8. AMBIENTE DE EXECUÇÃO JAVASCRIPT SERVER-SIDE (NODE.JS)

O QUE É NODE.JS?

Node.js é um ambiente de execução JavaScript que roda no lado do servidor, baseado no motor V8 (o mesmo usado pelo Google Chrome). Ele permite usar JavaScript fora do navegador para criar aplicações rápidas, escaláveis e orientadas a eventos.

Características principais:

- Assíncrono e orientado a eventos
- Baseado em event loop não bloqueante
- Ideal para aplicações como APIs, chats, sistemas em tempo real, etc.

EXPRESS.JS: FRAMEWORK PARA NODE.JS

Express.js é um framework minimalista para Node.js que simplifica a criação de servidores web e APIs REST. Ele fornece uma estrutura leve e robusta para lidar com rotas, requisições, middlewares e muito mais.

Vantagens do Express:

- Estrutura simples e flexívelSuporte a middlewares (filtros de requisição)
- Rotas organizadas e fáceis de criar
- Suporte a templates (EJS, Pug)
- Integração com bancos de dados (MySQL, MongoDB etc.).

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

1. **Criar um projeto Node.js:**

- mkdir meu-servidor
- cd meu-servidor
- npm init –y

2. **Instalar o Express:**

- npm install express

3. **Instale o mysql**

- npm install mysql

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

4. Crie o banco de dados e a tabela

```
CREATE DATABASE sistema_usuarios;

CREATE TABLE usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(100),
  email VARCHAR(100)
);
```

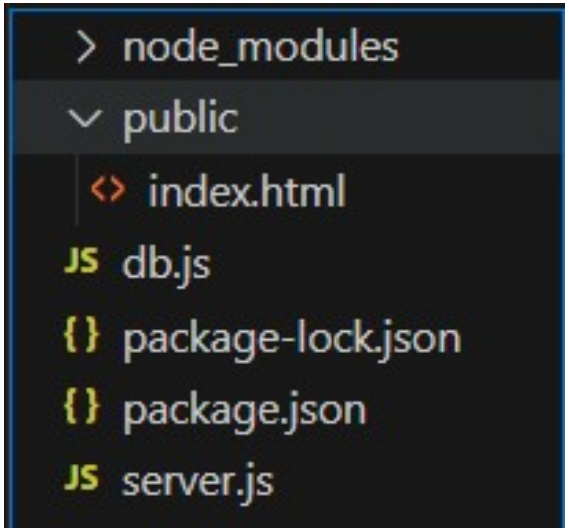
CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

5. Crie os arquivos:

- db.js <-- conexão com MySQL
- server.js <-- API Express

6. Crie a pasta public e crie o arquivo:

- index.html <-- Front end (Formulário)



CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

7. db.js

const mysql = require("mysql"); - Importa o módulo mysql.

const db = mysql.createConnection({...}); - Cria uma conexão com o banco.

db.connect((err) => {...}); - Tenta se conectar ao banco.

module.exports = db; - Exporta a conexão db para que outros arquivos do seu projeto possam utilizá-la (por exemplo, para fazer consultas SQL).

```
const mysql = require("mysql");

const db = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "",
  database: "sistema_usuarios",
});

db.connect((err) => {
  if (err) {
    console.error("Erro ao conectar ao banco:", err);
    return;
  }
  console.log("Conectado ao MySQL!");
});

module.exports = db;
```

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

8. server.js

IMPORTAÇÕES

- **express:** Framework para criar servidores HTTP de forma simples.
- **bodyParser:** Lê o corpo das requisições em JSON.
- **path:** Trabalha com caminhos de arquivos.
- **db:** Conexão com o banco de dados MySQL (arquivo db.js que você criou).

```
const express = require("express");
const bodyParser = require("body-parser");
const path = require("path");
const db = require("./db");

const app = express();
const port = 3000;
```

INICIALIZAÇÃO DO APP

- Cria o app Express e define que o servidor usará a porta 3000.

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

9. Middleware

```
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, "public")));
```

Middleware no Express é uma função que fica entre a requisição (**request**) e a resposta (**response**). Ele intercepta e trata as requisições antes que elas cheguem à rota final ou depois que a resposta é gerada.

app.use(bodyParser.json()); - Permite que o Express entenda e processe JSON no corpo das requisições (como POST e PUT).

app.use(express.static(path.join(dirname, "public"))); - Permite que o Express sirva arquivos estáticos (HTML, CSS, JS, imagens, etc.) da pasta "public".

