FUNDAÇÃO VALEPARAIBANA DE ENSINO
COLÉGIOS UNIVAP – UNIDADE CENTRO

CURSO TÉCNICO EM INFORMÁTICA

RODRIGO RODA OLIVETO ALVES
2°H
RA: 50230574

.

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#:

```
const express = require('express'); //importa o express

const RouterA = require('./router/RouterA'); const routerA =
new RouterA();
const RouterB = require('./router/RouterB'); const routerB =
new RouterB();
const RouterC = require('./router/RouterC'); const routerC =
new RouterC();
const RouterD = require('./router/RouterD'); const routerD =
new RouterD();
const RouterE = require('./router/RouterE'); const routerE =
new RouterE();
const RouterF = require('./router/RouterF'); const routerF =
new RouterF();
const RouterG = require('./router/RouterG'); const routerG =
new RouterG();

const app = express(); //recupera uma instancia de express
const portaServico = 2007;
app.use(express.json());


// a) Converte horas para minutos
app.use('/convert-hours',
    routerA.createRoutes()
);


// b) Informar dias horas minutos e segundos vividos por
Idade informada
app.use('/calculate-age',
    routerB.createRoutes()
);


// c) Calcular a média a partir de duas notas informadas
```

```
app.use('/calculate-grade',
    routerC.createRoutes()
)


// d) Calcular área do terreno a partir das dimensões
app.use('/calculate-area',
    routerD.createRoutes()
)


// e) Calcular média e aprovação a partir de três notas
app.use('/calculate-aprovation',
    routerE.createRoutes()
)


// f) Salário e bonificação
app.use('/calculate-salary',
    routerF.createRoutes()
)


// g) Classificar triângulo de acordo com lados
app.use('/classify-triangles',
    routerG.createRoutes()
)

//inicia a espera por requisições http na porta 3000
app.listen(portaServico);
console.log(`Api rodando no endereço:
http:localhost:${portaServico}/`)
```

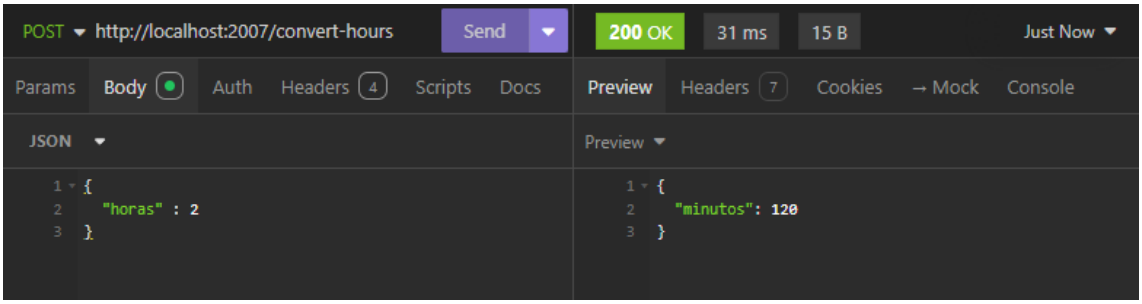
a. Converter uma quantidade de horas digitadas pelo usuário em minutos. Informe o resultado em minutos.

```
const express = require('express');
const                    timeControl                    =
require('../controller/timeController');
const                    MiddlewareA                    =
require('../middleware/MiddlewareA');

module.exports = class RouterA {
```

```
    constructor(){
        this._router = express.Router();
        this._timeControl = new timeControl();
        this._middlewareA = new MiddlewareA();
    }

    createRoutes(){
        this._router.post('/',
            this._middlewareA.validar_hora,
            this._timeControl.convert_hours
        );
        return this._router;
    }
}
```

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

module.exports = class MiddlewareA {
    validar_hora(request,response,next){
        const horas = request.body.horas
        if(isNaN(horas) || horas <= 0){
            const objResposta = {
                status : false,
                 msg : "Horas deve ser um número e maior
que zero!"
            }
            response.status(200).send(objResposta)
        }else{
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class timeControl {
    async convert_hours(request,response){
        const horas = request.body.horas;
        const minutes = horas * 60;

        const resposta = { minutos : minutes};
        response.status(200).send(resposta);
    }
}
```



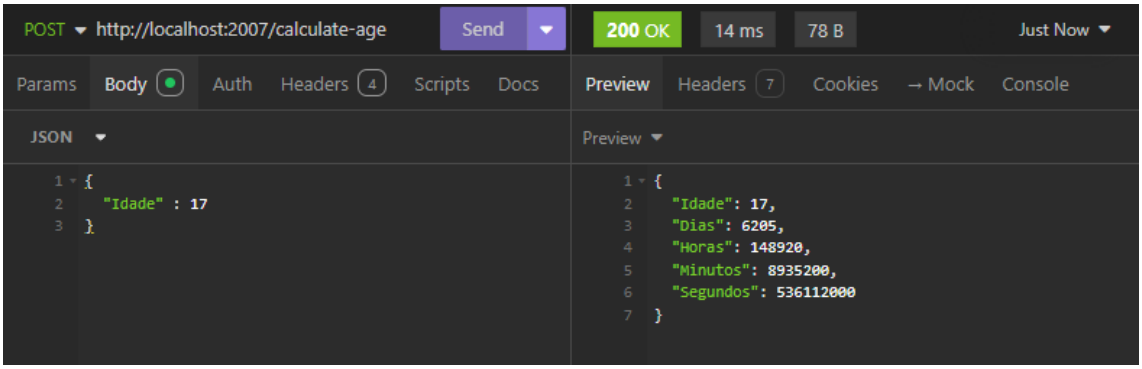
b. Crie um programa que dada a idade de uma pessoa calcule quantos dias, horas, minutos e segundo essa pessoa já viveu

```
const express = require('express');
const                    ageControl                    =
require('../controller/ageController');
const MiddlewareB = require('../middleware/MiddlewareB')

module.exports = class RouterB {
    constructor(){
        this._router = express.Router();
        this._ageControl = new ageControl();
        this._middlewareB = new MiddlewareB();
    }

    createRoutes(){
        this._router.post('/',
            this._middlewareB.validar_idade,
            this._ageControl.calculate_age
        );
        return this._router;
    }
}
```

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

module.exports = class MiddlewareB {
    validar_idade(request,response,next){
        const idade = request.body.Idade
        if(isNaN(idade) || idade <= 0){
            const objResposta = {
                status : false,
                 msg : "Idade deve ser um número e maior
que zero!"
            }
            response.status(200).send(objResposta)
        }else{
```

```
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class ageControl {
    async calculate_age(request,response){
        const age = request.body.Idade;
        const days = age * 365;
        const hours = days*24;
        const minutes = hours*60;
        const seconds = minutes*60;

          const resposta = { Idade: age, Dias : days ,
Horas : hours, Minutos : minutes, Segundos : seconds };
        response.status(200).send(resposta);
    }
}
```



