**Fundação valeparaibana de ensino**colégios univap – unidade centro

Curso técnico em informática

Thiago cesar carvalho

.

**LISTA DE EXERCÍCIOS 4º bimestre**

PROGRAMAÇÃO AVANÇADA PARA WEB

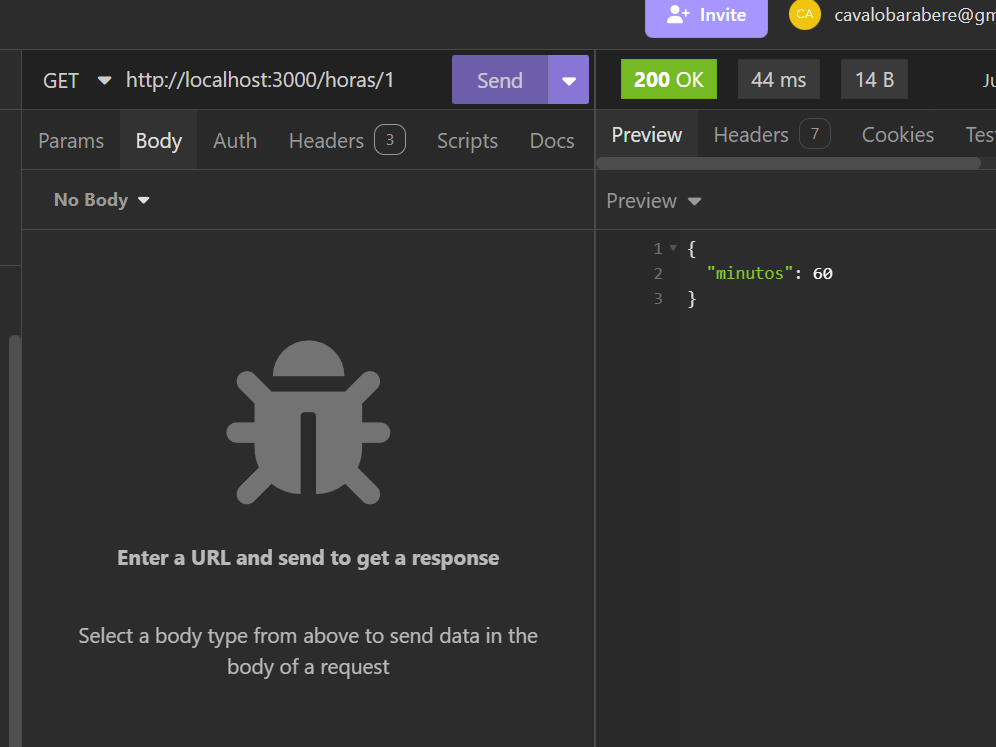
Lista apresentada ao Curso Técnico de informática como composição de nota.

Prof. Me. Hélio Lourenço Esperidião Ferreira

SÃO JOSÉ DOS CAMPOS  
2023

**PROGRAMAÇÃO AVANÇADA PARA WEB  
LISTA DE EXERCÍCIOS II – PHP BÁSICO**

1. Crie apis na arquitetura rest API utilizando conceitos de middleware e roteador para os seguintes problemas. Para cada problema crie uma api em Javascript, flask e c#:  
   1. Converter uma quantidade de horas digitadas pelo usuário em minutos. Informe o resultado em minutos.



-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**

App.js

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

console.log("oi")

app.use ('/horas' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

Router

const express = require("express")

const controlehorario= require("../controle/controlehorario")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleHoras= new controlehorario()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.get ('/:horas' ,

this.\_middleWare.validar\_positivo,

this.\_controleHoras.controle\_horas

)

return this.\_router;

}

}

MiddleWare

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_positivo (request,response,next) {

const horas = request.params.horas;

if (horas < 0) {

const objResposta = {

status: false,

msg: "Digite um valor valido mair que zero!"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

Controle

const express = require("express")

const horario = require("../modelo/horas")

module.exports = class controlehorario {

constructor () {

this.\_router = express.Router( )

this.\_horario = new horario()

}

async controle\_horas (request , response) {

try {

const horas = request.params.horas ;

const objhorario = new horario()

objhorario.horas = horas

console.log ("muinutos >>>")

const existeMinutos = await objhorario.minutos()

const objResposta = {

status : true,

msg : `A conversao de ${horas} em minutos = ${existeMinutos}`

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

Modelo

const express = require("express")

module.exports = class horas {

constructor ( ) {

this.\_horas = null

}

async minutos ( ) {

return this.\_horas \* 60

}

set horas(valor) {

this.\_horas = valor;

}

get horas() {

return this.\_horas;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

App

from flask import Flask, jsonify

from controle.controleHorario import horarioController

app = Flask("api horario")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/horarios/<float:horas>', methods=['get'])

def get\_horas (horas):

try:

controlHorario = horarioController()

controlHorario.horario.horas = horas

min = controlHorario.calcularMinutos()

jsonResposta = {

"min": min

}

return jsonify(jsonResposta),200

except ValueError as error :

return handle\_validation\_error(error)

app.run(host='0.0.0.0', port=3000)

Controle

from model.Horario import Horario

class horarioController:

def \_\_init\_\_(self):

self.\_horario = Horario()

def validarHoras(self):

if self.\_horario.horas <= 0.0:

raise ValueError("As horas devem ser maiores que zero.")

def calcularMinutos(self):

self.validarHoras()

return self.\_horario.CalcularMinutos() # Supondo que o método seja lowercase

@property

def horario(self):

return self.\_horario

# Se necessário, adicione métodos para manipular horas

def set\_horas(self, value):

self.\_horario.horas = value

Modelo

import math

class Horario :

def \_\_init\_\_(self):

self.\_horas =None

def CalcularMinutos(self) :

if self.\_horas is not None :

return self.\_horas \* 60

else:

return None

@property

def horas(self):

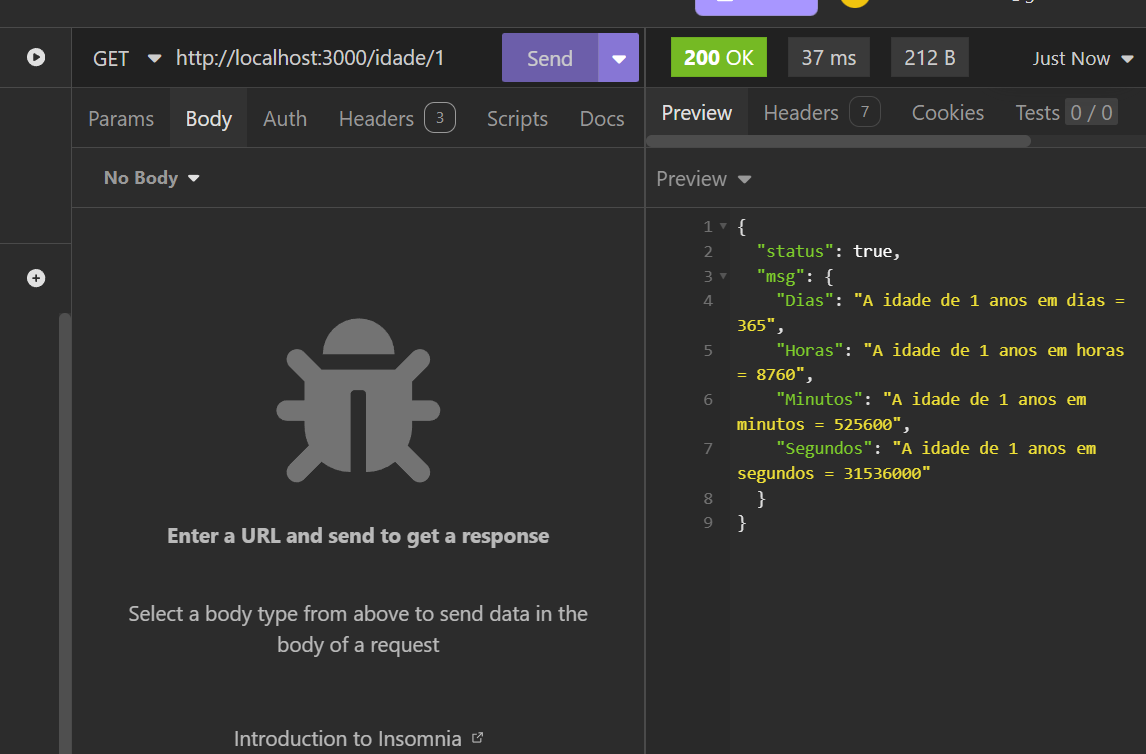
return self.\_horas

@horas.setter

def horas(self,value):

self.\_horas = value

* 1. Crie um programa que dada a idade de uma pessoa calcule quantos dias, horas, minutos e segundo essa pessoa já viveu



-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**

APP

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/idade' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