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

10. CORS

```
app.use((req, res, next) => {
  res.header("Access-Control-Allow-Origin", "*");
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
  next();
});
```

CORS (Cross-Origin Resource Sharing) é um mecanismo de segurança dos navegadores que bloqueia requisições feitas de um site para outro domínio diferente, a menos que o servidor permita isso explicitamente.

res.header("Access-Control-Allow-Origin", "*"); - Permite que qualquer origem (*) acesse sua API.

res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept"); - Permite certos cabeçalhos HTTP personalizados na requisição.

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

11. app.post

```
app.post("/usuarios", (req, res) => {
  const { nome, email } = req.body;
  const sql = "INSERT INTO usuarios (nome, email) VALUES (?, ?)";
  db.query(sql, [nome, email], (err, result) => {
    if (err) return res.status(500).json({ erro: err });
    res.status(201).json({ id: result.insertId, nome, email });
  });
});
```

A rota app.post("/usuarios") serve para cadastrar um novo usuário no banco de dados. Ela:

- Recebe os dados nome e email do corpo da requisição.
- Monta uma query SQL INSERT INTO usuarios.
- Executa a query com os dados recebidos.
- Retorna o novo usuário criado (com o id gerado).

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

12. app.get

```
app.get("/usuarios", (req, res) => {
  db.query("SELECT * FROM usuarios", (err, results) => {
    if (err) return res.status(500).json({ erro: err });
    res.json(results);
  });
});
```

Essa rota app.get("/usuarios") serve para listar todos os usuários do banco de dados. Ela:

- Executa a query SELECT * FROM usuarios.
- Retorna a lista completa de usuários em formato JSON.

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

13.app.get

```
app.get("/usuarios/:id", (req, res) => {
  db.query("SELECT * FROM usuarios WHERE id = ?", [req.params.id], (err, results) => {
    if (err) return res.status(500).json({ erro: err });
    if (results.length === 0) return res.status(404).json({ mensagem: "Usuário não encontrado" })
    res.json(results[0]);
  });
});
```

Essa rota app.get("/usuarios/:id") serve para buscar um usuário específico pelo ID. Ela:

- Pega o id da URL (req.params.id);
- Executa a query SELECT * FROM usuarios WHERE id = ?;
- Retorna os dados do usuário se encontrado, ou erro 404 se não existir.

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

14.app.put

```
app.put("/usuarios/:id", (req, res) => {
  const { nome, email } = req.body;
  const sql = "UPDATE usuarios SET nome = ?, email = ? WHERE id = ?";
  db.query(sql, [nome, email, req.params.id], (err) => {
    if (err) return res.status(500).json({ erro: err });
    res.json({ id: req.params.id, nome, email });
  });
});
```

Essa rota app.put("/usuarios/:id") serve para atualizar um usuário existente. Ela:

- Pega o id da URL e os novos nome e email do corpo da requisição;
- Executa a query UPDATE para atualizar os dados no banco;
- Retorna os dados atualizados como resposta.

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

14.app.delete

```
app.delete("/usuarios/:id", (req, res) => {
  db.query("DELETE FROM usuarios WHERE id = ?", [req.params.id], (err, result) => {
    if (err) return res.status(500).json({ erro: err });
    if (result.affectedRows === 0) return res.status(404).json({ mensagem:
      "Usuário não encontrado" });
    res.json({ mensagem: "Usuário deletado" });
  });
});
```

Essa rota app.delete("/usuarios/:id") serve para deletar um usuário pelo ID. Ela:

- Recebe o id da URL;
- Executa o comando DELETE no banco de dados;
- Se nenhum registro for afetado, responde com erro 404;
- Caso contrário, responde com a mensagem: "Usuário deletado".

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

15.app.listen

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

Esse trecho inicia o servidor Express:

- app.listen(port, () => {...}) faz o servidor escutar na porta 3000.
- Quando o servidor estiver no ar, imprime no console:"Servidor rodando em http://localhost:3000".