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

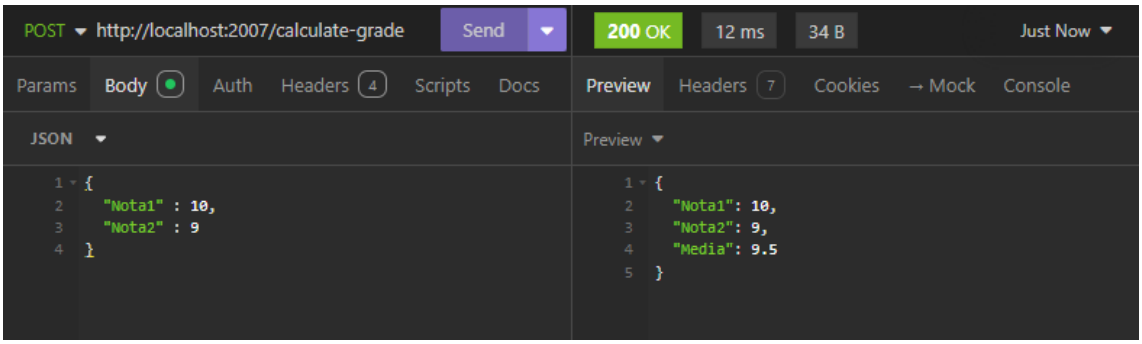
```
const express = require('express');
const                    gradeControl                    =
require('../controller/gradeController');
const MiddlewareC = require('../middleware/MiddlewareC')

module.exports = class RouterC {
    constructor(){
        this._router = express.Router();
        this._gradeControl = new gradeControl();
        this._middlewareC = new MiddlewareC();
    }

    createRoutes(){
```

```
        this._router.post('/',
            this._middlewareC.validar_notas,
            this._gradeControl.calculate_media
        );
        return this._router;
    }
}
```

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

module.exports = class MiddlewareC {
    validar_notas(request,response,next){
        const n1 = request.body.Nota1
        const n2 = request.body.Nota2

        if(isNaN(n1)){
            const objResposta = {
                status : false,
                msg : "As notas devem ser números!"
            }
            response.status(200).send(objResposta)
          }else if(n1 < 0 || n1 > 10 || n2 < 0 || n2 >
10){
            const objResposta = {
                status : false,
                msg : "As notas devem ser de 0 a 10!"
            }
            response.status(200).send(objResposta)
        }else{
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class gradeControl {
    async calculate_media(request,response){
        const grade1 = request.body.Nota1;
        const grade2 = request.body.Nota2;
        const average = (grade1+grade2)/2;
           const resposta = { Nota1 : grade1, Nota2 :
grade2, Media : average};
        response.status(200).send(resposta);
```

```
    }
}
```



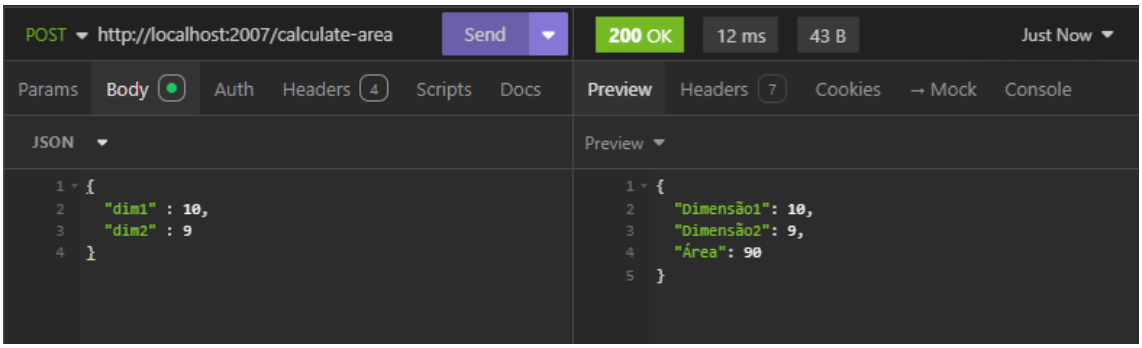
d. Uma imobiliária vende terrenos retangulares. Faça um programa para ler as dimensões de um terreno e depois exibir a área e comprimento.

```
const express = require('express');
const areaControl =
require('../controller/areaController');
const MiddlewareD = require('../middleware/MiddlewareD')

module.exports = class RouterD {
    constructor(){
        this._router = express.Router();
        this._areaControl = new areaControl();
        this._middlewareD = new MiddlewareD();
    }

    createRoutes(){
        this._router.post('/',
            this._middlewareD.validar_dimensoes,
            this._areaControl.calculate_area
        );
        return this._router;
    }
}
```

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

module.exports = class MiddlewareD {
    validar_dimensoes(request,response,next){
        const dim1 = request.body.dim1;
        const dim2 = request.body.dim2;
        if(isNaN(dim1) || isNaN(dim2)){
            const objResposta = {
                status : false,
```

```
                msg : "As dimensões devem ser números!"
            }
            response.status(200).send(objResposta)
        }else if(dim1 <= 0 || dim2 <= 0){
            const objResposta = {
                status : false,
                  msg : "As dimensões devem ser valores
numéricos positivos diferentes de zero!"
            }
            response.status(200).send(objResposta)
        }
        else{
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class areaControl {
    async calculate_area(request,response){
        const dim1 = request.body.dim1;
        const dim2 = request.body.dim2;
        const area = dim1*dim2;
        const resposta = { Dimensão1 : dim1, Dimensão2 :
dim2, Área : area};
        response.status(200).send(resposta);
    }
}
```



e. 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).

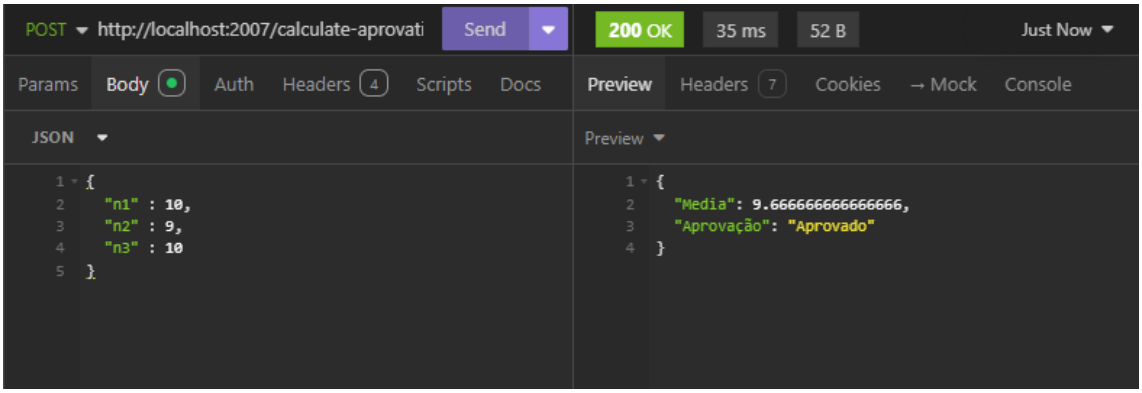
```
const express = require('express');
```

```
const                aprovationControl                =
require('../controller/aprovationController');
const MiddlewareE = require('../middleware/MiddlewareE')

module.exports = class RouterE {
    constructor(){
        this._router = express.Router();
                        this._aprovationControl   =   new
aprovationControl();
        this._middlewareE = new MiddlewareE();
    }

    createRoutes(){
        this._router.post('/',
            this._middlewareE.validar_notas,
            this._aprovationControl.verify_aprovation
        );
        return this._router;
    }
}
```

