

# Guia de Implementação

Plataforma de Gestão de Reservas para Restaurantes

## Equipe Responsável

**Nome:** Alana Rosa de Jesus Evangelista dos Reis – **Matrícula:** 12724134591

**Nome:** Lucas Vinicius Medrado de Assis – **Matrícula:** 12724135390

**Nome:** Felipe Damasceno de Santana – **Matrícula:** 12724134520

**Nome:** Augusto Monteiro De Jesus – **Matrícula:** 1272321018

## 1. Pré-requisitos de Ambiente

Para a correta execução da aplicação, certifique-se de que os seguintes componentes estejam instalados em seu sistema:

- Node.js:** Essencial como ambiente de execução do servidor.
- Git (Opcional):** Recomendado para clonar o repositório de forma eficiente.
- Interface de Linha de Comando:** Qualquer terminal, como CMD, PowerShell, etc.
- Editor de Código (Sugestão):** Ferramentas como o VS Code para facilitar a manipulação dos arquivos.

## 2. Configuração e Inicialização

### Etapa 1: Aquisição do Código-Fonte

Existem duas formas principais para obter os arquivos do projeto:

#### Alternativa 1: Clonar via Git

```
git clone URL_DO_SEU_REPOSITORIO_AQUI
```

#### Alternativa 2: Download do Pacote ZIP

- Navegue até a página do projeto no GitHub.
- Clique no botão `<> Code` e selecione a opção `Download ZIP`.
- Após baixar, descompacte o arquivo em um diretório de sua preferência.

### Etapa 2: Acesso ao Diretório Raiz

Utilize seu terminal para navegar até a pasta onde o projeto foi descompactado ou clonado.

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

### Etapa 3: Instalação dos Pacotes

Execute o comando a seguir para instalar todas as dependências necessárias para o projeto.

```
npm install
```

#### Etapa 4 (Opcional): Reinicialização do Banco de Dados

Se desejar começar com um banco de dados limpo, você pode executar este script. Ele apagará o banco de dados atual e criará um novo com os dados padrão.

```
npm run init-db
```

### Etapa 5: Iniciar a Aplicação

Com tudo pronto, inicie o servidor. Use `npm start` para produção ou `npm run dev` para o modo de desenvolvimento.

```
npm start
```

### Etapa 6: Confirmação

O terminal informará que o servidor está ativo e mostrará a URL para acesso local.

```
Servidor em execução na porta 3000
Acesse em: http://localhost:3000
```

## 3. Arquitetura de Comunicação

A aplicação foi projetada sobre o robusto e escalável paradigma **Cliente-Servidor**. Essa abordagem consiste em separar a aplicação em duas partes independentes que colaboram entre si, trazendo organização, segurança e flexibilidade ao projeto.

### Os Dois Pilares: Cliente e Servidor

- Cliente (Frontend):** É a parte da aplicação com a qual o usuário interage diretamente. Pense nela como a "vitrine" de uma loja. Construída com HTML, CSS e JavaScript (na pasta `public`), sua única responsabilidade é apresentar os dados de forma amigável e capturar as ações do usuário (cliques, preenchimento de formulários, etc.).
- Servidor (Backend):** É o "cérebro" e o "cofre" da aplicação (na pasta `server`). Ele é responsável por toda a lógica de negócio, como validar se uma reserva é possível, processar pagamentos, e o mais importante: acessar e gerenciar o banco de dados de forma segura. O cliente nunca toca diretamente no banco de dados.

### A Ponte de Comunicação: API RESTful

Para que o Cliente e o Servidor possam "conversar", eles precisam de um intermediário com regras claras. Esse intermediário é a **API** (Application Programming Interface).

#### Analogia do Restaurante:

Imagine que você (o **Cliente**) está em um restaurante. A cozinha (o **Servidor**) é onde os pratos (os **dados**) são preparados. Você não pode entrar na cozinha para pegar sua comida. Em vez disso, você chama o garçom (a **API**), consulta um cardápio (a **documentação da API**) e faz um pedido claro. O garçom leva seu pedido à cozinha, que o prepara e o devolve a você através do mesmo garçom.

A API funciona como esse "garçom": ela recebe pedidos padronizados do cliente, os leva ao servidor para processamento e retorna com uma resposta.

Nossa aplicação usa uma API do tipo **RESTful**, que é um padrão de mercado amplamente adotado. Ela organiza a comunicação usando os métodos do protocolo HTTP para representar ações:

- GET:** Para solicitar/ler dados (Ex: "listar todas as reservas").
- POST:** Para criar um novo dado (Ex: "Criar uma nova reserva").
- PUT / PATCH:** Para atualizar um dado existente.
- DELETE:** Para remover um dado.

O formato de dados utilizado nessas trocas é o **JSON** (JavaScript Object Notation), um formato leve, legível por humanos e facilmente interpretado por máquinas, tornando-se o idioma universal das APIs modernas.

### Exemplo Prático Detalhado: Criando uma Reserva

- Interação no Cliente:** O atendente preenche os dados da reserva no formulário da página `atendente.html`.
- Requisição via API:** Ao clicar em "Salvar", o JavaScript da página cria um objeto JSON com os dados e envia uma requisição **POST** para o endpoint `/api/reservas` do servidor.
- Processamento no Servidor:** O servidor recebe a requisição. Primeiro, ele valida os dados (o nome está preenchido? o telefone é válido?). Depois, ele executa uma consulta no banco de dados para garantir que a mesa está livre. Se tudo estiver correto, ele insere a nova reserva na tabela correspondente.
- Resposta do Servidor:** O servidor envia de volta uma resposta em JSON. Se a operação foi bem-sucedida, ele retorna algo como `{ "status": "sucesso", "id_reserva": 123 }`. Se houve um erro, retorna `{ "status": "erro", "mensagem": "Mesa já ocupada neste horário" }`.
- Atualização da Interface:** O cliente recebe a resposta. Se for de sucesso, exibe uma mensagem positiva ao usuário e limpa o formulário. Se for um erro, exibe a mensagem de erro para que o usuário possa corrigir.

### Por que essa abordagem é superior?

- Desenvolvimento e Manutenção Simplificados:** Equipes diferentes podem trabalhar no frontend e no backend simultaneamente sem interferir uma na outra. Corrigir um bug visual não quebra a lógica do servidor, e vice-versa.
- Flexibilidade e Evolução:** Como a comunicação é padronizada pela API, podemos facilmente criar novos "clientes" no futuro, como um aplicativo para celular (Android/iOS) ou um painel para tablets, todos consumindo a mesma API e a mesma lógica de negócio, sem reescrever o servidor.
- Segurança e Controle Centralizados:** Toda a lógica sensível e o acesso ao banco de dados estão protegidos no servidor. Isso impede que um usuário mal-intencionado burlie as regras diretamente pelo navegador, garantindo a integridade e a segurança dos dados.