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

16. Index.html

```
<body>
  <h1>Gerenciar Usuários</h1>

  <form id="formUsuario">
    <input type="hidden" id="id" />
    <input type="text" id="nome" placeholder="Nome" required />
    <input type="email" id="email" placeholder="Email" required />
    <button type="submit">Salvar</button>
    <button type="button" onclick="limparFormulario()">Cancelar</button>
  </form>

  <h2>Lista de Usuários</h2>
  <table>
    <thead>
      <tr>
        <th>ID</th>
        <th>Nome</th>
        <th>Email</th>
        <th>Ações</th>
      </tr>
    </thead>
    <tbody id="tabelaUsuarios"></tbody>
  </table>
```

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

16. Index.html

Esse código define um evento de envio de formulário que, ao ser acionado, impede o comportamento padrão (recarregar a página). Ele verifica se o ID do usuário existe para decidir entre fazer uma requisição POST (novo usuário) ou PUT (editar um usuário existente). Envia os dados (nome e email) para a API usando fetch, e ao final, recarrega os usuários e limpa o formulário.

```
<script>
  const API_URL = "/usuarios";

  document.getElementById("formUsuario").addEventListener("submit", function (e) {
    e.preventDefault();
    const id = document.getElementById("id").value;
    const nome = document.getElementById("nome").value;
    const email = document.getElementById("email").value;

    const metodo = id ? "PUT" : "POST";
    const url = id ? `${API_URL}/${id}` : API_URL;

    fetch(url, {
      method: metodo,
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ nome, email })
    })
    .then(res => res.json())
    .then(() => {
      carregarUsuarios();
      limparFormulario();
    });
  });
```

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

16. Index.html

```
function carregarUsuarios() {
  fetch(API_URL)
    .then(res => res.json())
    .then(data => {
      const tabela = document.getElementById("tabelaUsuarios");
      tabela.innerHTML = "";
      data.forEach(usuario => {
        tabela.innerHTML += `
          <tr>
            <td>${usuario.id}</td>
            <td>${usuario.nome}</td>
            <td>${usuario.email}</td>
            <td>
              <button onclick="editarUsuario(${usuario.id}, '${usuario.nome}', '${usuario.email}')">Editar</button>
              <button onclick="deletarUsuario(${usuario.id})">Excluir</button>
            </td>
          </tr>
        `;
      });
    });
}
```

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

16. Index.html

A função **carregarUsuarios** faz uma requisição GET para a API e, ao receber a lista de usuários, preenche uma tabela HTML com os dados de cada usuário (ID, nome e email). Para cada usuário, ela também adiciona botões de Editar e Excluir, que chamam as funções correspondentes ao serem clicados.

Requisição para obter os dados: A função faz uma requisição fetch para o API_URL e, ao receber a resposta, converte os dados para JSON.
Limpeza da tabela: Antes de adicionar os novos dados, ela limpa o conteúdo da tabela com **tabela.innerHTML** = "".
Exibição dos dados: Para cada usuário na resposta, ela cria uma linha (<tr>) com as informações do usuário (id, nome e email) e botões para "Editar" e "Excluir".
Atualiza a tabela: As linhas geradas são inseridas na tabela através de tabela.innerHTML +=, preenchendo a tabela com os dados de todos os usuários.

CRIANDO UM SERVIDOR COM NODE.JS E EXPRESS

16. Index.html

Essas três funções controlam ações básicas da interface de usuário para manipulação de registros:

1. **editarUsuario(id, nome, email):** Preenche o formulário com os dados do usuário clicado, permitindo edição.
2. **deletarUsuario(id):** Exibe uma confirmação. Se o usuário confirmar, envia uma requisição DELETE para a API e recarrega a lista de usuários.
3. **limparFormulario():** Limpa todos os campos do formulário, útil para cancelar uma edição ou preparar para um novo cadastro.

```
function editarUsuario(id, nome, email) {
  document.getElementById("id").value = id;
  document.getElementById("nome").value = nome;
  document.getElementById("email").value = email;
}

function deletarUsuario(id) {
  if (confirm("Tem certeza que deseja excluir este usuário?")) {
    fetch(`${API_URL}/${id}`, { method: "DELETE" })
      .then(() => carregarUsuarios());
  }
}

function limparFormulario() {
  document.getElementById("id").value = "";
  document.getElementById("nome").value = "";
  document.getElementById("email").value = "";
}

carregarUsuarios();
</script>
```