

Universidade Católica Moçambique
Faculdade de Gestão de Turismo e Informática

**Desenvolvimento-de-Página-Web-com-Integração-de-Tecnologias-Web-
Moderna**

Leocênia Hilário Álvaro

Pemba, Maio, 2025

Documentação da Aplicação SoluStyle

1. Introdução

Esta documentação descreve todos os aspectos relevantes da aplicação **SoluStyle**, uma aplicação web desenvolvida para proporcionar interação em tempo real com usuários, gerenciamento de contatos e controle dinâmico de estoque. A aplicação inclui um chat com respostas automáticas e interação com administradores, um formulário de contato, e uma página de visualização de estoque, tudo integrado com um back-end Node.js e WebSocket. O objetivo é detalhar as funcionalidades, estrutura, tecnologias, instalação, uso, testes, hospedagem, e sugerir melhorias futuras, garantindo facilidade de manutenção e escalabilidade.

2. Informações Gerais

- **Nome da Aplicação:** SoluStyle
- **Objetivo Principal:** Fornecer uma plataforma interativa para clientes da SoluStyle, permitindo:
 - Envio de mensagens via chat com limite de 2 perguntas por usuário e respostas automáticas.
 - Interação com administradores após o limite de perguntas.
 - Envio de formulários de contato com salvamento de dados.
 - Visualização e atualização em tempo real do estoque de produtos.
- **Público-Alvo:** Clientes da SoluStyle interessados em suporte via chat, envio de mensagens de contato, e consulta de estoque; administradores que gerenciam interações com clientes.
- **Tecnologias Utilizadas:**
 - **Front-end:**
 - HTML5, CSS3 (Bootstrap 5.3 para estilo **lindo azul**)
 - JavaScript (vanilla, com WebSocket para comunicação em tempo real)
 - **Back-end:**
 - Node.js (v20.x)
 - Express.js (framework para servidor HTTP)
 - WebSocket (via ws para comunicação em tempo real)
 - **Outros:**
 - JSON (dados.json para armazenamento de dados)
 - Render (hospedagem)
 - npm (gerenciador de pacotes)

3. Instalação e Configuração

Pré-requisitos

- **Node.js:** Versão 20.x ou superior (instalar via nodejs.org).
- **npm:** Incluído com o Node.js.
- **Git:** Para clonar o repositório.
- **Sistema Operacional:** Windows, macOS, ou Linux.
- **Navegador:** Chrome, Firefox, ou Edge (com suporte a WebSocket).

➤ Comandos de Instalação

1. Clone o repositório:
2. `git clone https://github.com/seu_usuario/solustyle.git`
3. `cd solustyle`
4. Instale as dependências:
5. `npm install`

➤ Dependências principais (definidas no package.json):

- express
- ws (WebSocket)
- Outras dependências podem incluir cors ou dotenv (se usadas).

➤ Como Iniciar o Projeto

1. Inicie o servidor:
2. `cd server`
3. `node server/server.js > server.log 2>&1/npm start`
4. Acesse no navegador:
 - Chat: `http://localhost:3000/chat.html`
 - Contato: `http://localhost:3000/CONTATO.html`

- Estoque: <http://localhost:3000/stock.html>
- Admin: <http://localhost:3000/admin.html>
- Página principal: <http://localhost:3000/principal.html>

➤ Configuração de Variáveis de Ambiente

- Crie um arquivo `.env` na raiz do projeto (se necessário):
- `PORT=3000`
- O arquivo `.env` é opcional, pois o `server.js` usa a porta 3000 por padrão.
- Certifique-se de que o `dados.json` em `public/` tenha permissões de escrita.

4. Estrutura do Projeto

A estrutura do projeto é organizada para separar o front-end (arquivos estáticos em `public/`) e o back-end (lógica do servidor em `server/`).