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

module.exports = class MiddlewareE {
    validar_notas(request,response,next){
        const n1 = request.body.n1;
        const n2 = request.body.n2;
        const n3 = request.body.n3;

        if(isNaN(n1) || isNaN(n2) || isNaN(n3)){
            const objResposta = {
                status : false,
                msg : "As notas devem ser números!"
            }
            response.status(200).send(objResposta)
         }else if(n1 < 0 || n1 > 10 || n2 < 0 || n2 > 10
|| n3 < 0 || n3 > 10){
            const objResposta = {
                status : false,
                msg : "As notas devem ser de 0 a 10!"
            }
            response.status(200).send(objResposta)
        }else{
```

```
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class aprovationControl {
    async verify_aprovation (request,response){
        const n1 = request.body.n1;
        const n2 = request.body.n2;
        const n3 = request.body.n3;
        const media = (n1+n2+n3)/3;
           const aprovacao = media >= 7 ? 'Aprovado' :
"Reprovado";
          const resposta = { Media : media, Aprovação :
aprovacao};
        response.status(200).send(resposta);
    }
}
```



f. 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.

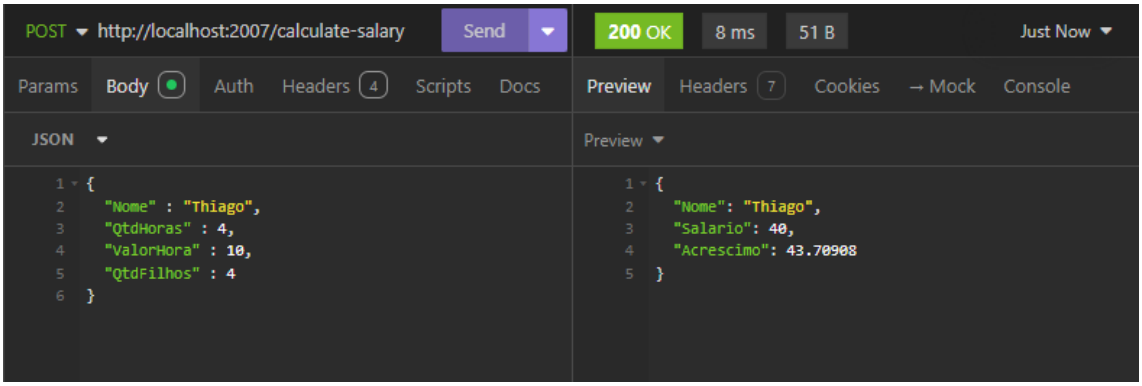
```
const express = require('express');
const                    salaryControl                =
require('../controller/salaryController');
const                     MiddlewareF                 =
require('../middleware/MiddlewareF');

module.exports = class RouterF {
```

```
    constructor(){
        this._router = express.Router();
        this._salaryControl = new salaryControl();
        this._middlewareF = new MiddlewareF();
    }

    createRoutes(){
        this._router.post('/',
            this._middlewareF.validar_dados,
            this._salaryControl.calculate_salary
        );
        return this._router;
    }
}
```

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

module.exports = class MiddlewareF {
    validar_dados(request,response,next){
        let nome = request.body.Nome;
        let horas_trabalhadas = request.body.QtdHoras;
        let valor_hora = request.body.ValorHora;
        let qtd_filhos = request.body.QtdFilhos;

        if(isNaN(horas_trabalhadas) || isNaN(valor_hora)
|| isNaN(qtd_filhos)){
            const objResposta = {
                status : false,
                  msg : "Insira valores válidos para os
campos numéricos!"
            }
            response.status(200).send(objResposta)
        }else if(nome == ""){
            const objResposta = {
                status : false,
                msg : "Insira nome válido!"
            }
            response.status(200).send(objResposta)
        }else if(horas_trabalhadas <= 0 || valor_hora <=
0 || qtd_filhos < 0){
            const objResposta = {
                status : false,
                msg : "Insira numéricos válidos!"
```

```
            }
            response.status(200).send(objResposta)
        }else{
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class salaryControl {
    async calculate_salary(request,response){
        let nome = request.body.Nome;
        let horas_trabalhadas = request.body.QtdHoras;
        let valor_hora = request.body.ValorHora;
        let qtd_filhos = request.body.QtdFilhos;

        let salario = horas_trabalhadas*valor_hora;

        let acrescimo = salario;
        let resposta = {};

        let i = 2;
        if (qtd_filhos > 3) {
            for(i = 2; i<=qtd_filhos ; i++){
                acrescimo += acrescimo*0.03;
            }
            resposta = { Nome : nome, Salario : salario,
Acrescimo : acrescimo};
        }else{
            resposta = {Nome : nome, Salario : salario};
        }
        response.status(200).send(resposta);

    }
}
```



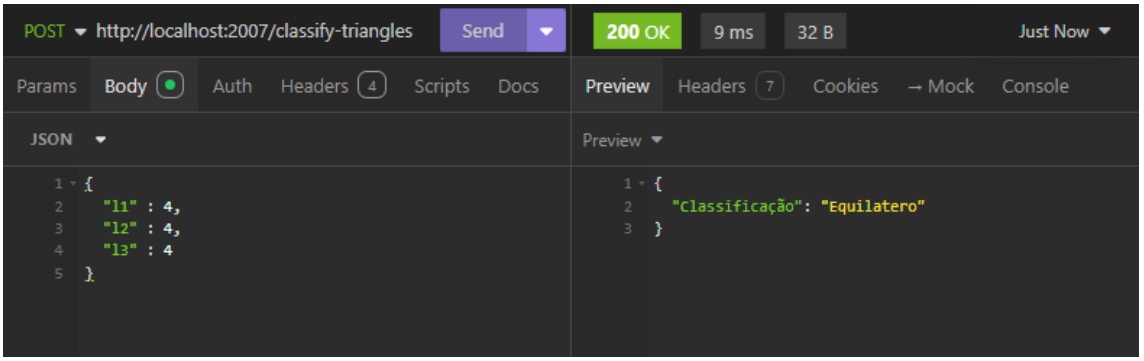
g. Construa programa que leia três lados de um triângulo, verifique e escreva que tipo de triângulo eles formam (considere triângulo equilátero com três lados iguais, triângulo isósceles com dois lados iguais e triângulo escaleno com todos os lados diferentes).

```
const express = require('express');
const                    triangleControl                    =
require('../controller/triangleController');
const MiddlewareG = require('../middleware/MiddlewareG')

module.exports = class RouterG {
    constructor(){
        this._router = express.Router();
        this._triangleControl = new triangleControl();
        this._middlewareG = new MiddlewareG();
    }

    createRoutes(){
        this._router.post('/',
            this._middlewareG.validar_lados ,
            this._triangleControl.classify_triangle
        );
        return this._router;
    }
}
```

