**Relatório de Aula Prática - Desenvolvimento**

**Com Framework Para Node.js**

**Aluno: Caio Eduardo A. A. Da Silva**

**FUNDAMENTOS DO NODE.JS.**

**CONSTRUINDO UM SERVIDOR WEB BÁSICO**

Segui o roteiro do portfólio para criar o projeto. Este é o meu arquivo servidor.js:

const *http* = require('http');

const server = *http*.createServer((*req*, *res*) => {

*res*.setHeader('Content-Type', 'text/plain');

    switch (*req*.url) {

        case '/':

*res*.statusCode = 200;

*res*.end('Hello, word');

            break;

        case '/sobre':

*res*.statusCode = 200;

*res*.end('Pagina Sobre');

            break;

        case '/contato':

*res*.statusCode = 200;

*res*.end('Pagina de Contato');

            break;

        default:

*res*.statusCode = 404;

*res*.end('Pagina não encontrada');

            break;

    }

});

const PORT = 3000;

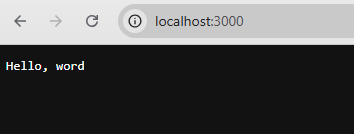
server.listen(PORT, () => {

    console.log(`Servidor rodando em http://localhost:${PORT}`);

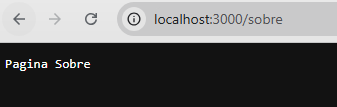
});

Evidências e prints do meu projeto em execução:

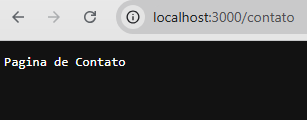
/



/sobre



/contato



**IMPLEMENTAÇÃO E DEPURAÇÃO UTILIZANDO O NODE JS**

**TESTANDO E DEPURANDO APLICAÇÕES NODE.JS**

**1. Criei um novo diretório para o projeto e navegue até ele via terminal:**

mkdir projeto-soma

cd projeto-soma

**2.Inicializei o projeto Node.js:**

npm init -y

**3.Instalei o pacote Mocha:**

npm install mocha --save-dev

**4.Criei um arquivo chamado math.js na raiz do projeto:**

* function soma(*a*, *b*) {
* return *a* + *b*;
* }
* *module*.*exports* = { soma };

module.exports = { soma };

**5.Criei um diretório chamado test**

**6.Dentro do diretório test, criei um arquivo math.test.js:**

* const assert = require('assert');
* const { soma } = require('../math');
* describe('Função Soma', () => {
* it('Deve retornar 5 quando somar 2 e 3', () => {
* assert.strictEqual(soma(2, 3), 5);
* });
* it('Deve retornar -1 quando somar -2 e 1', () => {
* assert.strictEqual(soma(-2, 1), -1);
* });
* it('Deve retornar 0 quando somar 0 e 0', () => {
* assert.strictEqual(soma(0, 0), 0);
* });
* });

**7.Adicione um script de teste no package.json para executar o Mocha:**

"scripts": {

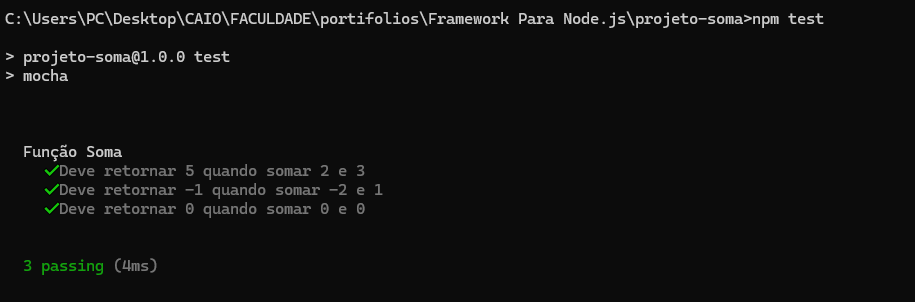
"test": "mocha"

}

**8.Executei os testes com o seguinte comando:**

npm test

**9.Esse foi o resultado:**



**INTERFACE E SEGURANÇA NO NODE.JS**

**DESENVOLVIMENTO DE INTERFACES DE USUÁRIO COM NODE.JS**

**1. Configuração do Projeto**

Para começar, criei uma pasta onde o projeto ficaria armazenado. Fiz isso no terminal com os comandos:

mkdir validacao-cpf

cd validacao-cpf

**2. Segundo Passo: Criação dos Arquivos**

Dentro da pasta do projeto, criei três arquivos principais: cpf.html, cpf.css, e cpf.js. Esses arquivos são responsáveis, respectivamente, pelo layout HTML, pelos estilos CSS, e pela lógica JavaScript de validação.

Para criar esses arquivos pelo terminal, usei:

touch cpf.html cpf.css cpf.js

**3. Terceiro Passo: Estruturação do Código**

Agora eu adicionei o código em cada um desses arquivos.