/Desenvolvimento-de-Página-Web-com-Integração-de-Tecnologias-Web-Moderna

```

├── .env                # Variáveis de ambiente (opcional)
├── .gitignore          # Arquivos ignorados pelo Git
├── package.json        # Dependências e scripts npm
├── README.md           # Instruções gerais do projeto
├── LICENSE             # Licença do projeto
├── public/             # Arquivos estáticos (front-end)
│   ├── css/           # Estilos CSS
│   │   └── styles.css  # Estilos globais (Bootstrap + customizações azul)
│   ├── js/            # Scripts JavaScript
│   │   ├── app.js      # Lógica geral do front-end
│   │   └── websocket.js # Comunicação WebSocket
│   ├── images/         # Imagens da aplicação
│   ├── contato.html    # Página do formulário de contato
│   ├── chat.html       # Página do chat em tempo real
│   └── stock.html      # Página de visualização de estoque

```

```

|   |—— admin.html      # Painel administrativo
|   |—— principal.html   # Página inicial da aplicação
|   |—— dados.json       # Armazenamento de dados (contato, chat, estoque)
|—— server/              # Lógica do back-end
|   |—— controllers/      # Controladores da API
|   |   |—— apiController.js # Lógica de endpoints
|   |—— routes/           # Rotas da API
|   |   |—— apiRoutes.js   # Definição de rotas
|   |—— sockets/          # Comunicação WebSocket
|   |   |—— socketHandler.js # Lógica do WebSocket
|   |—— config/           # Configurações do servidor
|   |   |—— config.js      # Configurações gerais
|—— server.js             # Ponto de entrada do servidor

```

- **public/:** Contém arquivos acessíveis pelo navegador, incluindo HTML, CSS, JS, e dados.json para armazenamento.
- **server/:** Contém a lógica do back-end, com separação em controladores, rotas, sockets, e configurações.
- **dados.json:** Armazena:
 - contato: Dados de formulários com timestamp (ISO).
 - chat: Mensagens do chat com timestamp (HH:mm).
 - estoque: Produtos com ultimaAtualizacao (ISO).

5. Funcionalidades

A aplicação SoluStyle possui as seguintes funcionalidades:

Chat em Tempo Real

- **Descrição:** Permite que usuários enviem mensagens com limite de 2 perguntas, recebam respostas automáticas, e interajam com administradores após o limite.
- **Detalhes:**
 - Página: chat.html

- Funciona via WebSocket (websocket.js e socketHandler.js).
- Limite de 2 perguntas por usuário, com respostas automáticas para:
 - "Como funciona o estoque?" → Informações sobre produtos.
 - "Como funciona o cadastro de usuários?" → Instruções de cadastro.
- Após 2 perguntas, conecta ao administrador.
- Mensagens salvas no dados.json (chat) com timestamp (ex.: "05:03").
- **Exemplo:**
 - Usuário Lalaia envia: "Como funciona o estoque?"
 - Resposta: "Temos camisetas e calças disponíveis. Veja os detalhes na página de Estoque! (1 pergunta restante)"
 - Administrador responde: "Olá Lalaia, posso ajudar com mais detalhes!"

➤ **Formulário de Contato**

- **Descrição:** Permite que usuários enviem mensagens de contato (nome, email, mensagem).
- **Detalhes:**
 - Página: CONTATO.html
 - Envia dados via POST para /api/CONTATO (server.js).
 - Exibe mensagem de sucesso após envio.
 - Dados salvos no dados.json (contato) com timestamp (ex.: "2025-05-16T03:03:00Z").
- **Exemplo:**
 - Usuário preenche: Nome: Lalaia, Email: lalaia@example.com, Mensagem: Teste
 - Salvamento: { id: 1, nome: "Lalaia", email: "lalaia@example.com", mensagem: "Teste", timestamp: "2025-05-16T03:03:00Z" }

➤ **Gerenciamento de Estoque**

- **Descrição:** Exibe e atualiza o estoque de produtos em tempo real.
- **Detalhes:**
 - Página: stock.html
 - Atualiza quantidade e preço a cada 2 segundos via WebSocket (socketHandler.js).