ROUTER

const express = require("express")

const controleIdade= require("../controle/controleIdade")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleIdade= new controleIdade()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.get ('/:idade' ,

this.\_middleWare.validar\_positivo,

this.\_controleIdade.controle\_idade

)

return this.\_router;

}

}

MIDDLEWARE

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_positivo (request,response,next) {

const horas = request.params.idade;

if (horas < 0) {

const objResposta = {

status: false,

msg: "Digite uma idade valido maior que zero!"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

CONTROLE

const express = require("express")

const idade = require("../modelo/idade")

module.exports = class controleIdade {

constructor () {

this.\_router = express.Router( )

this.\_idade = new idade()

}

async controle\_idade (request , response) {

try {

const idadeval = request.params.idade ;

const objIdade = new idade()

objIdade.idade = idadeval

const dias = await objIdade.calcularDias()

const horas = await objIdade.calcularHoras()

const minutos = await objIdade.calcularMinutos()

const segundos = await objIdade.calcularSegundos()

const objResposta = {

status : true,

msg : {

Dias : `A idade de ${idadeval} anos em dias = ${dias}` ,

Horas : `A idade de ${idadeval} anos em horas = ${horas}` ,

Minutos : `A idade de ${idadeval} anos em minutos = ${minutos}` ,

Segundos : `A idade de ${idadeval} anos em segundos = ${segundos}` ,

}

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

MODELO

// ./modelo/idade.js

module.exports = class Idade {

constructor() {

this.\_idade = null

}

async calcularDias() {

return this.\_idade \* 365

}

async calcularHoras() {

const dias = await this.calcularDias()

return dias \* 24

}

async calcularMinutos() {

const horas = await this.calcularHoras()

return horas \* 60

}

async calcularSegundos() {

const minutos = await this.calcularMinutos()

return minutos \* 60

}

set idade(valor) {

this.\_idade = valor;

}

get idade() {

return this.\_idade;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

App

from flask import Flask, jsonify

from controle.controleIdade import idadeController

app = Flask("api")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/idades/<int:idade>', methods=['GET'])

@app.route('/idades/<float:idade>', methods=['GET'])

def get\_horas(idade):

try:

controlIdade = idadeController()

controlIdade.idade.idade = float(idade) # Converte o inteiro para float se necessário

dias = controlIdade.calcularDias()

horas = controlIdade.calcularHoras()

min = controlIdade.calcularMinutos()

seg = controlIdade.calcularsegundo()

jsonResposta = {

"dias": dias,

"horas" : horas,

"min": min,

"seg" : seg

}

return jsonify(jsonResposta), 200

except ValueError as error:

return handle\_validation\_error(error)

app.run(host='0.0.0.0', port=3000,debug=True)

Controle

from model.Idade import Idade

class idadeController:

def \_\_init\_\_(self):

self.\_idade = Idade()

def validarIdade(self):

if self.\_idade.idade <= 0.0:

raise ValueError("A idade devem ser maior que zero.")

def calcularDias(self):

self.validarIdade()

return self.\_idade.CalcularDias()

def calcularHoras(self):

return self.\_idade.CalcularHoras()

def calcularMinutos(self):

return self.\_idade.CalcularMinutos()

def calcularsegundo (self):

return self.\_idade.CalcularSegundo()

@property

def idade(self):

return self.\_idade

@idade.setter

def set\_idade(self, value):

self.\_idade.idade = value

Modelo

import math

class Idade :

def \_\_init\_\_(self):

self.\_idade =None

def CalcularDias(self) :

return self.\_idade \* 365

def CalcularHoras(self) :

self.CalcularDias()

return self.\_idade \* 24

def CalcularMinutos(self) :

self.CalcularHoras()

return self.\_idade \* 60

def CalcularSegundo(self) :

self.CalcularMinutos()

return self.\_idade \* 60

@property

def idade(self):

return self.\_idade

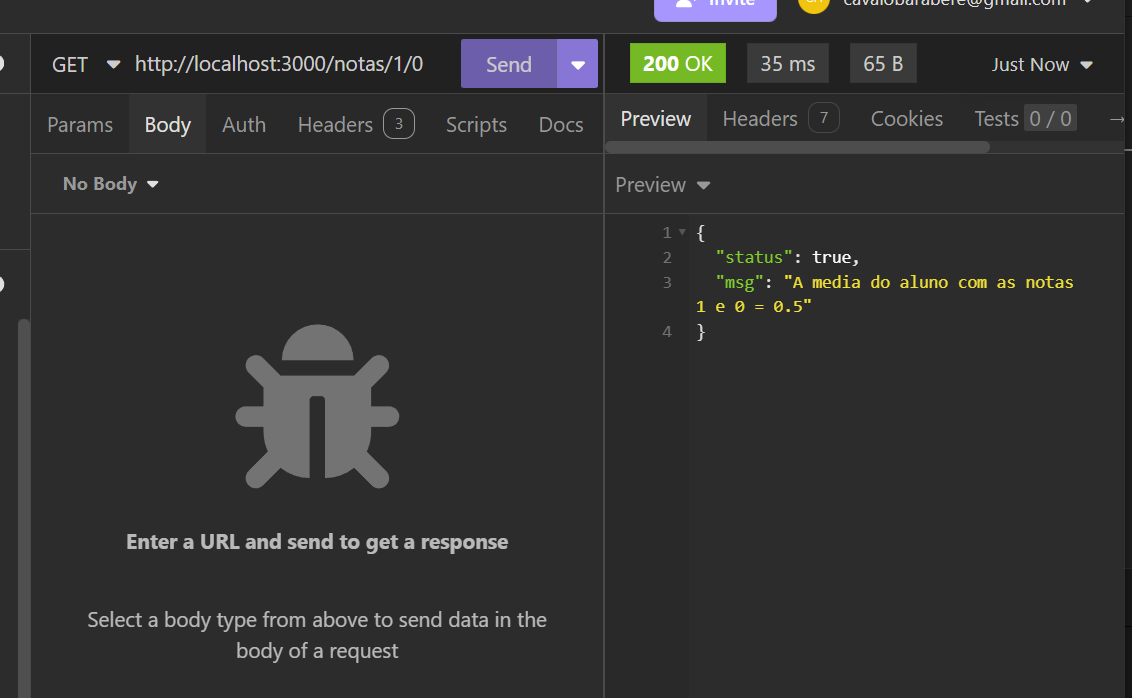
@idade.setter

def idade(self,value):

self.\_idade = value

* 1. Crie um programa que leia duas notas de um aluno e apresente a média.

-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**



APP

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/notas' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

ROUTER

const express = require("express")

const controlenotas= require("../controle/controlenotas")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controlenota= new controlenotas()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.get ('/:n1/:n2' ,

this.\_middleWare.validar\_n1,

this.\_middleWare.validar\_n2,

this.\_controlenota.controle\_media

)

return this.\_router;

}

}

MIDDLEWARE

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_n1 (request,response,next) {

const n1 = request.params.n1;

if (n1 < 0) {

const objResposta = {

status: false,

msg: "Digite a primeira nota valida e maior que zero!"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_n2 (request,response,next) {

const n2 = request.params.n1;

if (n2 < 0) {

const objResposta = {

status: false,

msg: "Digite a segunda nota valida e maior que zero!"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

CONTROLE

const express = require("express")

const notas = require("../modelo/notas")

module.exports = class controlenotas {

constructor () {

this.\_router = express.Router( )

this.\_nota = new notas()

}

async controle\_media (request , response) {

try {

const n1 = parseFloat

(request.params.n1) ;

const n2 = parseFloat

( request.params.n2) ;

const objnota = new notas()

objnota.n2 = n2

objnota.n1 = n1

const media = objnota.media()

const objResposta = {

status : true,

msg :`A media do aluno com as notas ${n1} e ${n2} = ${media}`

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

MODELO

// ./modelo/idade.js

module.exports = class Idade {

constructor() {

this.\_n1 = null

this.\_n2 = null

}

media () {

return (this.\_n1 + this.\_n2 )/2

}

set n1(valor) {

this.\_n1 = valor;

}

get n1() {

return this.\_n1;

}

set n2(valor) {

this.\_n2 = valor;

}

get n2() {

return this.\_n2;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

App

from flask import Flask, jsonify

from controle.control import mediaController

app = Flask("api")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/medias/<int:nota1>/<int:nota2>', methods=['GET'])

@app.route('/medias/<float:nota1>/<int:nota2>', methods=['GET'])

def get\_horas(nota1,nota2):

try:

controle = mediaController()

controle.media.nota1 = float(nota1) # Converte o inteiro para float se necessário

controle.media.nota2 = float(nota2) # Converte o inteiro para float se necessário

media = controle.calcularMedia()

jsonResposta = {

"media": media

}

return jsonify(jsonResposta), 200

except ValueError as error:

return handle\_validation\_error(error)

app.run(host='0.0.0.0', port=3000,debug=True)

