model = new require('../../models/usuario')
const rota = 'usuarios'
module.exports = (app)=>{
    app.get(`/${rota}/:id?`, async (req, res)=>{
        try {
            let dados=req.params.id?
                  await model.findOne({where:{id:req.params.id}}):
                  await model.findAll()
            res.json(dados).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.post(`/${rota}`, async (req, res)=>{
        try {
            let dados = req.body
            const salt = bcrypt.genSaltSync()
            dados.senha = bcrypt.hashSync(dados.senha, salt)
            let respBd = await model.create(dados)
            res.json(respBd).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.put(`/${rota}/:id`, async (req, res) => {
        try {
            let id = req.params.id
            let dados = req.body
            console.log(dados)
            let respBd = await model.update(dados, {where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.delete(`/${rota}/:id`, async (req, res) => {
        try {
            let id = req.params.id
            let respBd = await model.destroy({where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
}
```

11. Agora crie em “rotas” um módulo chamado “**pet.js**” e insira o seguinte código:

```
const model = new require('../../models/pet')
const usuario = new require('../../models/usuario')
const rota = 'pets'
module.exports = (app)=>{
    app.get(`/${rota}/:id?`, async (req, res)=>{
        try {
            let dados = req.params.id?
                await model.findOne({include:[{model:usuario}]},
{where:{id:req.params.id}}) :
                await model.findAll({include:[{model:usuario}]}, {raw:
true, order:[['id','DESC']]})
            res.json(dados).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.post(`/${rota}`, async (req, res)=>{
        try {
            let dados = req.body
            console.log(dados)
            let respBd = await model.create(dados)
            res.json(respBd).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.put(`/${rota}/:id`, async (req, res) => {
        try {
            let id = req.params.id
            let dados = req.body
            console.log(dados)
            let respBd = await model.update(dados, {where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.delete(`/${rota}/:id`, async (req, res) => {
        try {
            let id = req.params.id
            let respBd = await model.destroy({where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
}
```

12. Para finalizar, crie em “rotas” um módulo chamado “**doacao.js**” e insira o seguinte código:

```
const model = new require('../../models/doacao')
const usuario = new require('../../models/usuario')
const pet = new require('../../models/pet')
const rota = 'doacoes'
module.exports = (app)=>{
    app.get(`/${rota}/:id?`, async (req, res)=>{
        try {
            let dados = req.params.id?
                await model.findOne({include:[{model:usuario},
{model:pet}]}, {where:{id:req.params.id}}) :
                await model.findAll({include:[{model:usuario},
{model:pet}]}, {raw: true, order:[['id','DESC']]})
            res.json(dados).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.post(`/${rota}`, async (req, res)=>{
        try {
            let dados = req.body
            let respBd = await model.create(dados)
            res.json(respBd).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.put(`/${rota}/:id`, async (req, res) => {
        try {
            let id = req.params.id
            let dados = req.body
            let respBd = await model.update(dados, {where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
    app.delete(`/${rota}/:id`, async (req, res) => {
        try {
            let id = req.params.id
            let respBd = await model.destroy({where:{id:id}})
            res.json(respBd)
        } catch (error) {
            res.json(error).status(400)
        }
    })
}
```

13. Rode o código com o comando **npm start** e execute os testes de funcionalidades da API utilizando o Postman.

Atividade 3 – Autenticação e criptografia

Para realizar o processo de autenticação da API utilizaremos a biblioteca JWT(JSON Web Token) que armazenar de forma segura e compacta objetos JSON que armazena informações de autenticação usuário em um token criptografado utilizando o padrão Base64.

A senha do usuário é um dado sensível e não pode ser gravado em texto puro no banco de dados pois ficariam expostas. Desta forma, é necessário gerar um hash(senha criptografada) que será gravada no banco de dados. Durante o processo de login a senha digitada também transformada em hash e comparada com o hash gravado no banco.

Essa atividade tem como objetivo:

- Realizar o processo de autenticação na API;
- Gerar um token para autenticação utilizando “JWT”;
- Utilizar senhas criptografadas utilizando a biblioteca “bcrypt”;
- Criar middleware de autenticação em rotas protegidas.

Desenvolvimento da versão 1.1 da API do sistema “Amigo do Pet”.

Nesta aplicação vamos criar um middkeware de autenticação e vincular às rotas protegidas, bem como desenvolver funções para gravar hash de senha e realizar o processo de login.

1. Para iniciar a atividade, duplique o diretório “**at02-APIRest_Amigo_do_Pet-V_1_0**”, renomeie para “**at03-APIRest_Amigo_do_Pet-V_1_1**” e abra no VSCode;

2. Emita o comando de instalação das novas dependências JWT e bcrypt:
   npm install bcrypt jsonwebtoken

3. No subdiretório “controllers” crie um módulo chamado “auth.js” e insira o seguinte código:

```
const jwt = require('jsonwebtoken')
const bcrypt = require('bcrypt')
const jwtSecret ='errtyytuczvxvdgghfrytddvfgfvzcvfehrew'
const model = new require('../../models/usuario')
module.exports={
    criptografarSenha: async(senha)=>{
        const salt = bcrypt.genSaltSync(12)
        return bcrypt.hashSync(senha, salt)
    },
    //https://www.base64decode.org/
    gerarToken: async(usuario)=> await jwt.sign(usuario, jwtSecret,
{expiresIn:'1h'}),

    //Compara o hash da senha enviada na requisição com o hash do banco
    validarSenha: async(senha, hashSenha)=> await bcrypt.compare(senha,
hashSenha),

    validarToken:(req, res, next)=>{
        try {
            let token = req.headers.authorization
            token = token.split(' ');
            token = token[1]
            jwt.verify(token, jwtSecret, (erro, dados)=>{//Verifica a
validade do token
            if (erro){
                res.json({message:'Token inválido!', error:erro
}).status(400)
            } else {
                req.token = token //Insere o token na requisição
                req.usuarioAtual = {...dados} //Insere os dados do usuario
atual na requisição
                next() //CallBack que executa a proxima função(no caso a
rota protegida)
            }
        })
        } catch (erro) {
            res.json({message:'Não existe token na
requisição.'}).status(400)
        }
    }
}
```

4. Crie outro módulo em “controllers” com o nome “validacao.js” com o seguinte código:

```
const auth = require('./auth')
module.exports = {
    validarCadastro: async(dados, model)=>{
        let usuario = await model.findOne({ where: { email: dados.email } 
})
        if (usuario!=null){//Verifica se o email já está cadastrado
            return {erro:'email inválido', message:"Email já cadastrado!"}
        }
        if (dados.senha != dados.confirmacao){ //verifica se a senha e a 
confirmação conferem
            return {erro:'senha', message:"Senhas não coincidem!"}
        }
        return {validacao:true} //retorna o status de validação verdadeiro
    },
    validarLogin: async (dados, model)=>{
        //Carrega os dados do banco
        let usuario = await model.findOne({ where: { email:dados.email }})
        if ( usuario==null){ //verifica se o email enviado pela requisição 
existe
            return {message:'Conta de email inválida.', autenticado:false}
        } else {
            //verifica se o hash da senha enviada confere com o hash do 
banco
            let authSenha = await auth.validarSenha(dados.senha, 
usuario.senha)
            //usuario = authSenha? {...usuario} : {message:'Senha 
inválida'}
            return authSenha? {usuario, 
autenticado:true}:{erro:{message:'Senha inválida'}, autenticado:false}
        }

    }
}
```

5. No diretório “rotas” crie um módulo “login.js” e insira o seguinte código:

```
const model = new require('../../models/usuario')
const auth = require('../auth')
const validacao = require('../validacao')

module.exports =(app)=>{
    app.post(`/login`, async (req, res)=>{
        try {
         let dados = req.body
         let validaLogin = await validacao.validarLogin(dados, model)
         if(validaLogin.autenticado){//Verifica se o email e senha são
consistentes
           let {id, nome, email} = validaLogin.usuario.dataValues
           dados = {id, nome, email} //Desestruturação dos dados validados
           let token = await auth.gerarToken(dados) //Gera um token JWT
           return res.json({dados, autenticado:true, token:token}).status(200)
            } else{
                return res.json(validaLogin).status(200)
            }

        } catch (error) {
            return res.json(error).status(400)
        }
    })

    //Teste para verificar autenticação
    app.get('/testeAuth', auth.validarToken, async(req, res)=>{
        const usuarioAtual = req.usuarioAtual
        const token = req.token
        res.json({usuarioAtual, token})
    })
}
```

6. Agora basta inserir as seguintes linhas de importação dos métodos de autenticação em cada módulos que define rotas das entidades(no diretório /controllers/rotas

```
const auth = require('../auth')
const validacao = require('../validacao')
```

7. É necessário atualizar as rotas POST e PUT da entidade “usuário”, para que passe a gravar o hash da senha ao invés de senhas com texto puro. No arquivo “./controllers/rotas/usuario.js” realize a importação dos módulos “auth” e “validação” e reescreva as rotas POST e PUT conforme os códigos a seguir:

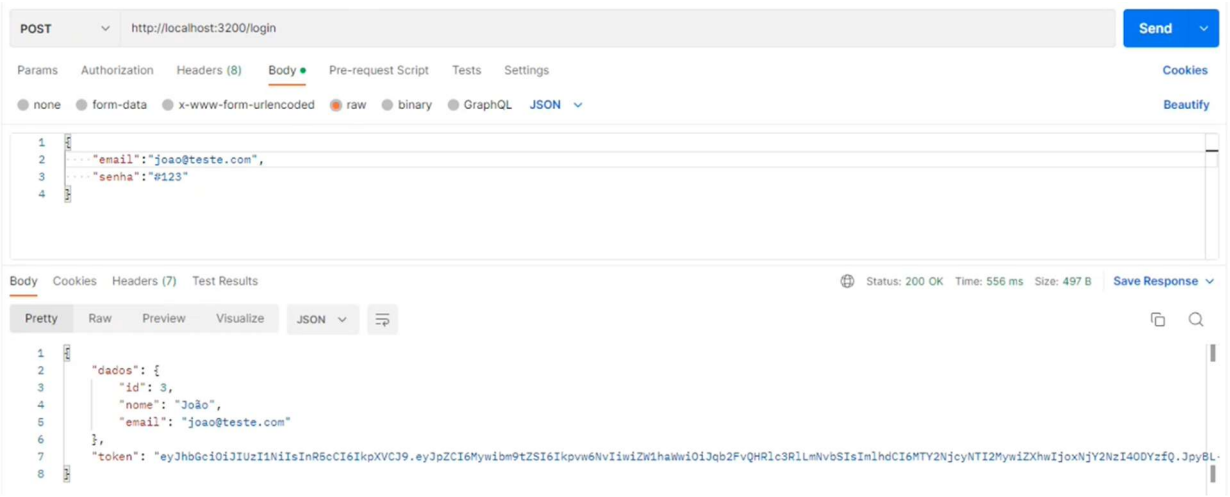
```
app.post(`/${rota}`, async (req, res)=>{
        try {
            let dados = req.body
            let dadosLogin = await validacao.validarCadastro(dados, model)
            if (dadosLogin.validacao){
                dados.senha = await auth.criptografarSenha(dados.senha)
                let respBd = await model.create(dados)
                delete respBd.dataValues.senha
                res.json(respBd).status(201)
            } else {
                res.json(dadosLogin).status(200)
            }