- Dados salvos no dados.json (estoque) com ultimaAtualizacao (ex.: "2025-05-16T03:03:00Z").
- **Exemplo:**
 - Produto: Camiseta, Quantidade: 45, Preço: 31.25, Última Atualização: "2025-05-16T03:03:00Z"

➤ **Painel Administrativo**

- **Descrição:** Permite que administradores visualizem mensagens dos usuários e respondam.
- **Detalhes:**
 - Página: admin.html
 - Carrega histórico de mensagens via type: 'loadAdminHistory' (socketHandler.js).
 - Envia mensagens para usuários com type: 'admin', exibidas no chat.html como type: 'chat'.
 - Mensagens salvas no dados.json (chat).

Página Principal

- **Descrição:** Página inicial da aplicação, com navegação para outras seções.
- **Detalhes:**
 - Página: principal.html
 - Inclui navbar com links para chat.html, CONTATO.html, stock.html, e admin.html.
 - Estilizada com Bootstrap (azul).

➤ **Responsividade**

- **Descrição:** Interface adaptável a dispositivos móveis e desktops.
- **Detalhes:**
 - Usa Bootstrap 5.3 para layouts responsivos.
 - Testado em Chrome, Firefox, e Edge.

Estilo Visual

- **Descrição:** Interface com tema **lindo azul**.
- **Detalhes:**
 - Navbar e footer em azul (#007bff).
 - Fundo claro (#e6f3ff) para melhor legibilidade.
 - Estilos definidos em public/css/styles.css.

➤ Documentação de Código

- **Comentários:**
 - O código contém comentários explicativos em pontos críticos, como:
 - socketHandler.js: Comentários nas funções readDados, writeDados, e blocos de envio de mensagens.
 - server.js: Comentários na configuração do servidor e rotas.
 - websocket.js: Comentários nos eventos WebSocket (onopen, onmessage).
 - Exemplo:
 - // Enviar mensagem do usuário para todos os clientes
 - wss.clients.forEach(client => {
 - if (client.readyState === client.OPEN) {
 - client.send(JSON.stringify({
 - type: adminClients.has(client) ? 'admin' : 'chat',
 - mensagem: novaMensagem.mensagem,
 - sender: 'user',
 - timestamp,
 - id: novaMensagem.id,
 - replyTo
 - }));
 - }
 - });

➤ Boas Práticas:

- Uso de async/await para operações assíncronas (fs.readFile, fs.writeFile).
- Tratamento de erros com try-catch em todos os blocos assíncronos.
- Normalização de texto (normalizeText) para busca insensível a acentos.
- Logs detalhados para depuração (console.log em socketHandler.js).

➤ **Ferramentas:**

- JSDoc não foi usado, mas os comentários seguem um formato claro e descritivo.
- Recomenda-se adotar JSDoc para documentação formal em futuras iterações.

7. Testes

➤ **Tipos de Testes:**

○ **Testes Manuais:**

- Chat: Envio de mensagens, verificação de respostas automáticas, limite de perguntas, interação com administrador.
- Contato: Envio de formulário, verificação de mensagem de sucesso, salvamento no dados.json.
- Estoque: Atualização de quantidade e preço, salvamento com ultimaAtualizacao.
- Admin: Exibição de mensagens dos usuários, envio de respostas.

➤ **Testes de Integração:**

- Verificação da comunicação WebSocket entre chat.html, admin.html, e socketHandler.js.
- Teste da rota /api/CONTATO com Postman ou navegador.

○ **Testes Unitários/End-to-End:** Não implementados.

➤ **Ferramentas Utilizadas:**

- Navegador (Chrome DevTools) para logs do console.
- Terminal para logs do servidor (server.log).

