Documentação da Aplicação

Sistema de Reservas do Restaurante

Equipe de Desenvolvimento

Nome: Hazael Yuri de Souza Tavares - **Matrícula:** 12724218042

Nome: Iron Cruz de Jesus - **Matrícula:** 12724124750

Nome: Guilherme Calleia do Nascimento - **Matrícula:** 12724117898

1. Requerimentos de Software

Para executar esta aplicação, os seguintes softwares são necessários:

- **Node.js**: Plataforma de execução para o backend.
- **Git (Opcional)**: Necessário para clonar o repositório.
- **Terminal de Linha de Comando**: Essencial para executar os comandos da aplicação.
- **Editor de Código (Recomendado)**: Como o VS Code.

2. Instalação e Execução

Passo 1: Obtenção do Código

Obtenha os arquivos do projeto a partir do GitHub. Você pode fazer isso de duas maneiras:

Opção 1: Clonando o Repositório (Requer Git)

```
git clone URL_DO_SEU_REPOSITORIO_AQUI
```

Opção 2: Baixando o arquivo ZIP

1. Acesse a página do repositório no GitHub.
2. Clique no botão verde **<> Code** e depois em **Download ZIP**.
3. Após o download, extraia o conteúdo para uma pasta no seu computador.

Passo 2: Acesso ao Diretório do Projeto

Abra seu terminal e navegue até a pasta do projeto.

```
cd nome-da-pasta-do-projeto
```

Passo 3: Instalação das Dependências

Execute o comando abaixo para instalar todas as dependências necessárias.

```
npm install
```

> **Passo 4 (Opcional): Inicialização Manual do Banco de Dados**
>
> Caso queira garantir que o banco de dados seja criado do zero, você pode usar um comando de inicialização.
>
> Execute o comando no terminal:
>
> ```
> npm run init-db
> ```

Passo 5: Execução da Aplicação

Com tudo pronto, inicie o servidor com o comando abaixo.

```
npm start
```

Passo 6: Verificação

O terminal exibirá a confirmação e a URL para acessar o sistema.

```
Servidor rodando na porta 3000
Acesse: http://localhost:3000
```

3. Justificativa da Abordagem de Comunicação

A aplicação adota uma arquitetura baseada em API REST utilizando o framework Express.js, comunicando-se com o frontend por meio de requisições HTTP (`fetch`). Essa abordagem foi escolhida por diversos motivos técnicos e funcionais:

Separação de responsabilidades (Frontend e Backend)

A comunicação via API permite que a lógica de negócio e persistência de dados (backend) seja completamente separada da camada de apresentação (frontend). Isso garante maior escalabilidade e facilita manutenções futuras.

Facilidade de integração e expansão

Utilizar endpoints HTTP torna a aplicação facilmente integrável com outros sistemas ou interfaces (como um app mobile, por exemplo), bastando realizar chamadas para as mesmas rotas existentes.

Comunicação assíncrona e responsiva

O uso do `fetch()` no frontend permite chamadas assíncronas ao backend, possibilitando que o sistema exiba mensagens de carregamento e erros sem travar a interface. Isso melhora significativamente a experiência do usuário.

Padronização RESTful

Os métodos HTTP (GET, POST, PUT, DELETE) foram usados conforme os padrões REST, tornando o código mais legível e previsível para outros desenvolvedores. Por exemplo:

- `POST /api/reservas` para criar uma reserva;
- `DELETE /api/reservas/:id` para cancelar;
- `PUT /api/reservas/:id/confirm` para confirmar;
- `GET` para buscar relatórios e listas.

Resposta estruturada e consistente

O backend utiliza uma função de resposta padronizada (`sendResponse`) para retornar sempre um objeto JSON com os campos `ok` , `message` e `data` . Isso simplifica o tratamento das respostas no frontend, pois o padrão é previsível.

Evita recarregamento da página

A comunicação via JavaScript e API elimina a necessidade de recarregar a página a cada operação, tornando o sistema mais fluido e dinâmico.

Segurança e controle de estado

A lógica de validação (como limite de mesas e número de pessoas) está centralizada no servidor, garantindo maior confiabilidade dos dados mesmo em caso de manipulação no frontend.