Projeto: Servidor de Irrigação Simulado

Estrutura de Pastas

```
Servidor-Irrigacao/
│
├── index.js                ← Arquivo principal do servidor Node.js
├── package.json            ← Arquivo de configurações do projeto
npm
└── public/                 ← Pasta com os arquivos públicos
(frontend)
    ├── index.html          ← Painel de controle da irrigação
    └── style.css           ← Estilos básicos para o painel
```

Nota: É importante que a pasta `public` esteja no mesmo nível de `index.js` e que não tenha erros de digitação no nome.

Sobre a Simulação

Este projeto simula um sistema de irrigação porque sensores físicos (como DHT11/DHT22 ou GPIO do Raspberry Pi) **não funcionam nativamente em sistemas operacionais como o Windows**.

- Por isso, usamos **valores aleatórios** para `temperatura` e `umidade`.
- A lógica de "ligar/desligar irrigação" também é simulada apenas com uma variável booleana (`irrigando`).

Isso permite **testar a lógica da aplicação e o painel de controle** mesmo sem o hardware real.

Código do Servidor (`index.js`)

```
const express = require('express');
const path = require('path');
const app = express();
const port = 3000;

// Serve os arquivos estáticos da pasta "public"
app.use(express.static('public'));

// Variáveis simuladas
let irrigando = false;
let temperatura = 24 + Math.random() * 2;
let umidade = 60 + Math.random() * 5;

// Endpoint para retornar o status atual
app.get('/api/status', (req, res) => {
  res.json({ irrigando, temperatura, umidade });
});

// Endpoint para ligar/desligar a irrigação
app.get('/api/irrigar/:acao', (req, res) => {
  const { acao } = req.params;
  irrigando = (acao === 'ligar');
  res.json({ sucesso: true, irrigando });
});

// Inicia o servidor
app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}`);
});
```

Frontend (`public/index.html`)

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
```

```
  <meta charset="UTF-8">
  <title>Painel de Irrigação</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Sistema de Irrigação</h1>
  <p><strong>Temperatura:</strong> <span id="temp">--</span> °C</p>
  <p><strong>Umidade:</strong> <span id="umid">--</span> %</p>
  <p><strong>Status:</strong> <span id="status">--</span></p>

  <button onclick="acionar('ligar')">Ligar Irrigação</button>
  <button onclick="acionar('desligar')">Desligar Irrigação</button>

  <script>
    async function atualizar() {
      const res = await fetch('/api/status');
      const data = await res.json();
      document.getElementById('temp').innerText =
data.temperatura.toFixed(1);
      document.getElementById('umid').innerText =
data.umidade.toFixed(1);
      document.getElementById('status').innerText = data.irrigando ?
'Ativada' : 'Desativada';
    }

    async function acionar(acao) {
      await fetch(`/api/irrigar/${acao}`);
      atualizar();
    }
    setInterval(atualizar, 3000);
    atualizar();
  </script>
</body>
</html>
```

Estilo Opcional (`public/style.css`)

```
body {
  font-family: sans-serif;
  text-align: center;
  margin-top: 50px;
  background-color: #f2f2f2;
}

button {
  margin: 10px;
  padding: 10px 20px;
  font-size: 16px;
  cursor: pointer;
}
```

Conceitos Envolvidos

- **Node.js + Express:** Cria o servidor web e os endpoints da API.
- **Frontend HTML/JS:** Faz chamadas aos endpoints e atualiza a interface.
- **Simulação de dados:** Gera valores fictícios para testes, sem hardware real.
- **API RESTful:** Define URLs para obter e alterar estados (`/api/status`, `/api/irrigar/ligar`).

Observações

- Quando migrar o projeto para um **Raspberry Pi ou outro hardware**, basta substituir a parte dos dados simulados por leituras reais dos sensores.
- Esse modelo é excelente para **desenvolver e testar a lógica da aplicação** antes da integração com componentes físicos.

o **servidor local** está funcionando, mas com **simulações**. Ou seja, a lógica do servidor foi configurada para **emular** o comportamento de um sistema de irrigação, mas não há hardware real envolvido (como sensores de temperatura/umidade ou GPIOs). Isso significa que ele está **operando corretamente**, mas os valores de temperatura e umidade são gerados aleatoriamente, e a irrigação é controlada através de uma variável simulada, sem interação com componentes físicos.

O que está funcionando no servidor:

- **Servidor Express (Node.js)**: O servidor está rodando no localhost (http://localhost:3000), respondendo às requisições HTTP.
- **Frontend**: O painel de controle no navegador está exibindo os valores simulados de temperatura e umidade, e você pode controlar o estado da irrigação.
- **API de controle**: A API REST funciona, permitindo que você acione o "ligar/desligar" da irrigação.

O que não é real (simulação):

- **Sensores**: A leitura de temperatura e umidade não vem de um dispositivo real (como um DHT11 ou DHT22). Em vez disso, os valores são aleatórios.
- **Controle de irrigação**: A irrigação não está realmente controlando um dispositivo de irrigação. Em vez disso, a mudança do status é apenas simulada na variável `irrigando`.

Para tornar real:

Quando for integrar com hardware real (como um **Raspberry Pi** com sensores de temperatura e umidade, ou pinos GPIO para ligar/desligar um sistema de irrigação), você substituirá as simulações por chamadas reais às bibliotecas e sensores para obter os valores corretos e controlar fisicamente o sistema.

Substituir os valores simulados por leituras reais dos sensores, como:

- `node-dht-sensor` para sensores de temperatura/umidade.
- `onoff` ou `pigpio` para controle de GPIO.