Controle

from model.Media import Media

class mediaController:

def \_\_init\_\_(self):

self.\_media = Media()

def validarNotas(self):

if self.\_media.nota1 <= 0.0 or self.\_media.nota2 <= 0.0:

raise ValueError("As notas devem ser maior que zero.")

def calcularMedia(self):

self.validarNotas()

return self.\_media.calcularMedia()

@property

def media(self):

return self.\_media

@media.setter

def set\_media(self, value):

self.\_media = value

Modelo

import math

class Media :

def \_\_init\_\_(self):

self.\_nota1 =None

self.\_nota2 =None

def calcularMedia(self) :

return (self.\_nota1 + self.\_nota2 )/2

@property

def nota1(self):

return self.\_nota1

@nota1.setter

def nota1(self,value):

self.\_nota1 = value

@property

def nota2(self):

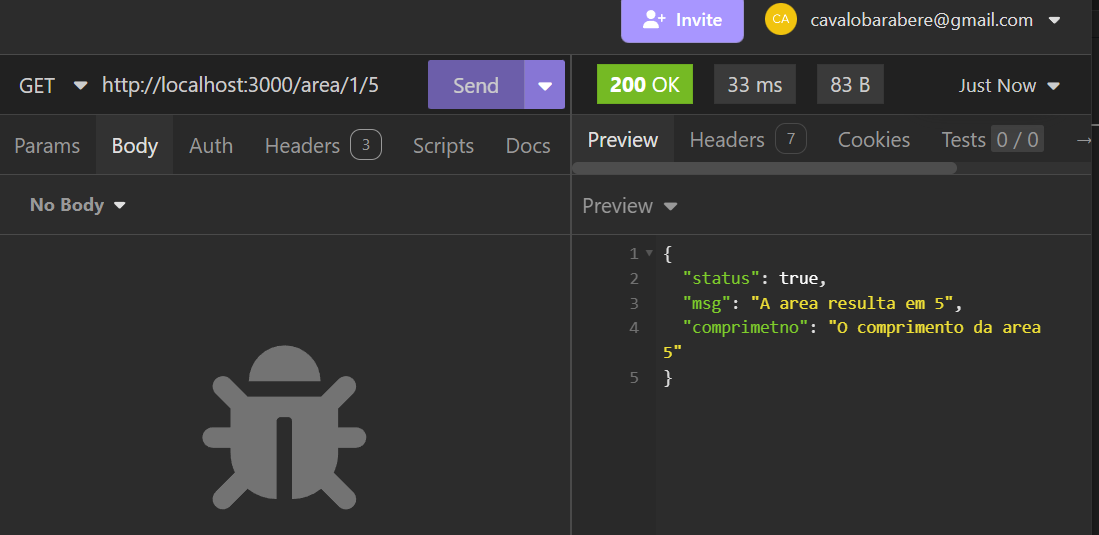
return self.\_nota2

@nota2.setter

def nota2(self,value):

self.\_nota2 = value

* 1. Uma imobiliária vende terrenos retangulares. Faça um programa para ler as dimensões de um terreno e depois exibir a área e comprimento.



APP

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/area' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

ROUTER

const express = require("express")

const controlArea= require("../controle/controleArea")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleArea= new controlArea()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.get ('/:largura/:comprimento' ,

this.\_middleWare.validar\_largura,

this.\_middleWare.validar\_comprimento,

this.\_controleArea.controleArea

)

return this.\_router;

}

}

MIDDLEWARE

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_largura (request,response,next) {

const largura = request.params.largura;

if (largura < 0) {

const objResposta = {

status: false,

msg: "Digite a largura valida e maior que zero!"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_comprimento (request,response,next) {

const comprimento = request.params.comprimento;

if (comprimento < 0) {

const objResposta = {

status: false,

msg: "Digite o comprimento valida e maior que zero!"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

CONTROLE

const express = require("express")

const area = require("../modelo/area")

module.exports = class controlenotas {

constructor () {

this.\_router = express.Router( )

this.\_objArea = new area()

}

async controleArea (request , response) {

try {

const largura = request.params.largura ;

const comprimento = request.params.comprimento ;

const objArea = new area()

objArea.comprimento = comprimento

objArea.largura = largura

const areaExiste = objArea.calcularArea()

const objResposta = {

status : true,

msg :`A area resulta em ${areaExiste}`

comprimetno :largura > comprimento ? `O comrpimento da area ${largura} ` : `O comprimento da area ${comprimento}`

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

MODELO

// ./modelo/idade.js

module.exports = class Idade {

constructor() {

this.\_comprimento = null

this.\_largura = null

}

calcularArea () {

return this.\_comprimento \* this.\_largura

}

set comprimento(valor) {

this.\_comprimento = valor;

}

get comprimento() {

return this.\_comprimento;

}

set largura(valor) {

this.\_largura = valor;

}

get largura() {

return this.\_largura;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

App

from flask import Flask, jsonify

from controle.control import areaController

app = Flask("api")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/medias/<int:medida1>/<int:medida2>', methods=['GET'])

@app.route('/medias/<float:medida1>/<int:medida2>', methods=['GET'])

def get\_horas(medida1,medida2):

try:

controle = areaController()

controle.area.medida1 = float(medida1) # Converte o inteiro para float se necessário

controle.area.medida2 = float(medida2) # Converte o inteiro para float se necessário

area = controle.calcularArea()

comprimento = medida1 if medida1 > medida2 else medida2

jsonResposta = {

"media": area,

"comprimento" : comprimento

}

return jsonify(jsonResposta), 200

except ValueError as error:

return handle\_validation\_error(error)

app.run(host='0.0.0.0', port=3000,debug=True)

Controle

from model.Media import Media

class areaController:

def \_\_init\_\_(self):

self.\_area = Media()

def validarMedidas(self):

if self.\_area.medida2 <= 0.0 or self.\_area.medida1 <= 0.0:

raise ValueError("As medidas devem ser maior que zero.")

def calcularArea(self):

self.validarMedidas()

return self.\_area.calcularArea()

@property

def area(self):

return self.\_area

@area.setter

def set\_area(self, value):

self.\_area = value

Modelo

import math

class Media :

def \_\_init\_\_(self):

self.\_medida1=None

self.\_medida2 =None

def calcularArea(self) :

return self.\_medida1 \* self.\_medida2

@property

def medida1(self):

return self.\_medida1

@medida1.setter

def medida1(self,value):

self.\_medida1 = value

@property

def medida2(self):

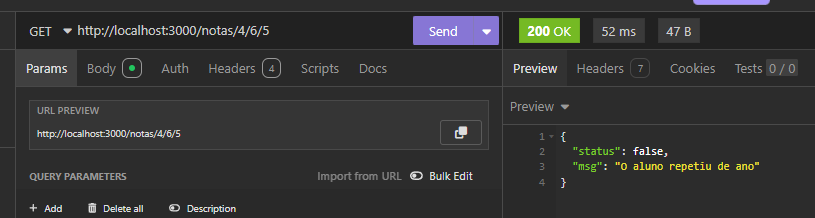
return self.\_medida2

@medida2.setter

def medida2(self,value):

self.\_medida2 = value

* 1. Construa um programa que leia três notas de um aluno, calcule a média obtida por este aluno e no final escreva o resultado indicando se o mesmo foi aprovado ou reprovado (considere que aluno aprovado obteve Média >= 7,0 e aluno reprovado Média < 7,0).



-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**

APP

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/notas' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

ROUTER

const express = require("express")

const controlNotas= require("../controle/controleNota")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleNotas= new controlNotas()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.get ('/:n1/:n2/:n3' ,

this.\_middleWare.validar\_n1,

this.\_middleWare.validar\_n2,

this.\_middleWare.validar\_n3,

this.\_controleNotas.controle\_calcularNotas

)

return this.\_router;

}

}

MIDDLEWARE

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_n1 (request,response,next) {

const n1 = request.params.n1;

if (n1 < 0 || n1> 10) {

const objResposta = {

status: false,

msg: "Digite a nota valida"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_n2 (request,response,next) {

const n2 = request.params.n2;

if (n2 < 0 || n2> 10) {

const objResposta = {

status: false,

msg: "Digite a nota valida"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_n3 (request,response,next) {

const n3 = request.params.n3;

if (n3 < 0 || n3> 10) {

const objResposta = {

status: false,

msg: "Digite a nota valida"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

CONTROLE

const express = require("express")

const boletim = require("../modelo/boletim")

module.exports = class controlenotas {

constructor () {

this.\_router = express.Router( )

this.\_objNotas = new boletim()