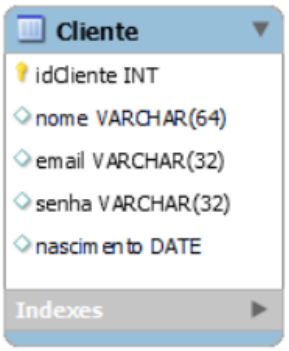
```
const express = require('express');

module.exports = class MiddlewareG {
    validar_lados(request,response,next){
        const l1 = request.body.l1;
        const l2 = request.body.l2;
        const l3 = request.body.l3;

        if(isNaN(l1) || isNaN(l2) || isNaN(l3)){
```

```
            const objResposta = {
                status : false,
                      msg : "Os lados devem ser valores
numéricos!"
            }
            response.status(200).send(objResposta)
        }else if(l1 <= 0 || l2 <= 0 || l3 <= 0){
            const objResposta = {
                status : false,
                      msg : "Os lados devem ser valores
numéricos positivos e maiores que zero!"
            }
            response.status(200).send(objResposta)
        }else{
            next()
        }
    }
}
```

```
const express = require('express');
module.exports = class triangleControl {
    async classify_triangle(request,response){
        const l1 = request.body.l1;
        const l2 = request.body.l2;
        const l3 = request.body.l3;
        let classificacao = "";

        if(l1 == l2 & l2 == l3){
            classificacao = "Equilatero"
        }else if(l1 == l2 || l2 == l3 || l1 == l3){
            classificacao = "Isoceles"
        }else if(l1 != l2 && l1 != l3 && l2 != l3){
            classificacao = "Escaleno"
        }
        const resposta = { Classificação : classificacao
};
        response.status(200).send(resposta);
    }
}
```



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



   a. 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.
   b. Para ser possível realizar qual operação na API o usuário deve ser autenticado.
   c. O acesso a todas as rotas só deve ser possível mediante validação de token JWT.

App.js:

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

const path = require('path');    // Módulo para manipular caminhos de arquivo

const ClienteRouter = require('./router/ClienteRouter');

const LoginRouter = require('./router/LoginRouter');



const app = express();



const portaServico = 8080;
```

```
app.use(express.json());


const clienteRouter = new ClienteRouter();

const loginRouter = new LoginRouter();




app.use('/login',

    loginRouter.createRoutes()

);


app.use('/clientes',

    clienteRouter.createRoutes()

);


// Inicia o servidor, escutando na porta definida, e exibe uma mensagem no console com a URL onde o servidor está rodando.

app.listen(portaServico, () => {

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

});
```

LoginRouter:

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

const LoginControl = require('../controle/LoginControl');
```

```
module.exports = class LoginRouter {

    constructor() {

        this._router = express.Router();
        this._loginControl =  new LoginControl();
    }

    createRoutes() {

        this._router.post('/',
            this._loginControl.login
        );

        return this._router;
    }
}
```

ClienteRouter:

```
const express = require('express');
const                    ClienteMiddleware                    =
require('../middleware/ClienteMiddleware');
const ClienteControl = require('../controle/ClienteControl');
```

```
const JwtMiddleware = require('../middleware/JWTMiddleware');


module.exports = class ClienteRouter {

    constructor() {

        this._router = express.Router();

        this._clienteMiddleware = new ClienteMiddleware();

        this._clienteControl = new ClienteControl();

        this._jwtMiddleware = new JwtMiddleware();

    }

    createRoutes() {

        this._router.get('/',

            this._jwtMiddleware.validate,

            this._clienteControl.readAll

        );


        this._router.get('/:idcliente',

            this._jwtMiddleware.validate,

            this._clienteControl.readById

        );


        this.router.post('/',

            this._jwtMiddleware.validate,

            this._clienteMiddleware.validate_nomecliente,

            this._clienteMiddleware.validate_emailcliente,

            this._clienteMiddleware.validate_senhacliente,

            this._clienteMiddleware.isNotEmailCadastrado,

            this._clienteControl.create
```

```
        );

        this.router.delete('/:idcliente',
            this._jwtMiddleware.validate,
            this._clienteControl.delete
        );

        this.router.put('/:idcliente',
            this._jwtMiddleware.validate,
            this._clienteMiddleware.validate_nomecliente,
            this._clienteMiddleware.validate_emailcliente,
            this._clienteMiddleware.validate_senhacliente,
            this._clienteControl.update
        );

        return this._router;
    }

    get router() {
        return this._router;
    }

    set router(newRouter) {
        this._router = newRouter;
    }
    // Getter e Setter para _clienteMiddleware
    get clienteMiddleware() {
```

```
        return this._clienteMiddleware;
    }

    set clienteMiddleware(newclienteMiddleware) {
        this._clienteMiddleware = newclienteMiddleware;
    }

    // Getter e Setter para _clienteControl
    get clienteControl() {
        return this._clienteControl;
    }

    set clienteControl(newclienteControl) {
        this._clienteControl = newclienteControl;
    }

    // Getter e Setter para _JWTMiddleware
    get jwtMiddleware() {
        return this._jwtMiddleware;
    }

    set jwtMiddleware(newJWTMiddleware) {
        this._jwtMiddleware = newJWTMiddleware;
    }
}
```

MeuTokenJWT:

```
const jwt = require('jsonwebtoken');



class MeuTokenJWT {

    constructor() {

                                                this._key      =
"x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYH
HTF3"; // 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 = "acesso_sistema"; // Assunto do token

         this._duracaoToken = 3600 * 24 * 30; // 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.email, // Claims públicas

            role: parametroClaims.role,

            name: parametroClaims.name,

               idFuncionario: parametroClaims.idFuncionario //
Claims privadas

        };



        // Gera o token utilizando a biblioteca jsonwebtoken

              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;

    }

}

module.exports = MeuTokenJWT;
```