        } catch (error) {
            res.json(error).status(422)
        }
    })
app.put(`/${rota}/:id`, auth.validarToken, async (req, res) => {
        try {
            let id = req.params.id
            let {nome, cpf, telefone, whatsapp} = req.body
            //Para evitar atualizaçção de dados de Login, atualizar estes
dados exigem regras específicas de validação
            let dados = {nome:nome, cpf:cpf, telefone:telefone,
whatsapp:whatsapp}
            console.log(dados)
            let respBd = await model.update(dados, {where:{id:id}})
            console.log(respBd)
            res.json(respBd).status(200)
        } catch (error) {
            res.json(error).status(400)
        }
    })
```

8. E inserir **EM TODAS AS ROTAS** que devem ser **PROTEGIDAS** o meddleware "**auth.validarToken**", seguindo a sintaxe do exemplo abaixo:

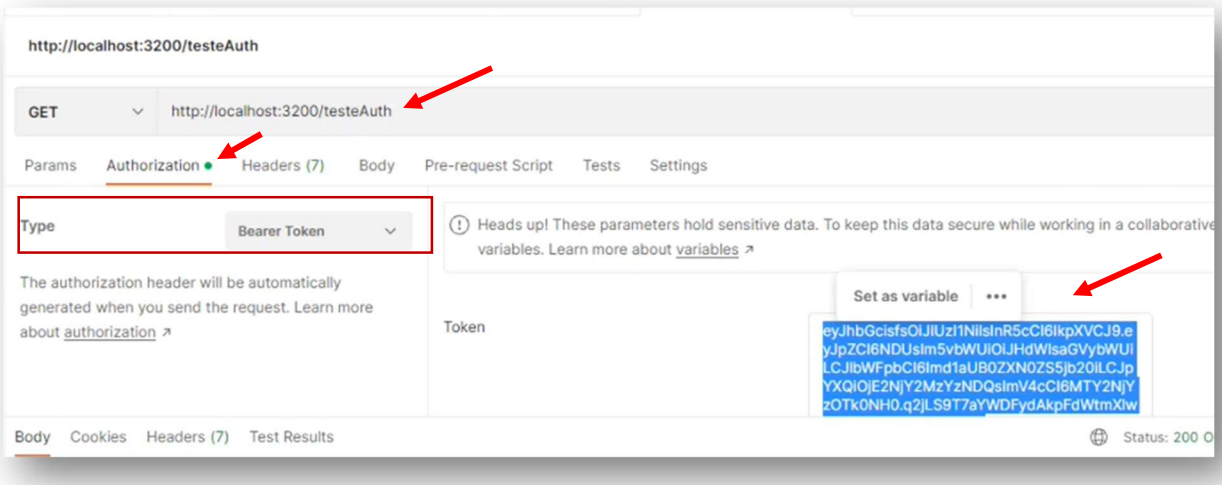
```
app.put(`/${rota}/:id`, auth.validarToken, async (req, res) => {
```

9. Vamos realizar primeiro um teste ao inserir registro de alguns usuários e depois observar o campo senha no banco de dados e perceber que o campo senha não contém a senha em texto puro, e sim um **hash** (senha criptografada).

10. Ao executar a rota "/login" com dados consistentes(combinação de e-mail e senha que existam no banco de dados), a resposta no Postman será a seguinte:



11. Agora é possível testar o acesso a uma rota protegida. Para isso, acesse uma rota protegida como no exemplo da imagem abaixo, **copiando o token gerado pela rota login** e colando em no cabeçalho “Authorization” com a clausula “Type” configurada com o balor “Bearer Token”:



Exercícios extras

Estas atividades extras têm o objetivo aperfeiçoar o processo proteção de rotas e teste de funcionalidade utilizando o Postman.

1. Inserir o middleware “**auth.validarToken**” nas rotas das estidades “pets” e “doações” que devem ser protegidas;
2. Fazer login e realizar teste acessando rotas GET, POST, PUT e DELETE com e sem o token e verificando o comportamento da aplicação (se nega ou autoriza o acesso conforme a passagem do token)