}

async controle\_calcularNotas (request , response) {

try {

const n1 = parseFloat(request.params.n1);

const n2 = parseFloat(request.params.n2);

const n3 = parseFloat(request.params.n3);

const objNotas = new boletim()

objNotas.n1 = n1

objNotas.n2 = n2

objNotas.n3 = n3

const rec = objNotas.calcularRec()

console.log(rec)

const objResposta = {

status : rec,

msg : rec ? `O aluno passou de ano` : `O aluno repetiu de ano`

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

MODELO

// ./modelo/idade.js

module.exports = class boletim {

constructor() {

this.\_n1 = null

this.\_n2 = null

this.\_n3 = null

}

calcularRec () {

console.log((this.\_n3 + this.\_n2 + this.\_n1 ))

const media = (this.\_n3 + this.\_n2 + this.\_n1 ) /3

console.log(media)

return media > 7.0

}

set n1(valor) {

this.\_n1 = valor;

}

get n1() {

return this.\_n1;

}

set n2(valor) {

this.\_n2 = valor;

}

get n2() {

return this.\_n2;

}

set n3(valor) {

this.\_n3 = valor;

}

get n3() {

return this.\_n3;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

App

from flask import Flask, jsonify

from controle.control import mediaController

app = Flask("api")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/medias/<int:nota1>/<int:nota2>/<int:nota3>', methods=['GET'])

@app.route('/medias/<float:nota1>/<float:nota2>/<float:nota3>', methods=['GET'])

def get\_horas(nota1,nota2,nota3):

try:

controle = mediaController()

controle.notas.nota1 = float(nota1) # Converte o inteiro para float se necessário

controle.notas.nota2 = float(nota2)

controle.notas.nota3 = float(nota3) # Converte o inteiro para float se necessário

# Converte o inteiro para float se necessário

media = controle.calcularMedia()

aprovado = True if media > 7 else False

jsonResposta = {

"media": media,

"msg" : aprovado

}

return jsonify(jsonResposta), 200

except ValueError as error:

return handle\_validation\_error(error)

app.run(host='0.0.0.0', port=3000,debug=True)

Controle

from modelo.Mediana import Media

class mediaController:

def \_\_init\_\_(self):

self.\_notas = Media()

def validarNotas(self):

if self.\_notas.nota1 <= 0.0 or self.\_notas.nota2 <= 0.0 or self.\_notas.nota3 <= 0.0:

raise ValueError("As medidas devem ser maior que zero.")

def calcularMedia(self):

self.validarNotas()

return self.\_notas.calcularMedia()

@property

def notas(self):

return self.\_notas

@notas.setter

def set\_notas(self, value):

self.\_notas = value

Modelo

import math

class Media :

def \_\_init\_\_(self):

self.\_nota1=None

self.\_nota2 =None

self.\_nota3 =None

def calcularMedia(self) :

return ( self.\_nota1 + self.\_nota2 + self.\_nota3) / 3

@property

def nota1(self):

return self.\_nota1

@nota1.setter

def nota1(self,value):

self.\_nota1 = value

@property

def nota2(self):

return self.\_nota2

@nota2.setter

def nota2(self,value):

self.\_nota2 = value

@property

def nota3(self):

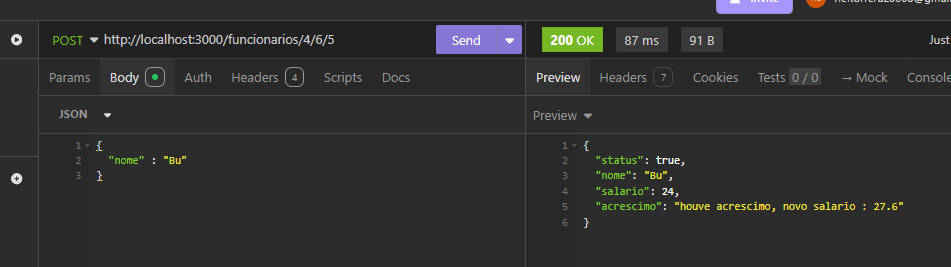
return self.\_nota3

@nota2.setter

def nota3(self,value):

self.\_nota3 = value

* 1. Construa um programa que leia nome de um funcionário, o número de horas trabalhadas, o valor que recebe por horas trabalhadas e o número de filhos, com estas informações, calcular o salário deste funcionário, considerando que o mesmo terá uma gratificação de 3% sobre o salário bruto por cada filho, caso o mesmo possua acima de três filhos. Escreva ao final, o nome do funcionário, seu respectivo salário e o acréscimo de salário, caso ela tenha tido direito a esta gratificação.



-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**

APP

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/funcionarios' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

ROUTER

const express = require("express")

const controlFuncionario= require("../controle//controleFuncionario")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleFuncionario= new controlFuncionario()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.post ('/:horas/:valorHoras/:filhos' ,

this.\_middleWare.validar\_nome,

this.\_middleWare.validar\_horas,

this.\_middleWare.validar\_valorHoras,

this.\_middleWare.validar\_filhos,

this.\_controleFuncionario.controle\_calcularSalario

)

return this.\_router;

}

}

MIDDLEWARE

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_nome (request,response,next) {

const nome = request.body.nome;

if (nome.lenght <= 3) {

const objResposta = {

status: false,

msg: "Digite um nome mairo que 3 caracteres"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_horas (request,response,next) {

const horas = request.params.horas;

if (horas < 0) {

const objResposta = {

status: false,

msg: "Digite um valor para horas valida"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_valorHoras (request,response,next) {

const valorHoras = request.params.valorHoras;

if (valorHoras < 0) {

const objResposta = {

status: false,

msg: "Digite um valor para horas valida"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_filhos (request,response,next) {

const filhos = request.params.filhos;

if (filhos < 0) {

const objResposta = {

status: false,

msg: "Nao existe filhos negativos"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

CONTROLE

const express = require("express")

const funcionario = require("../modelo/funcionario")

module.exports = class controlenotas {

constructor () {

this.\_router = express.Router( )

this.\_funcionario = new funcionario()

}

async controle\_calcularSalario (request , response) {

try {

const nome = request.body.nome;

const horas = parseFloat(request.params.horas);

const valorHoras = parseFloat(request.params.valorHoras);

const filhos = parseFloat(request.params.filhos);

const objFuncionario = new funcionario()

objFuncionario.horas = horas

objFuncionario.valorHoras = valorHoras

objFuncionario.filhos = filhos

objFuncionario.nome = nome

const salario = objFuncionario.calcularSalario()

const objResposta = {

status : true,

nome : nome,

salario : salario,

acrescimo : filhos > 0 ? `houve acrescimo, novo salario : ${objFuncionario.calcularAcrescimo()}` : "nao houve acrescimo"

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

MODELO

// ./modelo/idade.js

module.exports = class funcionario {

constructor() {

this.\_nome = ""

this.\_horas= null

this.\_valorHoras = null

this.\_filhos = null

}

calcularSalario () {

let salario = this.\_horas \* this.\_valorHoras

return salario

}

calcularAcrescimo () {

let salario = this.calcularSalario()

salario = salario + (salario \* ( (this.filhos \* 0.03)) )

return salario

}

get nome() {

return this.\_nome;

}

set nome(valor) {

this.\_nome = valor;

}

// Getter e Setter para \_horas

get horas() {

return this.\_horas;

}

set horas(valor) {

this.\_horas = valor;

}

// Getter e Setter para \_valorHoras

get valorHoras() {

return this.\_valorHoras;

}

set valorHoras(valor) {

this.\_valorHoras = valor;

}

// Getter e Setter para \_filhos

get filhos() {

return this.\_filhos;

}

set filhos(valor) {

this.\_filhos = valor;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

APP

from flask import Flask, jsonify

from Control.funcionarioControl import funcionarioControl

app = Flask("API Retangulo")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/funcionario/<string:nome>/<float:horas\_trabalhadas>/<float:valor\_por\_hora>/<int:numero\_filhos>', methods=['GET'])

def get\_funcionario(nome, horas\_trabalhadas, valor\_por\_hora, numero\_filhos):

try:

funcionarioControl = funcionarioControl()

funcionarioControl.funcionario.horas\_trabalhadas = horas\_trabalhadas

funcionarioControl.funcionario.valor\_por\_hora = valor\_por\_hora

funcionarioControl.funcionario.numero\_filhos = numero\_filhos

salario = funcionarioControl.calcular\_salario()

gratificacao = funcionarioControl.calcular\_gratificacao()

jsonResposta = {

"nome": nome,

"horas\_trabalhadas": horas\_trabalhadas,

"valor\_por\_hora": valor\_por\_hora,

"numero\_filhos": numero\_filhos,

"salario": salario,

"gratificacao": gratificacao

}

return jsonify(jsonResposta), 200

except ValueError as e:

return handle\_validation\_error(e)

app.run(host='0.0.0.0', port=8080)

CONTROLE

from Model.Funcionario import Funcionario

class funcionarioControl:

def \_\_init\_\_(self):

self.\_Funcionario = Funcionario()

def validarHoras(self):

if self.\_Funcionario.horas\_trabalhadas < 0:

raise ValueError("Horas trabalhadas inválidas.")