Cliente:

```
// Importa o módulo Banco para realizar conexões com o banco de dados.

const Banco = require('./Banco');


// Define a classe Cliente para representar a entidade Cliente.

class Cliente {

       // Construtor da classe Cliente que inicializa as propriedades.

    constructor() {

        this._idCliente = null;

        this._nome = null;

        this._email = null;

        this._senha = null;

        this._nascimento = null;

    }


     // Método assíncrono para criar um novo funcionário no banco de dados.

    async create() {

            const conexao = Banco.getConexao();   // Obtém a conexão com o banco de dados.
```

```
        const SQL = 'INSERT INTO Cliente (nome, email, senha,
nascimento) VALUES (?, ?, md5(?), ?);';


        try {
                                    const   [result]   =   await
conexao.promise().execute(SQL,    [this._nome,    this._email,
this._senha, this._nascimento]);
            this._idCliente = result.insertId;  // Armazena o
ID gerado pelo banco de dados.
             return result.affectedRows > 0;  // Retorna true
se a inserção foi bem-sucedida.
        } catch (error) {
            console.error('Erro ao criar o cliente:', error);
            return false;
        }
    }


    // Método assíncrono para excluir um funcionário do banco
de dados.
    async delete() {
        const conexao = Banco.getConexao();
          const SQL = 'DELETE FROM Cliente WHERE idCliente =
?;';


        try {
                                    const   [result]   =   await
conexao.promise().execute(SQL, [this._idCliente]);
            return result.affectedRows > 0;
        } catch (error) {
```

```
                console.error('Erro ao excluir o cliente:',
error);

            return false;

        }

    }



      // Método assíncrono para atualizar os dados de um
funcionário no banco de dados.

    async update() {

        const conexao = Banco.getConexao();

         const SQL = 'UPDATE Cliente SET nome = ?, email = ?,
senha = md5(?), nascimento = ? WHERE idCliente = ?;';

           console.log([this._nome, this._email, this._senha,
this._nascimento, this._idCliente]);

        try {

                                    const   [result]   =   await
conexao.promise().execute(SQL,    [this._nome,    this._email,
this._senha, this._nascimento, this._idCliente]);

            return result.affectedRows > 0;

        } catch (error) {

                console.error('Erro ao atualizar o cliente:',
error);

            return false;

        }

    }



     // Método assíncrono para ler todos os funcionários do
banco de dados.

    async readAll() {

        const conexao = Banco.getConexao();
```

```
        const SQL = 'SELECT * FROM Cliente ORDER BY nome;';

        try {
                                        const [rows] = await
conexao.promise().execute(SQL);
            return rows;
        } catch (error) {
            console.error('Erro ao ler clientes:', error);
            return [];
        }
    }

    // Método assíncrono para ler um funcionário pelo seu ID.
    async readByID(idCliente) {
        this._idCliente = idCliente;

        const conexao = Banco.getConexao();
         const SQL = 'SELECT * FROM Cliente WHERE idCliente =
?;';
        try {
                                        const [rows] = await
conexao.promise().execute(SQL, [this._idCliente]);
            return rows;
        } catch (error) {
              console.error('Erro ao ler o cliente pelo ID:',
error);
            return null;
        }
```

```
    }


    async isClienteByEmail(email) {

            const conexao = Banco.getConexao();   // Obtém a
conexão com o banco de dados.


          const SQL = 'SELECT COUNT(*) AS qtd FROM Cliente
WHERE email = ?;';

        try {

                                    const   [rows]   =   await
conexao.promise().execute(SQL, [email]);  // Executa a query.

                return rows[0].qtd > 0;   // Retorna true se
houver algum email no banco

        } catch (error) {

                 console.error('Erro ao verificar o email:',
error);  // Exibe erro no console se houver falha.

              return false;  // Retorna false caso ocorra um
erro.

        }

    }

    async login() {

            const conexao = Banco.getConexao(); // Obtém a
conexão com o banco de dados.

        const SQL = `

            SELECT COUNT(*) AS qtd, idCliente, nome, email

                  FROM Cliente WHERE email = ? AND senha =
MD5(?);`;

         // Query SQL para selecionar o funcionário com base
no email e senha.
```

```
        try {
                    // Prepara e executa a consulta SQL com
parâmetros.
                                        const [rows] = await
conexao.promise().execute(SQL, [this._email, this._senha]);


            if (rows.length > 0 && rows[0].qtd === 1) {
                const tupla = rows[0];
                // Configura os atributos do funcionário.
                this._idCliente = tupla.idCliente;
                this._nome = tupla.nome;
                this._email = tupla.email;
                return true; // Login bem-sucedido.
            }


            return false; // Login falhou.
        } catch (error) {
                  console.error('Erro ao realizar o login:',
error); // Exibe erro no console se houver falha.
                return false; // Retorna false caso ocorra um
erro.
        }
    }


    // Getters e setters para as propriedades da classe.


    get idCliente() {
        return this._idCliente;
```

```
    }

    set idCliente(idCliente) {
        this._idCliente = idCliente;

    }

    get nome() {
        return this._nome;
    }

    set nome(nome) {
        this._nome = nome;

    }

    get email() {
        return this._email;
    }

    set email(email) {
        this._email = email;

    }

    get senha() {
        return this._senha;
```

```
    }

    set senha(senha) {
        this._senha = senha;

    }

    get nascimento() {
        return this._nascimento;
    }

    set nascimento(nascimento) {
        this._nascimento = nascimento;
    }

}

// Exporta a classe Cliente para que possa ser utilizada em outros módulos.
module.exports = Cliente;
```

Banco:

```
const mysql = require('mysql2');

class Banco {
     // Propriedades estáticas para armazenar informações de conexão com o banco de dados
```

```
    static HOST = '127.0.0.1';

    static USER = 'root';

    static PWD = '';

    static DB = 'api';

    static PORT = 3306;

    static CONEXAO = null;


      // Método privado para estabelecer uma conexão com o
banco de dados

    static conectar() {

        // Verifica se já existe uma conexão estabelecida

        if (Banco.CONEXAO === null) {

             // Tenta estabelecer uma nova conexão utilizando
as informações fornecidas

            Banco.CONEXAO = mysql.createConnection({

                host: Banco.HOST,

                user: Banco.USER,

                password: Banco.PWD,

                database: Banco.DB,

                port: Banco.PORT

            });


            // Verifica se ocorreu algum erro na conexão

            Banco.CONEXAO.connect((err) => {

                if (err) {

                    const objResposta = {

                        cod: 1,

                        msg: "Erro ao conectar no banco",
```

```
                        erro: err.message
                    };

console.error(JSON.stringify(objResposta));
                    process.exit(1); // Encerra o script em
caso de erro
                }
            });
        }
    }

     // Método público para obter a conexão com o banco de
dados
static getConexao() {
        // Verifica se já existe uma conexão estabelecida
        if (Banco.CONEXAO === null) {
            // Se não houver, estabelece uma nova conexão
            Banco.conectar();
        }
        // Retorna a conexão
        return Banco.CONEXAO;
    }
}

module.exports = Banco;
```

JWTMiddleware:

```
const MeuTokenJWT = require('../modelo/MeuTokenJWT');

module.exports = class JwtMiddleware {

    validate(request, response, next) {
        const authHeader = request.headers['authorization'];

          // Verifica se o cabeçalho Authorization existe e
começa com "Bearer "
        if (authHeader && authHeader.startsWith('Bearer ')) {
            // Extrai o token removendo "Bearer " do início
da string
            const token = authHeader.split(' ')[1];

            const objMeuTokenJWT = new MeuTokenJWT();
            if (objMeuTokenJWT.validarToken(token) == true) {
                next();
            } else {
                    response.status(401).json({ status: false,
msg: 'Token inválido' });
            }

        } else {
            // Se não houver token, retorna uma mensagem de
erro
             response.status(401).json({ status: false, msg:
'Token não fornecido' });
```

```
        }


    }


}
```