➤ **Como Executar Testes:**

- Testes manuais:
 1. Inicie o servidor: `node server/server.js > server.log 2>&1`
 2. Acesse `http://localhost:3000/chat.html` e envie mensagens.
 3. Acesse `http://localhost:3000/admin.html` e responda.
 4. Acesse `http://localhost:3000/CONTATO.html` e envie formulário.
 5. Verifique `public/dados.json` e `server.log`.
- Testes automatizados: Não implementados (recomenda-se Jest para futuras iterações).

➤ **Resultados:**

- Problemas recentes (mensagens não enviadas/exibidas) foram corrigidos com a última versão do `socketHandler.js` (`artifact_id: 07b2f32e-e09b-4ba5-bc47-f265a0eb863c`, `version_id: 9e2e0a2b-9e6c-4f28-b9d7-4c8f2d8b3e2f`).
- Salvamento no `dados.json` confirmado para chat, contato, e estoque.

8. Hospedagem

- **Plataforma:** Render (escolhida para hospedagem do projeto).
- **Passos para Deploy:**
 1. Crie uma conta no Render.
 2. Crie um novo **Web Service** no painel do Render.
 3. Conecte ao repositório Git (ex.: GitHub).
 4. Configure o serviço:
 - **Runtime:** Node
 - **Build Command:** `npm install`

- **Start Command:** node server/server.js
- **Environment Variables:**
- PORT=3000

5. Faça o deploy:

- Clique em "Deploy" no Render.
- O Render atribui uma URL (ex.: <https://solustyle.onrender.com>).

6. Acesse a aplicação:

- Chat: <https://solustyle.onrender.com/chat.html>
- Contato: <https://solustyle.onrender.com/CONTATO.html>
- Estoque: <https://solustyle.onrender.com/stock.html>
- Admin: <https://solustyle.onrender.com/admin.html>
- Principal: <https://solustyle.onrender.com/principal.html>

• Configuração de Domínio:

- Não configurado. Para adicionar um domínio personalizado (ex.: www.solustyle.com), siga a documentação do Render para DNS.

• Link de Acesso:

- Após o deploy, o Render fornecerá a URL (ex.: <https://solustyle.onrender.com>).
- Como o deploy ainda não foi realizado, acesse localmente em
- <http://localhost:3000>.

9. Conclusão

A aplicação **SoluStyle** é uma solução robusta para interação em tempo real, gerenciamento de contatos, e controle de estoque, com uma interface amigável e estilo **lindo azul**. Durante o desenvolvimento, foram superados desafios como:

- Falhas no envio/exibição de mensagens no chat, corrigidas com simplificação da lógica no `socketHandler.js`.
- Problemas de salvamento no `dados.json`, resolvidos com verificação de permissões e logs detalhados.
- Integração do WebSocket para comunicação em tempo real.

Lições Aprendidas:

- A importância de logs detalhados para depuração.
- A necessidade de testes automatizados para evitar regressões.
- A simplificação da lógica de envio de mensagens melhora a confiabilidade.

Sugestões para Melhorias Futuras:

- Implementar testes unitários e end-to-end com Jest e Cypress.
- Adicionar autenticação para o painel administrativo (`admin.html`).
- Migrar o armazenamento de `dados.json` para um banco de dados (ex.: MongoDB).
- Melhorar a responsividade com testes em dispositivos móveis.
- Adicionar notificações em tempo real para novos contatos.

Esta documentação consolida todas as funcionalidades implementadas, garantindo que a aplicação seja facilmente compreendida, mantida, e escalada no futuro.

1. Link para acessar meu site no Render:

<https://desenvolvimento-de-p-gina-web-com.onrender.com>

2. Link para acessar meu projeto no Github:

<https://github.com/LeoceniaDaGloria/Desenvolvimento-de-P-gina-Web-com-Integra-o-de-Tecnologias-Web-Moderna>