def validarValor(self):

if self.\_Funcionario.valor\_por\_hora < 0:

raise ValueError("Valor por hora trabalhado inválido.")

def calcular\_salario(self):

self.validarHoras()

self.validarValor()

return self.\_Funcionario.calcularSalario()

def calcular\_gratificacao(self):

self.validarHoras()

self.validarValor()

return self.\_Funcionario.calcularGratificacao()

@property

def funcionario(self):

return self.\_Funcionario

@funcionario.setter

def funcionario(self, valor):

self.\_Funcionario = valor

MODELO

class Funcionario:

def \_\_init\_\_(self):

self.\_valor\_por\_hora = 0.0

self.\_horas\_trabalhadas = 0.0

self.\_numero\_filhos = 0

def calcularSalario(self):

if self.\_valor\_por\_hora > 0 and self.\_horas\_trabalhadas > 0:

return self.\_valor\_por\_hora \* self.\_horas\_trabalhadas

return 0

def calcularGratificacao(self):

salario = self.calcularSalario()

if self.\_numero\_filhos > 3:

filhos\_acima\_de\_3 = self.\_numero\_filhos - 3

return salario \* (filhos\_acima\_de\_3 \* 0.03)

return 0

@property

def valor\_por\_hora(self):

return self.\_valor\_por\_hora

@valor\_por\_hora.setter

def valor\_por\_hora(self, valor):

self.\_valor\_por\_hora = valor

@property

def horas\_trabalhadas(self):

return self.\_horas\_trabalhadas

@horas\_trabalhadas.setter

def horas\_trabalhadas(self, horas):

self.\_horas\_trabalhadas = horas

@property

def numero\_filhos(self):

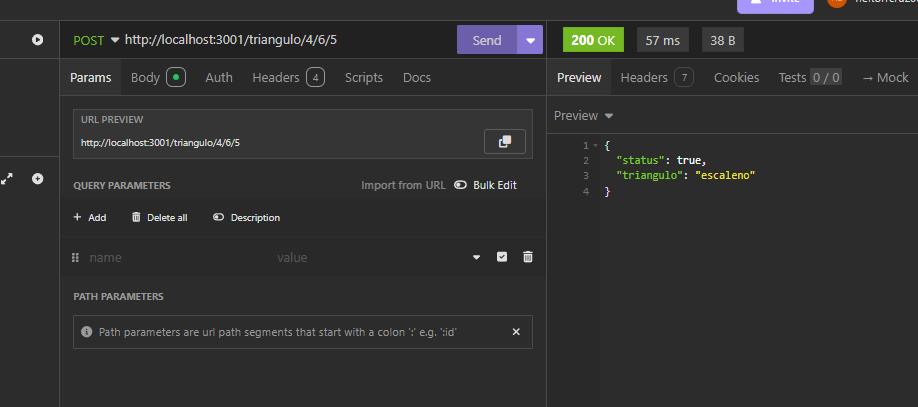
return self.\_numero\_filhos

@numero\_filhos.setter

def numero\_filhos(self, filhos):

self.\_numero\_filhos = filhos

* 1. Construa programa que leia três lados de um triângulo, verifique e escreva que tipo de triângulo eles formam (considere triângulo equilátero com três lados iguais, triângulo isósceles com dois lados iguais e triângulo escaleno com todos os lados diferentes).



-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**

APP

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/triangulo' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

ROUTER

const express = require("express")

const controlTriangulo= require("../controle//controlTriangulo")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleTriangulo= new controlTriangulo()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.post ('/:lado1/:lado2/:lado3' ,

this.\_middleWare.validar\_lado1,

this.\_middleWare.validar\_lado2,

this.\_middleWare.validar\_lado3,

this.\_controleTriangulo.controle\_calcularTipo

)

return this.\_router;

}

}

MIDDLEWARE

const express = require("express")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

validar\_lado1 (request,response,next) {

const lado1 = request.params.lado1;

if (lado1 < 0) {

const objResposta = {

status: false,

msg: "Digite um valor para o lado1 valido"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_lado2 (request,response,next) {

const lado2 = request.params.lado2;

if (lado2 < 0) {

const objResposta = {

status: false,

msg: "Digite um valor para o lado2 valido"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

validar\_lado3 (request,response,next) {

const lado3 = request.params.lado3;

if (lado3 < 0) {

const objResposta = {

status: false,

msg: "Digite um valor para o lado3 valido"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

}

CONTROLE

const express = require("express")

const triangulo = require("../modelo/triangulo")

module.exports = class controlenotas {

constructor () {

this.\_router = express.Router( )

this.\_objTriangulo = new triangulo()

}

async controle\_calcularTipo (request , response) {

try {

const lado1 = parseFloat(request.params.lado1);

const lado2 = parseFloat(request.params.lado2);

const lado3 = parseFloat(request.params.lado3);

const objTriangulo = new triangulo()

objTriangulo.lado1 = lado1

objTriangulo.lado2 = lado2

objTriangulo.lado3 = lado3

const tipo = objTriangulo.descobrirTipo()

const objResposta = {

status : true,

triangulo : tipo

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

MODELO

// ./modelo/idade.js

module.exports = class funcionario {

constructor() {

this.\_lado1 = null

this.\_lado2 = null

this.\_lado3 = null

}

descobrirTipo() {

let tipo = "";

if (this.lado1 === this.lado2 && this.lado1 === this.lado3) {

tipo = "equilátero";

} else if (this.lado1 === this.lado2 || this.lado2 === this.lado3 || this.lado1 === this.lado3) {

tipo = "isósceles";

} else {

tipo = "escaleno";

}

return tipo;

}

get lado1() {

return this.\_lado1;

}

set lado1(valor) {

this.\_lado1 = valor;

}

// Getter e Setter para \_lado2

get lado2() {

return this.\_lado2;

}

set lado2(valor) {

this.\_lado2 = valor;

}

// Getter e Setter para \_lado3

get lado3() {

return this.\_lado3;

}

set lado3(valor) {

this.\_lado3 = valor;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

APP

from flask import Flask, jsonify

from Control.trianguloControl import trianguloControl

app = Flask("API Triangulo")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/triangulo/<int:lado1>/<int:lado2>/<int:lado3>', methods=['GET'])

def get\_triangulo(lado1, lado2, lado3):

try:

control = trianguloControl()

control.triangulo.lado1 = lado1

control.triangulo.lado2 = lado2

control.triangulo.lado3 = lado3

tipo\_triangulo = control.calcular\_Triangulo()

jsonResposta = {

"tipo": tipo\_triangulo,

"lados": [lado1, lado2, lado3]

}

return jsonify(jsonResposta), 200

except ValueError as e:

return handle\_validation\_error(e)

app.run(host='0.0.0.0', port=8080)

CONTROLE

from Model.Triangulo import Triangulo

class trianguloControl:

def \_\_init\_\_(self):

self.\_triangulo = Triangulo()

def validarLado(self):

if self.\_triangulo.lado1 <= 0 or self.\_triangulo.lado2 <= 0 or self.\_triangulo.lado3 <= 0:

raise ValueError("Medida de lado inválida.")

def calcular\_Triangulo(self):

self.validarLado()

return self.\_triangulo.calcularTriangulo()

@property

def triangulo(self):

return self.\_triangulo

@triangulo.setter

def triangulo(self, valor):

self.\_triangulo = valor

MODELO

class Triangulo:

def \_\_init\_\_(self):

self.\_lado1 = 0

self.\_lado2 = 0

self.\_lado3 = 0

def calcularTriangulo(self):

# Verificação de validade dos lados para formar um triângulo

if (

self.\_lado1 + self.\_lado2 <= self.\_lado3 or

self.\_lado1 + self.\_lado3 <= self.\_lado2 or

self.\_lado2 + self.\_lado3 <= self.\_lado1

):

return "Não é um triângulo"

# Verificação do tipo do triângulo

if self.\_lado1 == self.\_lado2 == self.\_lado3:

return "Equilátero"

elif self.\_lado1 == self.\_lado2 or self.\_lado2 == self.\_lado3 or self.\_lado1 == self.\_lado3:

return "Isósceles"

else:

return "Escaleno"

@property

def lado1(self):

return self.\_lado1

@lado1.setter

def lado1(self, valor):

self.\_lado1 = valor

@property

def lado2(self):

return self.\_lado2

@lado2.setter

def lado2(self, valor):

self.\_lado2 = valor

@property

def lado3(self):

return self.\_lado3

@lado3.setter

def lado3(self, valor):

self.\_lado3 = valor

1. Crie uma api completa no padrão REST API em Javascript/ flask/C# da tabela abaixo:

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

* 1. Ao criar o banco de dados considere um registro inicial de administrador do sistema com: usuário: admin; senha: admin. Lembre-se de salvar a senha em md5.
  2. Para ser possível realizar qual operação na API o usuário deve ser autenticado.
  3. O acesso a todas as rotas só deve ser possível mediante validação de token JWT.