ClienteMiddleware:

```
const Cliente = require('../modelo/Cliente');


module.exports = class clienteMiddleware {



    async isNotEmailCadastrado(request, response, next) {


        const email = request.body.cliente.email;
        const cliente = new Cliente();
        const is = await cliente.isClienteByEmail(email);


        if (is == false) {
            next();
        } else {
            const objResposta = {
                status: false,
                    msg: "Já existe um usuário cadastrado com
este e-mail"
            }
```

```
            response.status(400).send(objResposta);
        }
    }

    async validate_nomecliente(request, response, next) {

        const nome = request.body.cliente.nomecliente;

        if (nome.length < 3) {

            const objResposta = {
                status: false,
                msg: "O nome deve ter mais do que 3 letras"
            }

            response.status(400).send(objResposta);
        } else {

            next();
        }
    }

    async validate_emailcliente(request, response, next) {

        const email = request.body.cliente.email;
```

```
        // Verifica se o e-mail contém "@" e "."
        const atIndex = email.indexOf('@');
        const dotIndex = email.lastIndexOf('.');

           // Verifica se o "@" vem antes do ".", se ambos existem e se há caracteres suficientes antes e depois
         if (atIndex < 1 || dotIndex < atIndex + 2 || dotIndex + 2 >= email.length) {
             const objResposta = {
                 status: false,
                   msg: "E-mail inválido. Por favor, insira um e-mail válido."
             };
             return response.status(400).send(objResposta);
        }

           // Se todas as verificações passarem, o e-mail é considerado válido
        next();
    }


    async validate_senhacliente(request, response, next) {

        const senha = request.body.cliente.senha;

        // Verifica se a senha tem pelo menos 6 caracteres
        if (senha.length < 6) {
```

```
            return response.status(400).send({
                status: false,
                        msg: "A senha deve ter no mínimo 6 caracteres."
            });
        }

        // Verifica se a senha contém pelo menos uma letra
        let temLetra = false;
        for (let i = 0; i < senha.length; i++) {
              if (isNaN(senha[i])) {  // isNaN verifica se o caractere não é um número
                temLetra = true;
                break;
            }
        }

        if (!temLetra) {
            return response.status(400).send({
                status: false,
                    msg: "A senha deve conter pelo menos uma letra."
            });
        }

         // Verifica se a senha contém pelo menos um caractere especial
                                const caracteresEspeciais = "!@#$%^&*()_+-=[]{}|;:'\",.<>?/`~";
```

```
        let temCaractereEspecial = false;


        for (let i = 0; i < senha.length; i++) {
            if (caracteresEspeciais.includes(senha[i])) {
                temCaractereEspecial = true;
                break;
            }
        }


        if (!temCaractereEspecial) {
            return response.status(400).send({
                status: false,
                    msg: "A senha deve conter pelo menos um caractere especial."
            });
        }


         // Se a senha atender a todos os critérios, continua para o próximo middleware
        next();
    }



}
```

LoginControl:

```
// Importa o módulo express para criação de APIs.
```

```
const express = require('express');
const Cliente = require('../modelo/Cliente');
const MeuTokenJWT = require('../modelo/MeuTokenJWT');

module.exports = class LoginControl {

    async login(request, response) {

        const cliente = new Cliente();
        cliente.email = request.body.cliente.email;
        cliente.senha = request.body.cliente.senha;

        const logou = await cliente.login();

        if (logou == true) {
            const payloadToken = {
                email: cliente.idCliente,
                name: cliente.nomeCliente,
                idCliente: cliente.idCliente
            }
            const jwt = new MeuTokenJWT();
                                        const   token_string   =
jwt.gerarToken(payloadToken);

            const objResposta = {
                status: true,
                cod: 1,
```

```
                msg: 'logado com sucesso',


                cliente: {
                    idCliente: cliente.idCliente,
                    nome: cliente.nomeCliente,
                    email: cliente.email,
                },
                token: token_string,


            }
            return response.status(200).send(objResposta);
        }else{
            const objResposta={
                status:false,
                msg:'usuário ou senha inválidos'
            }
            return response.status(401).send(objResposta);
        }



    }


};
```

ClienteControl:

```
const express = require('express');
const Cliente = require('../modelo/Cliente');

module.exports = class ClienteControl {
    async login(request, response) {

        const cliente = new Cliente();

        cliente.email = request.body.cliente.email
        cliente.senha = request.body.cliente.senha

        const logou = cliente.login();
        if (logou == true) {
            const objResposta = {
                cod: 1,
                status: logou,
                msg: 'logado com sucesso'
            };
            response.status(200).send(objResposta);
        } else {
            const objResposta = {
                cod: 2,
                status: isCreated,
                msg: "erro ao efetuar login"
            };
            response.status(401).send(objResposta);
        }
```

```
    }

    async readAll(request, response) {
        const cliente = new Cliente();
        const dadosclientes = await cliente.readAll();
        const objResposta = {
            cod: 1,
            status: true,
            clientes: dadosclientes
        }
        response.status(200).send(objResposta);

    }

    async readById(request, response) {
        const cliente = new Cliente();
        const idcliente = request.params.idcliente

                        const dadosclientes = await
cliente.readByID(idcliente);
        const objResposta = {
            cod: 1,
            status: true,
            clientes: dadosclientes
        }
        response.status(200).send(objResposta);
    }
```

```
async create(request, response) {
    const cliente = new Cliente();

    cliente.nome = request.body.cliente.nomecliente;
    cliente.email = request.body.cliente.email;
    cliente.senha = request.body.cliente.senha;
    cliente.nascimento = request.body.cliente.nascimento;

    const cadastrou = await cliente.create();
    if (cadastrou == true) {
        const objResposta = {
            cod: 1,
            status: true,
            clientes: [{
                "cliente": {
                    //"idcliente": cliente.idcliente,
                    "nomecliente": cliente.nome,
                    "email": cliente.email,
                    "senha": cliente.senha,
                    "nascimento": cliente.nascimento,
                }
            }]
        }
        response.status(201).send(objResposta);
    } else {
        const objResposta = {
```

```
                cod: 1,
                status: false,
                msg: "Falha ao cadastrar cliente",
                clientes: [{
                    "cliente": {
                        //"idcliente": cliente.idcliente,
                        "nomecliente": cliente.nome,
                        "email": cliente.email,
                        "senha": cliente.senha,
                        "nascimento": cliente.nascimento,
                    }
                }]
            }
            response.status(200).send(objResposta);
        }
    }

    async update(request, response) {

        const cliente = new Cliente();

        cliente.idCliente = request.params.idcliente
        cliente.nome = request.body.cliente.nomecliente;
        cliente.email = request.body.cliente.email;
        cliente.senha = request.body.cliente.senha;
        cliente.nascimento = request.body.cliente.nascimento;
```

