**Relatório**

**INTEGRANTES**

* Hugo Moisés da Silva Carvalho - 12724161831
* Raimundo Egídio Galrão Lima Neto - 12724119913
* João Luiz Nunes Neto -12724114136
* Jorge Eduardo Pinheiro Oliveira Filho -12724126154

**2. Requisitos de Software**

* **Linguagem de Programação**: JavaScript (Node.js) e HTML/CSS/JavaScript no front-end.
* **Ambiente de Tempo de Execução**:  
  + Node.js (versão ≥ 14.x)
  + npm (Node Package Manager)
* **Bibliotecas e Frameworks**
* **Backend:**  
  + express (servidor HTTP)
  + socket.io (comunicação em tempo real)
  + mongoose (ODM para MongoDB)
  + cors (Cross-Origin Resource Sharing)
  + dotenv (carregamento de variáveis de ambiente)
* **Frontend:**  
  Socket.io client (biblioteca /socket.io/socket.io.js)

Font Awesome (ícones)

Google Fonts – Roboto

Custom CSS (arquivo styles.css)

* **Serviço de Banco de Dados**:  
  + MongoDB (instância local ou Atlas)

**3. Instruções de Instalação e Execução**

**Clonar o Repositório** git clone https://github.com/SeuUsuario/reserva-mesas-backend.git

cd reserva-mesas-backend

1. **Instalar Dependências** npm install
2. **Configurar Variáveis de Ambiente**

* Crie um arquivo .env na raiz com:  
   MONGODB\_URI=mongodb://127.0.0.1:27017/reservasdb

PORT=3000

* + Se usar MongoDB Atlas, substitua MONGODB\_URI

**4. Executar o Servidor** npm start

1. **Abrir a Aplicação**
   * No navegador, acesse: http://localhost:3000/

*crie a reserva do cliente com localização do estabelecimento, horario, mesas, quantidade de pessoas. Na aba do garçom para confirmar a liberação da mesa precisa se identificar. Na aba gerente gere todos os relatorios sobre garçons ou mesas, e datas especificas!*

**4. Justificativa da Abordagem de Comunicação**

Optamos por combinar **WebSockets** (via Socket.io) e **APIs REST** pelos seguintes motivos:

1. **Atualização em Tempo Real**: as reservas são sincronizadas instantaneamente entre múltiplos clientes (atendentes e garçons) sem necessidade de recarregar a página.
2. **Eficiência**: Socket.io abstrai reconexões, fallback para polling e faz broadcast de eventos, simplificando a lógica de push.
3. **Relatórios Históricos**: as rotas REST permitem consultas pontuais (intervalos de datas, filtragem por mesa ou garçom), garantindo respostas determinísticas e fáceis de cachear ou registrar logs.
4. **Separação de Camadas**: mantendo estado mutável no front via WebSockets e consultas “snapshot” via REST, equilibramos carga no servidor e clareza de código.