**Resposta:**

-==-=-=-=-=-=-=- javascripit -==-=-= -=-=-=-=**-**

App

const express = require('express');

const app = express()

app.use(express.json());

app.use(express.static('js'));

const ClasseRouter = require("./router/Roteador")

const portaServico = 3000;

const objRouter = new ClasseRouter()

app.use ('/usuarios' ,

objRouter.criarRotas()

)

app.listen(portaServico, () => {

console.log(`API rodando no endereço: http://localhost:${portaServico}/`);

});

// ./router/Roteador.js

Router

const express = require("express")

const controlUsuario= require("../controle/controlUsuario")

const middleWare = require("../middleware/middleware")

module.exports = class Roteador {

constructor () {

this.\_router = express.Router( )

this.\_controleUsuario= new controlUsuario()

this.\_middleWare= new middleWare()

}

criarRotas () {

this.\_router.post ('/' ,

this.\_middleWare.validar\_usuario,

this.\_controleUsuario.controle\_cadastrar

)

this.\_router.post ('/:login' ,

this.\_middleWare.validar\_usuario\_login,

this.\_controleUsuario.controle\_login

)

this.\_router.put ('/:idUsuario' ,

this.\_middleWare.validarAutenticacao,

this.\_controleUsuario.controle\_atualizar

)

this.\_router.get ('/:idUsuario' ,

this.\_middleWare.validarAutenticacao,

this.\_controleUsuario.controle\_getID

)

this.\_router.get ('/ReadPage/:idUsuario' ,

this.\_middleWare.validarAutenticacao,

this.\_controleUsuario.controle\_getPage

)

this.\_router.delete ('/:idUsuario' ,

this.\_middleWare.validarAutenticacao,

this.\_controleUsuario.controle\_excluir

)

return this.\_router;

}

}

MiddleWare

const express = require("express")

const Usuario = require("../modelo/Usuario")

const MeuTokenJWT = require("../modelo/MeuTokenJWT")

module.exports = class middleWare

{

constructor () {

this.\_router = express.Router( )

}

async validar\_usuario (request,response,next) {

const email = request.body.emailUsuario;

const objUsuario = new Usuario()

objUsuario.emailUsuario = email

console.log(email)

const existeUsuario =await objUsuario.isUsuario()

if (existeUsuario) {

const objResposta = {

status: false,

msg: "Este email ja esta cadastrado"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}

async validar\_usuario\_login (request,response,next) {

const email = request.body.emailUsuario;

const objUsuario = new Usuario()

objUsuario.emailUsuario = email

try{

const existeUsuario = await objUsuario.isUsuario()

if (!existeUsuario) {

const objResposta = {

status: false,

msg: "Este email nao esta cadastrado ainda"

}

response.status(200).send(objResposta);

} else {

next(); // Chama o próximo middleware ou rota

}

}catch(error) {

console.log("Errorrr >>>" , error)

}

}

async validarAutenticacao (request,response,next) {

const header = request.headers.authorization

try{

const objToken = new MeuTokenJWT()

const existeToken = await objToken.validarToken(header)

if (existeToken) {

next();

}else {

const objResposta = {

status: false,

msg: "Token inválido"

}

response.status(200).send(objResposta);

}

}catch(error) {

console.log("Errorrr >>>" , error)

}

}

}

Controle

const express = require("express")

const Usuario = require("../modelo/Usuario")

const MeuTokenJWT = require("../modelo/MeuTokenJWT")

module.exports = class controlenotas {

constructor () {

this.\_router = express.Router( )

this.\_Usuario = new Usuario()

}

async controle\_cadastrar (request , response) {

try {

const nome = request.body.nomeUsuario;

const email = request.body.emailUsuario;

const senha = request.body.senhaUsuario;

const nascimento = request.body.nascimentoUsuario;

const objUsuario = new Usuario()

objUsuario.nomeUsuario = nome

objUsuario.emailUsuario = email

objUsuario.senhaUsuario = senha

objUsuario.nascimentoUsuario = nascimento

const existeCadastro = objUsuario.cadastrar()

const objResposta = {

status : true,

msg : "Cadastro feito com sucessso"

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

async controle\_login (request , response) {

try {

const email = request.body.emailUsuario;

const senha = request.body.senhaUsuario;

const objClaims = {

emailUsuario : email,

senhaUsuario : senha

}

const objToken = new MeuTokenJWT()

const novoToken = await objToken.gerarToken(objClaims)

const objResposta = {

resposta : "Sucesso ao logar" ,

token : novoToken,

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

async controle\_atualizar (request , response) {

try {

const idUsuario = request.params.idUsuario;

const nome = request.body.nomeUsuario;

const email = request.body.emailUsuario;

const senha = request.body.senhaUsuario;

const nascimento = request.body.nascimentoUsuario;

const objUsuario = new Usuario()

objUsuario.idUsuario = idUsuario

objUsuario.nomeUsuario = nome

objUsuario.emailUsuario = email

objUsuario.senhaUsuario = senha

objUsuario.nascimentoUsuario = nascimento

const existeUsuario = objUsuario.atualizar()

const objResposta = {

status : existeUsuario,

msg : "Sucesso ao atualizar"

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

async controle\_getID (request , response) {

try {

const idUsuario = request.params.idUsuario;

console.log(idUsuario)

const objUsuario = new Usuario()

objUsuario.idUsuario = idUsuario

const existeUsurio = await objUsuario.getID()

const objResposta = {

status : existeUsurio,

dados : existeUsurio

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

async controle\_getPage (request , response) {

try {

const idUsuario = request.params.idUsuario;

console.log(idUsuario)

const objUsuario = new Usuario()

objUsuario.idUsuario = idUsuario

const existeUsurio = await objUsuario.readPage(idUsuario)

const objResposta = {

status : existeUsurio,

dados : existeUsurio

}

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

async controle\_excluir (request , response) {

try {

const idUsuario = request.params.idUsuario;

const objUsuario = new Usuario()

objUsuario.idUsuario = idUsuario

const isDeleted = await objUsuario.delete()

const objResposta = {

cod: 1,

status: isDeleted,

msg: isDeleted ? 'Usuario excluído com sucesso' : 'Erro ao excluir o Usuario'

};

// Envia a resposta HTTP com status 200 e o objeto de resposta.

response.status(200).send(objResposta);

}catch (error) {

console.log("erro >>>>" + error)

return false

}

}

}

Modelo

const Banco = require("../modelo/Banco")

module.exports = class funcionario {

constructor() {

this.\_idUsuario = null

this.\_nomeUsuario = ""

this.\_emailUsuario = ""

this.\_senhaUsuario = ""

this.\_nascimentoUsuario = ""

}

async isUsuario ( ) {

const conexao = Banco.getConexao()

const sql = "select count(\*) as qtd from Usuario where emailUsuario = ?"

try {

const rows = await conexao.promise.execute( sql, [this.emailUsuario])

return rows[0].qtd > 1

} catch (error) {

console.log("Erro ---->>>>" , error)

return false

}

}

async cadastrar () {

const conexao = Banco.getConexao()

const sql = "insert into Usuario (nomeUsuario,emailUsuario,senhaUsuario,nascimentoUsuario) values (?,?,md5(?),?)"

try {

const [result] = await conexao.promise().execute(sql , [this.\_nomeUsuario, this.\_emailUsuario, this.\_senhaUsuario , this.nascimentoUsuario ])

this.\_idUSuario = result.insertId;

return result.affectedRows > 0;

}catch (error) {

console.log("Errro >>>" , error)

return false

}

}

async atualizar () {

const conexao = Banco.getConexao()

const sql = "update Usuario set nomeUsuario = ?,emailUsuario = ?,senhaUsuario = md5(?),nascimentoUsuario = ? where idUsuario = ?"

try {

const [result] = await conexao.promise().execute(sql , [this.\_nomeUsuario, this.\_emailUsuario, this.\_senhaUsuario , this.nascimentoUsuario , this.idUsuario])

return result.affectedRows > 0;

}catch (error) {

console.log("Errro >>>" , error)

return false

}

}

async getID () {

const conexao = Banco.getConexao()

const sql = "select \* from Usuario where idUsuario = ?"

try {

const [result] = await conexao.promise().execute(sql , [this.idUsuario]);

return result;

}catch (error) {

console.log("Errro >>>" , error)

return false

}

}

async readPage(pagina) {

try {

const conexao = Banco.getConexao();

let itens = 5;

let inicio = (pagina - 1) \* itens;

// Garantir que 'inicio' e 'itens' são números válidos

// Construir a query com valores literais

const sql = `SELECT \* FROM Usuario LIMIT ${inicio}, ${itens}`;

console.log("SQL a ser executado:", sql);

const [result] = await conexao.promise().execute(sql);

if (result.length === 0) {

console.log("Nenhum registro encontrado para esta página.");

return [];