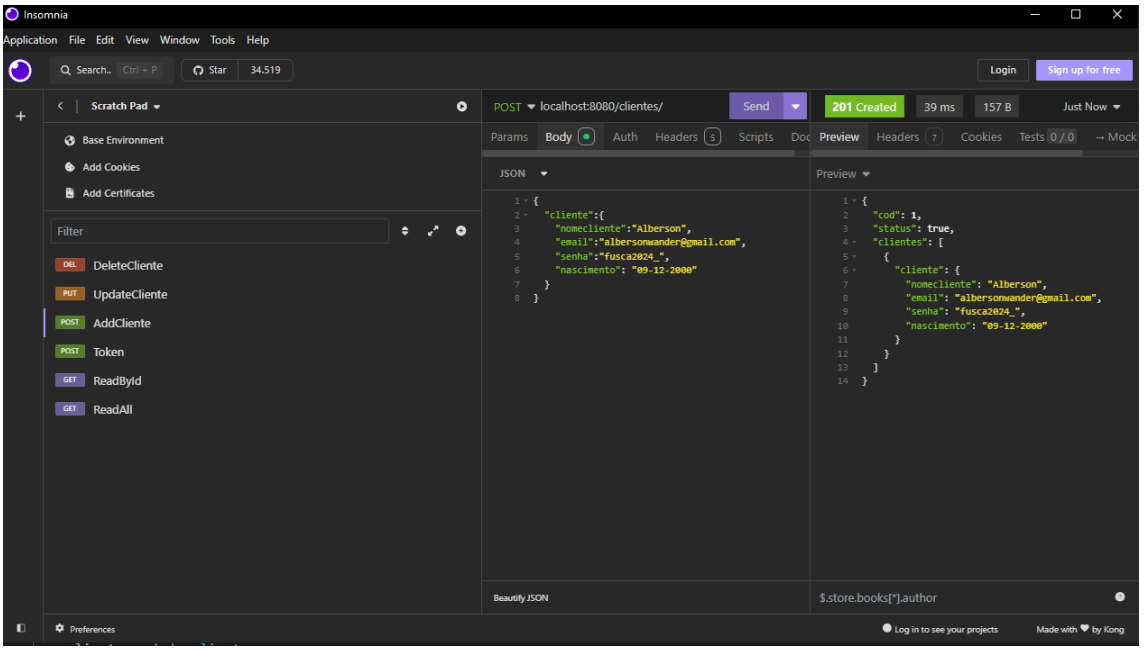
```
                                const    atualizou    =    await
cliente.update(cliente.idCliente);

        if (atualizou == true) {

            const objResposta = {

                cod: 1,

                status: true,

                clientes: [{

                    "cliente": {

                        //"idcliente": cliente.idcliente,

                        "nomecliente": cliente.nome,

                        "email": cliente.email,

                        "senha": cliente.senha,

                        "nascimento": cliente.nascimento,

                    }

                }]

            }

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

        } else {

            const objResposta = {

                cod: 1,

                status: false,

                msg: "Falha ao atualizar cliente",

                clientes: [{

                    "cliente": {

                        //"idcliente": cliente.idcliente,

                        "nomecliente": cliente.nome,

                        "email": cliente.email,
```

```
                    "senha": cliente.senha,
                    "nascimento": cliente.nascimento,
                }
            }]
        }
        response.status(200).send(objResposta);
    }
}



async delete(request, response) {

    const cliente = new Cliente();
    cliente.idCliente = request.params.idcliente



    const excluiu = await cliente.delete();
    if (excluiu == true) {
        const objResposta = {
            cod: 1,
            status: true,
            msg: "Excluido com sucesso",
            clientes: [{
                "cliente": {
                    //"idcliente": cliente.idcliente,
                    "nomecliente": cliente.nomecliente,
                    "email": cliente.email,
```

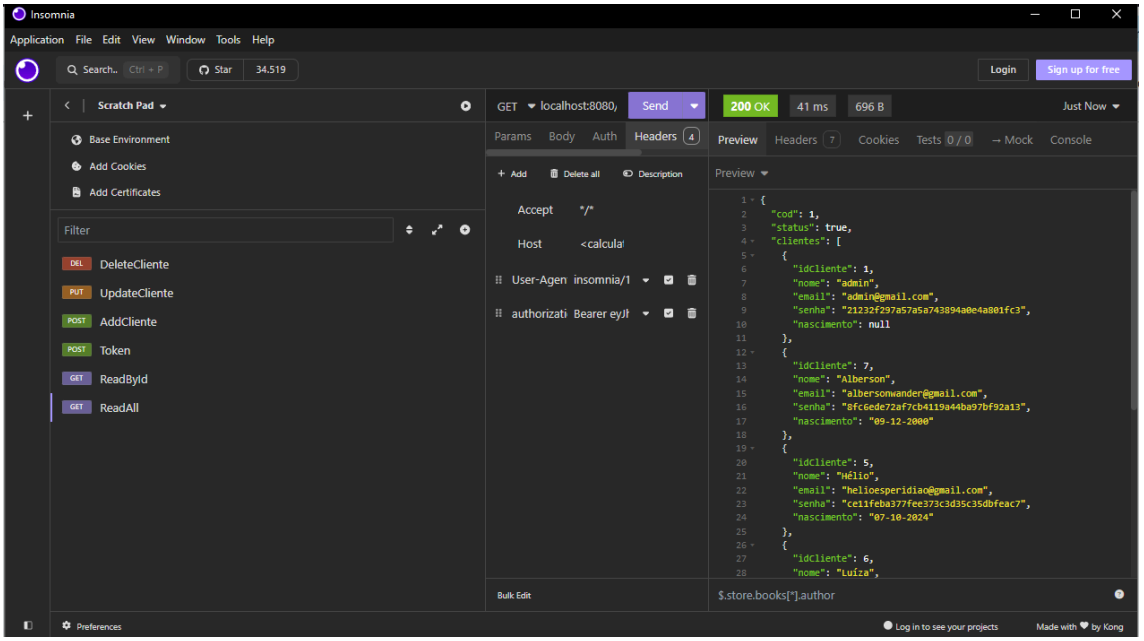
```
                        "senha": cliente.senha,
                        "nascimento": cliente.nascimento,
                    }
                }]
            }
            response.status(200).send(objResposta);
        } else {
            const objResposta = {
                cod: 1,
                status: false,
                msg: "Falha ao excluir cliente",
                clientes: [{
                    "cliente": {
                        "idcliente": cliente.idcliente,
                        "nomecliente": cliente.nomecliente,
                        "email": cliente.email,
                        "senha": cliente.senha,
                        "nascimento": cliente.nascimento,
                    }
                }]
            }
            response.status(200).send(objResposta);
        }
    }
};
```

FUNCIONAMENTO:

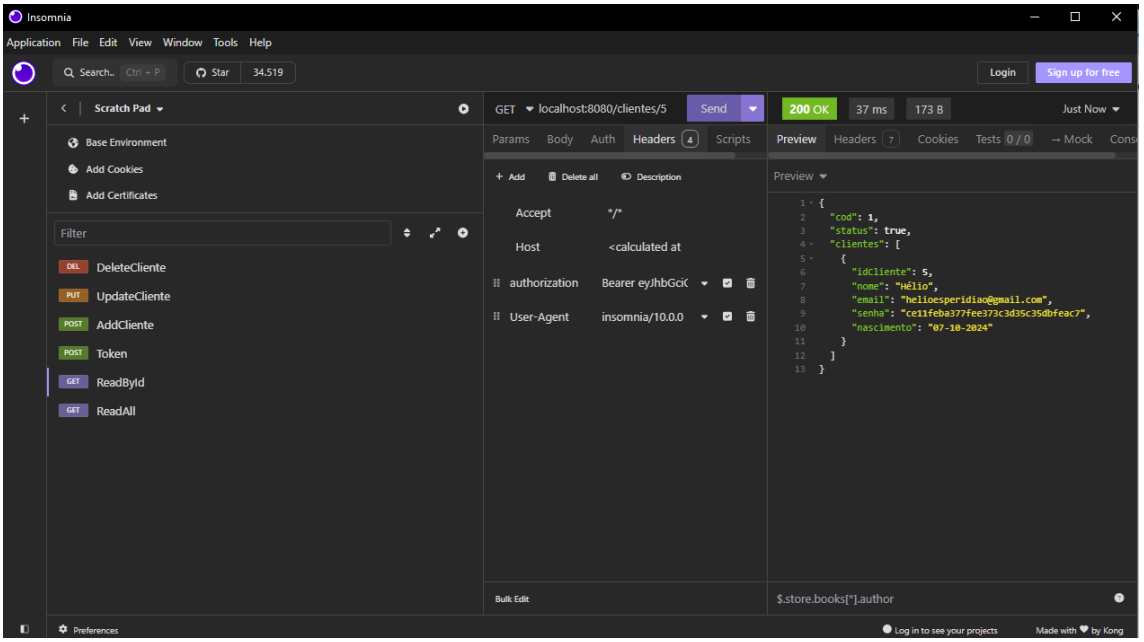
Add Cliente:



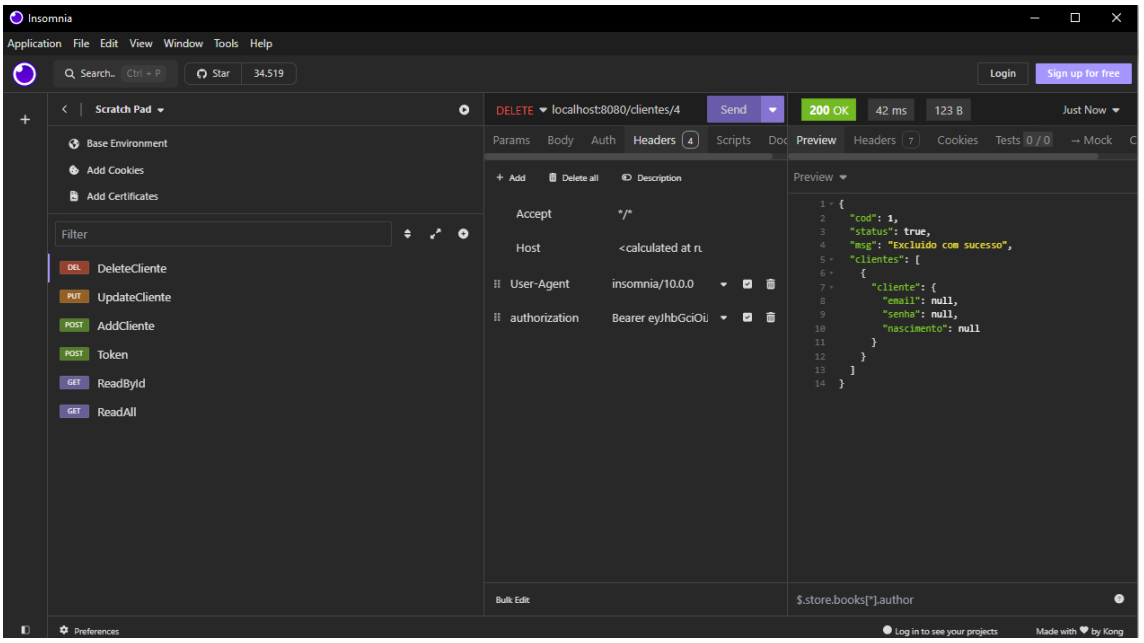
Read All:



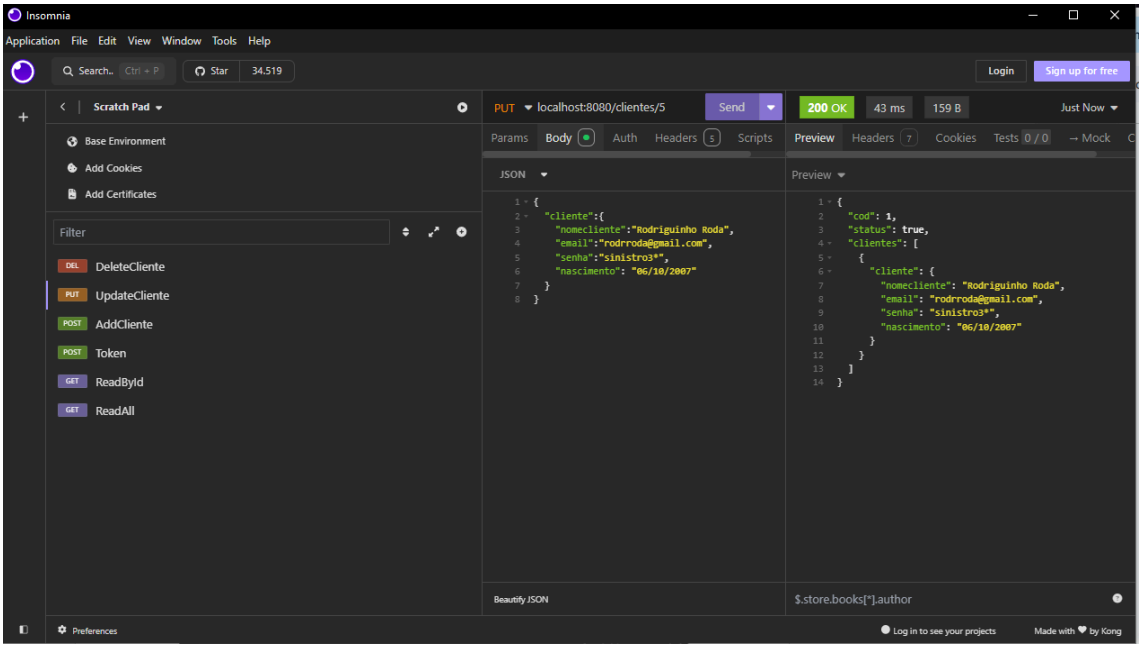
Read By Id:



Delete:



Update:


Insomnia
Application File Edit View Window Tools Help
Search.. Ctrl + P
Star 34.519
Login
Sign up for free
Scratch Pad
Base Environment
Add Cookies
Add Certificates
Filter
DEL DeleteCliente
PUT UpdateCliente
POST AddCliente
POST Token
GET ReadById
GET ReadAll
PUT localhost:8080/clientes/5
Send
200 OK
43 ms
159 B
Just Now
Params Body Auth Headers 5 Scripts
Preview Headers 7 Cookies Tests 0 / 0 → Mock
JSON
Preview
{
"cliente":{
"nomecliente":"Rodriguinho Roda",
"email":"rodrroda@gmail.com",
"senha":"sinistro3*",
"nascimento": "06/10/2007"
}
}
{
"cod": 1,
"status": true,
"clientes": [
{
"cliente": {
"nomecliente": "Rodriguinho Roda",
"email": "rodrroda@gmail.com",
"senha": "sinistro3*",
"nascimento": "06/10/2007"
}
}
]
}
Beautify JSON
$.store.books[*].author
Preferences
Log in to see your projects
Made with ♥ by Kong