**HTML (cpf.html)**

No cpf.html, criei um formulário simples que inclui um campo para inserir o CPF e uma área para exibir a mensagem de validação. O código ficou assim:

<!DOCTYPE *html*>

<html *lang*="pt-BR">

<head>

    <meta *charset*="UTF-8">

    <meta *name*="viewport" *content*="width=device-width, initial-scale=1.0">

    <title>Validação de CPF</title>

    <link *rel*="stylesheet" *href*="cpf.css">

</head>

<body>

    <div *class*="container">

        <h2>Validação de CPF</h2>

        <form *id*="form-cpf">

            <label *for*="cpf">CPF:</label>

            <input *type*="text" *id*="cpf" *maxlength*="11" *placeholder*="Digite o CPF (somente números)">

            <p *id*="mensagem-validacao"></p>

        </form>

    </div>

    <script *src*="cpf.js"></script>

</body>

</html>

**CSS (cpf.css)**

Para dar estilo à página, usei o cpf.css e escrevi algumas regras de estilo. Isso ajuda a mostrar as mensagens em cores diferentes (verde para válido e vermelho para inválido). O código ficou assim:

body {

font-family: Arial, sans-serif;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

margin: 0;

background-color: #f0f0f0;

}

.container {

text-align: center;

background-color: #ffffff;

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

width: 300px;

}

input {

width: 100%;

padding: 10px;

margin-top: 10px;

border: 1px solid #ccc;

border-radius: 4px;

box-sizing: border-box;

}

#mensagem-validacao {

margin-top: 10px;

font-weight: bold;

}

.sucesso {

color: green;

}

.erro {

color: red;

}

**JavaScript (cpf.js)**

Por último, escrevi o código JavaScript no cpf.js para validar o CPF. Quando o CPF é inserido, a função verifica se ele está correto e exibe a mensagem correspondente. Usei o seguinte código:

document.getElementById("cpf").addEventListener("input", function() {

const cpf = this.value;

const mensagemValidacao = document.getElementById("mensagem-validacao");

if (cpf.length === 11) {

if (validarCPF(cpf)) {

mensagemValidacao.textContent = "CPF válido!";

mensagemValidacao.className = "sucesso";

} else {

mensagemValidacao.textContent = "CPF inválido!";

mensagemValidacao.className = "erro";

}

} else {

mensagemValidacao.textContent = "";

}

});

function validarCPF(cpf) {

if (/^(\d)\1{10}$/.test(cpf)) return false;

let soma = 0;

let resto;

for (let i = 1; i <= 9; i++) {

soma += parseInt(cpf.substring(i - 1, i)) \* (11 - i);

}

resto = (soma \* 10) % 11;

if (resto === 10 || resto === 11) resto = 0;

if (resto !== parseInt(cpf.substring(9, 10))) return false;

soma = 0;

for (let i = 1; i <= 10; i++) {

soma += parseInt(cpf.substring(i - 1, i)) \* (12 - i);

}

resto = (soma \* 10) % 11;

if (resto === 10 || resto === 11) resto = 0;

if (resto !== parseInt(cpf.substring(10, 11))) return false;

return true;

}

A função validarCPF usa uma fórmula de cálculo para verificar os dígitos verificadores do CPF. Assim que digito o CPF, o JavaScript chama essa função para determinar se o CPF é válido.

**4. Quarto Passo: Testando a Aplicação**

Depois de escrever os códigos, testei a aplicação abrindo o arquivo cpf.html no navegador. No campo de CPF, inseri valores para verificar se o código estava funcionando. A mensagem "CPF válido!" aparece em verde quando o CPF está correto, e "CPF inválido!" aparece em vermelho caso contrário.

**Conclusão**

Esse projeto ajuda a validar o CPF diretamente no navegador, oferecendo uma resposta visual instantânea para o usuário, com um aviso colorido indicando se o CPF está correto ou incorreto.

Evidencias:





**TESTES UTILIZANDO NODE.JS**

**ESTRATÉGIAS DE TESTES**

**1. Configuração do Projeto**

Primeiro, criei uma nova pasta para o projeto e inicializei o projeto Node.js:

mkdir servidor-http

cd servidor-http

npm init -y

**2.Agora instalei as dependências necessárias:**

npm install express mocha chai chai-http --save-dev

**3. Criação do Servidor HTTP**

Em seguida, criei um arquivo chamado server.js para configurar o servidor HTTP com Express.

touch server.js

Dentro de server.js, escrevi o código para criar o servidor e definir as rotas:

const express = require('express');

const app = express();

app.use(express.json());

app.get('/', (*req*, *res*) => {

*res*.send('Hello World');

});

app.post('/data', (*req*, *res*) => {

    const data = *req*.body;

*res*.json({ message: 'Sucesso', data });

});

const PORT = 3000;

app.listen(PORT, () => {

    console.log(`Servidor rodando em http://localhost:${PORT}`);

});

*module*.*exports* = app;

**4. Estrutura do Projeto**

Agora criei o arquivo math.js onde implemento funções de matemática simples, para que o servidor fique mais completo.

touch math.js

// math.js

function soma(a, b) {

return a + b;

}

module.exports = { soma };

**5. Escrever Testes de Integração**

const chai = require('chai');

const chaiHttp = require('chai-http');

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

chai.use(chaiHttp);

const { expect } = chai;

describe('Testes de Integração do Servidor', () => {

    it('Deve retornar "Hello World" na rota GET /', (*done*) => {

        chai.request(server)

            .get('/')

            .end((*err*, *res*) => {

                expect(*res*).to.have.status(200);

                expect(*res*.text).to.equal('Hello World');

                done();

            });

    });

    it('Deve retornar JSON com mensagem de sucesso na rota POST /data', (*done*) => {

        const data = { nome: 'Teste' };

        chai.request(server)

            .post('/data')

            .send(data)

            .end((*err*, *res*) => {

                expect(*res*).to.have.status(200);

                expect(*res*.body).to.be.an('object');

                expect(*res*.body).to.have.property('message').equal('Sucesso');

                expect(*res*.body).to.have.property('data').eql(data);

                done();

            });

    });

});

Agora, criei uma pasta test e dentro dela o arquivo integration.test.js para escrever os testes.

mkdir test

touch test/integration.test.js

**6. Executar os Testes**

Agora configurei o script de teste no package.json para rodar o Mocha:

"scripts": {

    "test": "mocha"

  },

**7.Finalmente, executei os testes com o comando:**

npm test

**8.Evidencias**

