### Documentação para Aplicação Front-end: Sistema de Suporte Técnico

#### 1. Introdução

Esta documentação descreve a aplicação Front-end do **Sistema de Suporte Técnico**, uma plataforma web desenvolvida para gerenciar solicitações de suporte técnico, permitindo que clientes enviem pedidos de ajuda e administradores gerenciem essas solicitações, incluindo chat em tempo real, designação de técnicos e administração de usuários. O objetivo é fornecer uma visão geral das funcionalidades, estrutura, tecnologias utilizadas e instruções para uso e manutenção.

#### 2. Informações Gerais

* **Nome da aplicação**: Sistema de Suporte Técnico
* **Objetivo principal**: Facilitar a comunicação entre clientes e administradores para resolução de problemas técnicos, com funcionalidades de chat, gerenciamento de solicitações e administração de usuários e técnicos.
* **Público-alvo**: Clientes que precisam de suporte técnico e administradores responsáveis por gerenciar solicitações e equipes técnicas.
* **Tecnologias utilizadas**:
  + **HTML5**, **CSS3** e **JavaScript** para estrutura, estilo e lógica Front-end.
  + **Bootstrap 5.3.0** para design responsivo e componentes visuais.
  + **Socket.IO** para comunicação em tempo real (chat).
  + **Fontes externas**: Poppins (via Google Fonts) e Bootstrap Icons para estilização.
  + **UUID** (via biblioteca uuid) para geração de identificadores únicos no chat e solicitações.

#### 3. Instalação e Configuração

Aqui estão os passos para instalar e executar a aplicação localmente:

##### **Pré-requisitos**

* **Node.js** (versão 14 ou superior) e **npm** (gerenciador de pacotes do Node.js).
* Navegador moderno (Chrome, Firefox, Edge).
* Banco de dados SQLite (o arquivo database.db será criado automaticamente).

##### **Comandos de Instalação**

1. Clone o repositório do projeto:

git clone <URL\_DO\_REPOSITORIO>

cd <NOME\_DO\_REPOSITORIO>

1. Instale as dependências do backend:

npm install

As dependências incluem:

* + express: Framework para o servidor.
  + socket.io: Comunicação em tempo real.
  + sqlite3: Banco de dados SQLite.
  + uuid: Geração de identificadores únicos.

##### **Como Iniciar o Projeto**

1. Inicie o servidor:

npm start

O servidor será iniciado na porta padrão 3000 (ou na porta especificada pela variável de ambiente PORT).

1. Acesse a aplicação no navegador:

http://localhost:3000

##### **Configuração de Variáveis de Ambiente**

Não há variáveis de ambiente obrigatórias, mas você pode configurar a porta do servidor:

* Crie um arquivo .env na raiz do projeto:

PORT=3000

* Use uma biblioteca como dotenv para carregar as variáveis (adicione require('dotenv').config() no início do app.js).

#### 4. Estrutura do Projeto

A aplicação segue uma organização simples para o Front-end, localizada na pasta /public:

/public

├── /admin.html (Página do administrador para gerenciar solicitações, técnicos e usuários)

├── /chat.html (Página de chat para comunicação entre cliente e administrador)

├── /index.html (Página inicial para clientes enviarem solicitações)

├── /login.html (Página de login para clientes e administradores)

├── /register.html (Página de registro de novos usuários)

* **/public**: Contém todos os arquivos Front-end estáticos servidos pelo servidor Express.
* **Estilos**: CSS inline dentro de cada arquivo HTML, utilizando Bootstrap para consistência.
* **Scripts**: JavaScript inline nos arquivos HTML, com dependências externas (Bootstrap, Socket.IO).

#### 5. Funcionalidades

A aplicação possui as seguintes funcionalidades principais:

* **Página de Login (login.html)**:  
  Permite que clientes e administradores façam login usando username e senha. Os dados são armazenados no SQLite e validados no backend.
* **Página de Registro (register.html)**:  
  Usuários podem criar uma conta fornecendo username, e-mail e senha. Apenas administradores podem criar outros administradores.
* **Página Inicial (index.html)**:  
  Clientes autenticados podem enviar solicitações de suporte preenchendo um formulário com informações como nome, e-mail, produto e problema. Após envio, são redirecionados para o chat.
* **Chat em Tempo Real (chat.html)**:
  + Clientes podem conversar com um bot ou administrador.
  + O bot responde automaticamente a mensagens simples (ex.: "olá", "roteador não conecta").
  + Clientes podem clicar em "Ainda Preciso de Ajuda" para notificar o administrador.
  + Suporte a sugestões de mensagens e indicador de digitação.
* **Área do Administrador (admin.html)**:
  + **Solicitações Pendentes**: Lista solicitações com opção de responder, atribuir técnicos e agendar visitas.
  + **Chat com Cliente**: Permite que administradores conversem com clientes em tempo real.
  + **Histórico de Chat**: Exibe o histórico de mensagens de uma solicitação.
  + **Adicionar Técnico**: Formulário para cadastrar novos técnicos (nome, e-mail, telefone).
  + **Adicionar Administrador**: Formulário para criar novos administradores (username, e-mail, senha).
  + **Gerenciar Usuários**: Lista usuários com opção de exclusão (exceto o próprio administrador logado).
* **Responsividade**:  
  A aplicação é responsiva, utilizando Bootstrap para layouts adaptáveis a diferentes tamanhos de tela.
* **Integração com APIs**:
  + Comunicação com o backend via rotas REST (/api/auth/login, /api/support, /api/users, etc.).
  + WebSocket (Socket.IO) para chat em tempo real.

#### 6. Documentação de Código

* **Comentários**:  
  O código JavaScript nos arquivos HTML contém comentários inline explicando a lógica principal (ex.: "Renderiza a barra de navegação" ou "Carrega solicitações pendentes").
* **Boas Práticas**:
  + Uso de async/await para chamadas assíncronas, com tratamento de erros via try/catch.
  + Nomenclatura clara para variáveis e funções (ex.: loadUsers, addMessage).
  + Modularidade: Funções separadas para cada funcionalidade (ex.: renderNavbar, loadTechnicians).
* **Ferramentas**: Não foi utilizado JSDoc, mas os comentários são suficientes para manutenção.

#### 7. Testes

Atualmente, a aplicação não possui testes automatizados. No entanto, foram realizados testes manuais:

* **Testes Manuais**:
  + Teste de login e registro com diferentes usuários.
  + Envio de solicitações e verificação do redirecionamento para o chat.
  + Teste do chat entre cliente e administrador, incluindo mensagens automáticas do bot.
  + Teste das abas do administrador (adicionar técnico, administrador, gerenciar usuários).
  + Verificação da responsividade em dispositivos móveis e desktops.
* **Sugestões para Testes Futuros**:
  + Implementar testes unitários com **Jest** para funções JavaScript (ex.: addMessage).
  + Testes de integração para verificar a comunicação com o backend.
  + Testes end-to-end com **Cypress** para simular fluxos completos (ex.: login → envio de solicitação → chat).
  + Comando sugerido para testes futuros:

npm test

#### 8. Hospedagem

A aplicação ainda não foi hospedada em uma plataforma pública, mas pode ser hospedada em serviços como **Netlify**, **Vercel**, ou **Render**. Seguem os passos para um deploy em Vercel:

##### **Passos para Deploy**

1. Certifique-se de que o projeto está em um repositório Git (ex.: GitHub).
2. Instale o CLI da Vercel:

npm install -g vercel

1. Execute o comando de deploy na raiz do projeto:

vercel deploy

1. Siga as instruções do CLI para configurar o projeto (ex.: selecione o tipo de projeto como "Node.js").
2. Após o deploy, a Vercel fornecerá um link de acesso (ex.: https://sistema-suporte-tecnico.vercel.app).

##### **Configuração de Domínio**

* Na Vercel, acesse as configurações do projeto e adicione um domínio personalizado (ex.: suporte-tecnico.com), se desejar.

##### **Link de Acesso**

* Ainda não hospedado, mas o link será fornecido após o deploy.

#### 9. Conclusão

O **Sistema de Suporte Técnico** é uma aplicação funcional que atende às necessidades de clientes e administradores, com um chat em tempo real, gerenciamento de solicitações e administração de usuários e técnicos. Durante o desenvolvimento, aprendemos:

* **Lições Aprendidas**:
  + A importância de gerenciar o estado do WebSocket (Socket.IO) para evitar mensagens duplicadas.
  + A necessidade de tratamento de erros robusto em chamadas assíncronas.
  + Benefícios do uso de Bootstrap para agilizar o design responsivo.
* **Sugestões para Melhorias Futuras**:
  + Adicionar autenticação mais segura (ex.: JWT, criptografia de senhas com bcrypt).
  + Implementar testes automatizados para aumentar a confiabilidade.
  + Melhorar a interface com animações e feedback visual (ex.: loaders durante carregamento).
  + Adicionar suporte a notificações push para alertar administradores sobre novas solicitações.

Esta documentação deve facilitar a manutenção e evolução da aplicação por outros desenvolvedores.