}

console.log("Registros encontrados:", result);

return result;

} catch (error) {

console.log("Erro ao executar readPage:", error);

return false;

}

}

async delete () {

const conexao = Banco.getConexao()

const sql = "delete from Usuario where idUsuario = ?"

try {

const [result] = await conexao.promise().execute(sql , [this.idUsuario])

return result.affectedRows > 0;

}catch (error) {

console.log("Errro >>>" , error)

return false

}

}

get idUsuario() {

return this.\_idUsuario;

}

set idUsuario(valor) {

this.\_idUsuario = valor;

}

get nomeUsuario() {

return this.\_nomeUsuario;

}

set nomeUsuario(valor) {

this.\_nomeUsuario = valor;

}

// Getter e Setter para \_emailUsuario

get emailUsuario() {

return this.\_emailUsuario;

}

set emailUsuario(valor) {

this.\_emailUsuario = valor;

}

// Getter e Setter para \_senhaUsuario

get senhaUsuario() {

return this.\_senhaUsuario;

}

set senhaUsuario(valor) {

this.\_senhaUsuario = valor;

}

// Getter e Setter para \_nascimentoUsuario

get nascimentoUsuario() {

return this.\_nascimentoUsuario;

}

set nascimentoUsuario(valor) {

this.\_nascimentoUsuario = valor;

}

}

Meutoken.js

const jwt = require('jsonwebtoken');

module.exports = class MeuTokenJWT {

constructor() {

this.\_key = "K7k6ezDzv7t0DEatig07fdRF2Oe4Y46UG9h35vYgJVk"; // Chave secreta

this.\_alg = 'HS256'; // Algoritmo de criptografia

this.\_type = 'JWT';

this.\_iss = '<http://localhost>'; // Emissor do token

this.\_aud = '<http://localhost>'; // Destinatário do token

this.\_sub = "feira\_tecnica"; // Assunto do token

this.\_duracaoToken = 3600 \* 24 \* 3000; // Duração do token (30 dias)

}

gerarToken(parametroClaims) {

const headers = {

alg: this.\_alg,

typ: this.\_type

};

const payload = {

iss: this.\_iss, // Emissor do token

aud: this.\_aud, // Destinatário do token

sub: this.\_sub, // Assunto do token

iat: Math.floor(Date.now() / 1000), // Momento de criação (em segundos)

exp: Math.floor(Date.now() / 1000) + this.\_duracaoToken, // Expiração (em segundos)

nbf: Math.floor(Date.now() / 1000), // Não é válido antes do tempo especificado

jti: require('crypto').randomBytes(16).toString('hex'), // Identificador único (jti)

email: parametroClaims.emailUsuario,

};

const token = jwt.sign(payload, this.\_key, { algorithm: this.\_alg, header: headers });

return token;

}

validarToken(stringToken) {

if (!stringToken || stringToken.trim() === "") {

return false;

}

const token = stringToken.replace("Bearer ", "").trim();

try {

const decoded = jwt.verify(token, this.\_key, { algorithms: [this.\_alg] });

this.payload = decoded;

return true;

} catch (err) {

// Lidar com diferentes tipos de erro

if (err instanceof jwt.TokenExpiredError) {

console.error("Token expirado");

} else if (err instanceof jwt.JsonWebTokenError) {

console.error("Token inválido");

} else {

console.error("Erro geral", err);

}

return false;

}

}

getPayload() {

return this.payload;

}

setPayload(payload) {

this.payload = payload;

}

getAlg() {

return this.\_alg;

}

setAlg(alg) {

this.\_alg = alg;

}

}

-==-=-=-=-=-=-=- flask-==-=-= -=-=-=-=**-**

App

from flask import Flask, jsonify, request, Response

from controle.control import UsuarioController

app = Flask("api")

def handle\_validation\_error(e):

return jsonify({"erro": str(e)}), 400

@app.route('/usuarios/', methods=['POST'])

def post\_usuarios():

try:

body = request.get\_json()

controle = UsuarioController()

controle.\_objUsuario.nomeUsuario = body['nomeUsuario']

controle.\_objUsuario.emailUsuario = body['emailUsuario']

controle.\_objUsuario.senhaUsuario = body['senhaUsuario']

controle.\_objUsuario.nascimentoUsuario = body['nascimentoUsuario']

return controle.create()

except ValueError as error:

return handle\_validation\_error(error)

@app.route('/usuarios/login', methods=['POST'])

def login\_usuarios():

try:

body = request.get\_json()

controle = UsuarioController()

controle.\_objUsuario.emailUsuario = body['emailUsuario']

controle.\_objUsuario.senhaUsuario = body['senhaUsuario']

return controle.login()

except ValueError as error:

return handle\_validation\_error(error)

@app.route('/usuarios/<int:idUsuario>', methods=['PUT'])

def put\_usuarios(idUsuario):

try:

headers = request.headers

authorization = headers.get('Authorization')

body = request.get\_json()

controle = UsuarioController()

controle.\_objUsuario.idUsuario = idUsuario

controle.\_objUsuario.nomeUsuario = body['nomeUsuario']

controle.\_objUsuario.emailUsuario = body['emailUsuario']

controle.\_objUsuario.senhaUsuario = body['senhaUsuario']

controle.\_objUsuario.nascimentoUsuario = body['nascimentoUsuario']

return controle.update(idUsuario,authorization)

except ValueError as error:

return handle\_validation\_error(error)

@app.route('/usuarios/<int:idUsuario>', methods=['DELETE'])

def delete\_usuarios(idUsuario):

try:

headers = request.headers

authorization = headers.get('Authorization')

controle = UsuarioController()

linhas\_afetadas = controle.delete(idUsuario,authorization)

if linhas\_afetadas:

return jsonify({"message": "Usuario deletado com sucesso"}), 200

else:

return jsonify({"message": "Usuario não encontrado"}), 404

except ValueError as e:

return handle\_validation\_error(e)

@app.route('/usuarios/<int:idUsuario>', methods=['GET'])

def get\_usuariosId(idUsuario):

try:

headers = request.headers

authorization = headers.get('Authorization')

controle = UsuarioController()

controle.\_objUsuario.idUsuario = idUsuario

return controle.readId(idUsuario,authorization)

except ValueError as error:

return handle\_validation\_error(error)

@app.route('/usuarios/readPage/<int:idUsuario>', methods=['GET'])

def get\_usuarios\_readPage(idUsuario):

try:

headers = request.headers

authorization = headers.get('Authorization')

controle = UsuarioController()

controle.\_objUsuario.idUsuario = idUsuario

return controle.readPage(idUsuario,authorization)

except ValueError as error:

return handle\_validation\_error(error)

app.run(host='0.0.0.0', port=3000,debug=True)

Controle

from flask import jsonify # Importação necessária

from modelo.Usuario import Usuario

from modelo.MeuTokenJWT import MeuTokenJWT

class UsuarioController:

def \_\_init\_\_(self):

self.\_objUsuario = Usuario()

def validar\_nomeUsuario(self):

if not self.\_objUsuario.nomeUsuario:

raise ValueError("O nome do Usuario não pode ser vazio")

if len(self.\_objUsuario.nomeUsuario) < 3:

raise ValueError("O nome do Usuario deve ter pelo menos 3 caracteres.")

def validar\_autenticacao(self,token) :

objToken = MeuTokenJWT()

if objToken.validar\_token(token) == True :

return True

else :

return False

def login(self):

objClaims = {

"emailUsuario": self.objUsuario.emailUsuario

}

objToken = MeuTokenJWT() # Correção da instância

try:

token = objToken.gerar\_token(objClaims) # Método correto

return jsonify({

"status": True,

"token": token

}), 201

except Exception as e:

