**Documentação para Aplicação Front-end: Biblioteca Geek**

**1. Introdução**

Esta documentação descreve os aspectos da aplicação front-end do projeto **Biblioteca Geek**, uma plataforma para gerenciamento de livros, aluguel e interação entre usuários e administradores. O objetivo é fornecer uma visão detalhada das funcionalidades, estrutura, tecnologias utilizadas, e instruções para uso e manutenção.

**2. Informações Gerais**

* **Nome da aplicação**: Biblioteca Geek
* **Objetivo principal**: Permitir que usuários cadastrem-se, façam login, consultem e aluguem livros, enquanto administradores gerenciam o acervo, usuários e interagem via chat.
* **Público-alvo**: Leitores interessados em livros de ficção científica, fantasia e quadrinhos, e administradores de bibliotecas.
* **Tecnologias utilizadas**:
  + HTML5, CSS3 (Bootstrap 5.3 para estilização), JavaScript (ES6+)
  + WebSocket para chat em tempo real
  + Fetch API para comunicação com o back-end
  + LocalStorage para gerenciamento de estado de login

**3. Instalação e Configuração**

**Pré-requisitos**

* Node.js (versão 18 ou superior) e npm instalados
* Navegador moderno (Chrome, Firefox, etc.)
* Servidor back-end configurado (consulte server.js)

**Comandos de Instalação**

1. Clone o repositório ou copie os arquivos para um diretório local:
2. git clone <url-do-repositorio>

cd biblioteca-geek

1. Instale as dependências do back-end (necessárias para o funcionamento do front-end):

npm install express ws multer

1. Certifique-se de que os arquivos front-end estão na pasta public:
2. /public
3. ├── index.html
4. ├── catalogo.html
5. ├── admin.html
6. ├── chat.html
7. ├── /scripts
8. │ ├── busca.js
9. │ ├── catalogo.js
10. │ ├── chat.js
11. │ ├── admin.js
12. ├── /estilo.css

└── /uploads (para imagens dos livros)

**Como Iniciar o Projeto**

1. Inicie o servidor back-end:

node server.js

O servidor estará rodando em http://localhost:3000.

1. Acesse o front-end diretamente pelo navegador:
   * Página inicial: http://localhost:3000/index.html
   * Catálogo: http://localhost:3000/catalogo.html
   * Administração: http://localhost:3000/admin.html
   * Chat: http://localhost:3000/chat.html

**Configuração de Variáveis de Ambiente**

Não são necessárias variáveis de ambiente para o front-end. O back-end espera que os arquivos JSON (livros.json, usuarios.json, admins.json) estejam no diretório raiz do projeto.

**4. Estrutura do Projeto**

A estrutura do front-end está organizada na pasta public: /public  
├── index.html (página inicial com busca de livros)  
├── catalogo.html (página de catálogo com opção de aluguel)  
├── admin.html (página de administração para gerenciar livros, usuários e chat)  
├── chat.html (página de chat para usuários)  
├── /scripts  
│ ├── busca.js (lógica de busca e exibição inicial de livros)  
│ ├── catalogo.js (lógica de catálogo e aluguel)  
│ ├── admin.js (lógica de administração)  
│ ├── chat.js (lógica de chat com WebSocket)  
├── /estilo.css (estilos personalizados)  
└── /uploads (armazenamento de imagens dos livros)

**5. Funcionalidades**

**Página Inicial (index.html)**

* **Busca de Livros**: Permite buscar livros por título usando uma barra de busca.
* **Exibição de Livros**: Mostra uma lista de livros com detalhes (título, autor, quantidade, status, descrição, imagem).
* **Login/Cadastro**: Modal para login ou cadastro de usuários, salvando o estado no localStorage.

**Catálogo (catalogo.html)**

* **Listagem de Livros**: Exibe todos os livros disponíveis com detalhes.
* **Aluguel de Livros**: Usuários logados podem alugar livros disponíveis (botão "Alugar" aparece após login). Após o aluguel, o status muda para "Alugado".
* **Status de Login**: Mostra o e-mail do usuário logado e um botão para logout.

**Administração (admin.html)**

* **Gerenciamento de Livros**: Permite adicionar, editar, deletar livros, aumentar/reduzir quantidade e definir datas de chegada.
* **Gerenciamento de Usuários e Administradores**: Lista, edita e deleta usuários e administradores.
* **Chat com Usuários**: Interface para administradores conversarem com usuários via WebSocket.
* **Estatísticas**: Exibe estatísticas sobre livros, usuários e aluguéis, com gráficos (usando Chart.js).

**Chat (chat.html)**

* **Interação com Bot e Administrador**: Usuários podem conversar com um bot (5 mensagens) e depois são conectados a um administrador via WebSocket.
* **Autenticação**: Requer login para iniciar o chat.

**Responsividade**

* Uso do Bootstrap 5.3 para layouts responsivos, garantindo boa visualização em dispositivos móveis e desktops.

**Integração com APIs**

* Comunicação com o back-end via Fetch API para endpoints como /livros, /alugar, /usuarios, /admins, e /busca.
* WebSocket (ws://localhost:3000) para chat em tempo real.

**6. Documentação de Código**

* O código contém comentários inline para explicar funcionalidades principais.
* Logs de console foram adicionados em pontos críticos (ex.: requisições Fetch, eventos de WebSocket) para facilitar depuração.
* Boas práticas:
  + Uso de async/await para chamadas assíncronas.
  + Sanitização de entradas no back-end para prevenir XSS (embora o front-end passe os dados diretamente).
  + Modularização das funções em arquivos separados (busca.js, catalogo.js, etc.).

**7. Testes**

* **Tipos de Testes**:
* Testes manuais foram realizados:
  + Teste de login/cadastro e verificação da exibição do botão "Alugar".
  + Teste de aluguel e mudança de status para "Alugado".
  + Teste de responsividade em diferentes tamanhos de tela.
  + Teste do chat com WebSocket (bot e administrador).
* **Ferramentas Utilizadas**: Testes manuais via console do navegador e logs no terminal.
* **Como Executar Testes**: Acesse as páginas no navegador e siga os fluxos de uso (login, aluguel, chat, administração).

**8. Hospedagem**

O site foi hospedado no Render

**9. Conclusão**

A aplicação Biblioteca Geek oferece uma interface funcional para usuários e administradores gerenciarem livros e interagirem via chat. Durante o desenvolvimento, foram aprendidas boas práticas de integração front-end/back-end e uso de WebSocket para comunicação em tempo real. Sugestões de melhorias incluem:

* Implementar testes automatizados (ex.: Jest para unitários).
* Adicionar mais validações no front-end (ex.: impedir aluguel duplicado).
* Melhorar a responsividade com mais testes em dispositivos móveis.
* Hospedar a aplicação em uma plataforma pública para acesso remoto.