# Melhor tratamento de erro

return jsonify({"message": f"Erro ao gerar token: {str(e)}"}), 500

def create(self):

try:

self.validar\_nomeUsuario()

id\_novo\_usuario = self.\_objUsuario.create()

if id\_novo\_usuario:

return jsonify({

"id\_novo\_usuario": id\_novo\_usuario,

"nomeUsuario": self.\_objUsuario.nomeUsuario

}), 201

else:

return jsonify({"message": "Não foi possível criar o usuário"}), 500

except ValueError as ve:

return jsonify({"message": str(ve)}), 400

except Exception as e:

return jsonify({"message": f"Erro interno: {str(e)}"}), 500

def update(self,idUsuario,authorization):

if (self.validar\_autenticacao(authorization) == True):

try:

self.validar\_nomeUsuario()

id\_atualizado\_usuario = self.\_objUsuario.update(idUsuario)

print(id\_atualizado\_usuario)

if id\_atualizado\_usuario:

return jsonify({

"id\_atualizado\_usuario": id\_atualizado\_usuario,

"nomeUsuario": self.\_objUsuario.nomeUsuario

}), 200

else:

return jsonify({"message": "Não foi possível atualizar o usuário"}), 500

except ValueError as ve:

return jsonify({"message": str(ve)}), 400

except Exception as e:

return jsonify({"message": f"Erro interno: {str(e)}"}), 500

else :

return jsonify({"message": "Não foi possível atualizar o usuário devido ao token invalido"}), 500

def readId(self, id,authorization): # Adicionado parâmetro 'id'

if (self.validar\_autenticacao(authorization) == True):

usuario\_data = self.\_objUsuario.readID(id)

if usuario\_data:

return jsonify(usuario\_data), 200 # Correção para 'usuario\_data'

else:

return jsonify({"message": "Usuário não encontrado"}), 404

else :

return jsonify({"message": "Não foi possível atualizar o usuário devido ao token invalido"}), 500

def readPage(self,idUsuario,authorization):

if (self.validar\_autenticacao(authorization) == True):

usuarios = self.\_objUsuario.readPage(idUsuario)

if usuarios is not None:

return jsonify(usuarios), 200

else:

return jsonify({"message": "Não foi possível obter os usuários"}), 500

else :

return jsonify({"message": "Não foi possível atualizar o usuário devido ao token invalido"}), 500

def delete(self, id,authorization): # Adicionado parâmetro 'id'

if (self.validar\_autenticacao(authorization) == True):

try:

linhas\_afetadas = self.\_objUsuario.delete(id) # Usar Usuario em vez de Cargo

if linhas\_afetadas:

return jsonify({"message": "Usuário excluído com sucesso"}), 200

else:

return jsonify({"message": "Usuário não encontrado"}), 404

except Exception as e:

return jsonify({"message": f"Erro ao excluir usuário: {str(e)}"}), 500

else :

return jsonify({"message": "Não foi possível atualizar o usuário devido ao token invalido"}), 500

@property

def objUsuario(self):

return self.\_objUsuario

@objUsuario.setter

def objUsuario(self, value):

self.\_objUsuario = value

Modelo

Usuario.py

import math

from modelo.Banco import Banco

from mysql.connector import Error

class Usuario :

def \_\_init\_\_(self):

self.\_IdUsuario=None

self.\_nomeUsuario=None

self.\_emailUsuario =None

self.\_senhaUsuario =None

self.\_nascimentoUsuario =None

def create (self) :

conexao = Banco.getConexao()

if conexao:

try:

cursor = conexao.cursor()

sql = "INSERT INTO Usuario (nomeUsuario,emailUsuario,senhaUsuario,nascimentoUsuario) VALUES (%s,%s,%s,%s)"

cursor.execute(sql, (self.nomeUsuario,self.emailUsuario,self.senhaUsuario,self.nascimentoUsuario))

conexao.commit()

self.idUsuario = cursor.lastrowid # Atualiza o idUsuario após criação

return self.idUsuario

except Error as e:

print(f"Erro ao criar cargo: {e}")

return None

finally:

cursor.close()

def update (self , idUsuario) :

conexao = Banco.getConexao()

if conexao:

try:

cursor = conexao.cursor()

sql = "update Usuario set nomeUsuario =%s,emailUsuario =%s,senhaUsuario =%s,nascimentoUsuario =%s where idUsuario = %s"

cursor.execute(sql, (self.nomeUsuario,self.emailUsuario,self.senhaUsuario,self.nascimentoUsuario, idUsuario))

conexao.commit()

return cursor.rowcount

except Error as e:

print(f"Erro ao criar cargo: {e}")

return None

finally:

cursor.close()

def delete (self , idUsuario ) :

conexao = Banco.getConexao()

if conexao:

try:

cursor = conexao.cursor()

sql = "delete from Usuario where idUsuario = %s "

cursor.execute(sql, (idUsuario,))

conexao.commit()

return cursor.rowcount

except Error as e:

print(f"Erro ao criar cargo: {e}")

return None

finally:

cursor.close()

def readID (self ,idUsuario) :

conexao = Banco.getConexao()

if conexao:

try:

cursor = conexao.cursor(dictionary=True)

sql = "select \* from Usuario where idUsuario = %s "

cursor.execute(sql, (idUsuario,))

result = cursor.fetchone()

if result:

self.idUsuario = result['idUsuario']

self.nomeUsuario = result['nomeUsuario']

self.emailUsuario = result['emailUsuario']

self.senhaUsuario = result['senhaUsuario']

self.nascimentoUsuario = result['nascimentoUsuario']

return result

except Error as e:

print(f"Erro ao criar cargo: {e}")

return None

finally:

cursor.close()

def readPage (self ,pagina) :

conexao = Banco.getConexao()

if conexao:

try:

itens = 5

inicio = (pagina -1 ) \* 5

cursor = conexao.cursor(dictionary=True)

sql = "select \* from Usuario limit %s, %s "

cursor.execute(sql, (inicio,itens))

return cursor.fetchall()

except Error as e:

print(f"Erro ao criar cargo: {e}")

return None

finally:

cursor.close()

@property

def idUsuario (self):

return self.\_idUsuario

@idUsuario.setter

def idUsuario(self,value):

self.\_idUsuario = value

@property

def nomeUsuario(self):

return self.\_nomeUsuario

@nomeUsuario.setter

def nomeUsuario(self,value):

self.\_nomeUsuario = value

@property

def emailUsuario(self) :

return self.\_emailUsuario

@emailUsuario.setter

def emailUsuario (self, value):

self.\_emailUsuario = value

@property

def senhaUsuario (self) :

return self.\_senhaUsuario

@senhaUsuario.setter

def senhaUsuario (self , value) :

self.\_senhaUsuario = value

@property

def nascimentoUsuario (self):

return self.\_nascimentoUsuario

@nascimentoUsuario.setter

def nascimentoUsuario (self , value):

self.\_nascimentoUsuario = value

MeuTokenJWT.py

import jwt

import datetime

import secrets

class MeuTokenJWT:

def \_\_init\_\_(self):

self.\_key = "x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYHHTF3" # Chave secreta

self.\_alg = 'HS256' # Algoritmo de criptografia

self.\_type = 'JWT'

self.\_iss = '<http://localhost>' # Emissor do token

self.\_aud = '<http://localhost>' # Destinatário do token

self.\_sub = "acesso\_sistema" # Assunto do token

self.\_duracaoToken = 3600 \* 24 \* 30 # Duração do token (30 dias)

self.payload = None

def gerar\_token(self, parametro\_claims):

headers = {

'alg': self.\_alg,

'typ': self.\_type

}

agora = datetime.datetime.now(datetime.timezone.utc)

payload = {

'iss': self.\_iss, # Emissor do token

'aud': self.\_aud, # Destinatário do token

'sub': self.\_sub, # Assunto do token

'iat': int(agora.timestamp()), # Momento de criação (em segundos)

'exp': int((agora + datetime.timedelta(seconds=self.\_duracaoToken)).timestamp()), # Expiração (em segundos)

'nbf': int(agora.timestamp()), # Não é válido antes do tempo especificado

'jti': secrets.token\_hex(16), # Identificador único (jti)

'email': parametro\_claims['emailUsuario'], # Claims públicas

}

# Gera o token utilizando a biblioteca PyJWT

token = jwt.encode(payload, self.\_key, algorithm=self.\_alg, headers=headers)

return token

def validar\_token(self, string\_token):

if not string\_token or not isinstance(string\_token, str):

print("Token inválido: o token fornecido é nulo ou não é uma string.")

return False

# Verifica se o token tem o prefixo 'Bearer' e o remove

if string\_token.startswith("Bearer "):

token = string\_token[len("Bearer "):].strip()

else:

token = string\_token.strip()

# Verifica se o token não está vazio após a limpeza

if not token:

print("Token inválido: o token está vazio após a remoção do prefixo.")

return False

try:

# Decodifica o token

decoded = jwt.decode(token, self.\_key, algorithms=[self.\_alg], audience=self.\_aud)

self.payload = decoded

return True

except jwt.ExpiredSignatureError:

print("Token expirado")

return False

except jwt.InvalidTokenError:

print("Token inválido")

return False

def get\_payload(self):

return self.payload

def set\_payload(self, payload):

self.payload = payload

def get\_alg(self):

return self.\_alg

def set\_alg(self, alg):

self.\_alg = alg

Banco.py

import mysql.connector

from mysql.connector import Error

class Banco:

@staticmethod

def getConexao():

try:

conexao = mysql.connector.connect(

host='localhost', # Exemplo: 'localhost'

database='sistema\_user', # Nome do banco de dados

user='root', # Seu usuário do MySQL

password='root123' # Sua senha do MySQL

)

if conexao.is\_connected():

print("Conexão com o banco de dados estabelecida.")

return conexao

except Error as e:

print(f"Erro ao conectar ao banco de dados: {e}")

return None

