Universidade Catolica De Moçambique

Faculdade de Gestao de Turismos e informatica

Documentação de Estudos de língua Portuguesa

Nome: Pascoal Bernardo Sualehe

1. Introdução

Esta documentação descreve a aplicação frontend da plataforma "Estudo de Português", detalhando suas funcionalidades, estrutura, tecnologias utilizadas, instruções de instalação, configuração, testes, hospedagem e sugestões de manutenção. A aplicação foi desenvolvida para fornecer uma interface amigável para estudantes, professores e administradores aprenderem e gerenciarem conteúdos de português.

2. Informações Gerais

Nome da Aplicação: Estudo de Português

Objetivo Principal: Oferecer uma plataforma de aprendizado de português com materiais interativos (vídeos, áudios, PDFs), chat para resolução de dúvidas e painel administrativo para gerenciamento de usuários e conteúdos.

Público-Alvo: Estudantes (níveis inicial, intermédio e avançado), professores e administradores.

Tecnologias Utilizadas:

HTML5: Estrutura das páginas.

CSS3: Estilização, com variáveis CSS, animações e design responsivo.

JavaScript (não fornecido, mas implícito): Integração com a API backend e Socket.IO para funcionalidades dinâmicas (chat, listagens, autenticação).

Socket.IO Client: Comunicação em tempo real com o backend.

Bootstrap (implícito em admin.html, chat.html, etc.): Componentes de interface como modais e formulários.

3. Instalação e Configuração

Pré-requisitos:

- Node.js (v16 ou superior) e npm (v8 ou superior) para executar o backend.
- Navegador moderno (Chrome, Firefox, Edge).
- Conexão com o backend hospedado ou rodando localmente.

Comandos de Instalação:

Clone o repositório do projeto:

- git clone <URL_DO_REPOSITORIO>
- cd Estudo

Instale as dependências do backend (necessário para servir os arquivos estáticos):

- cd server
- npm install

Como Iniciar o Projeto:

- Inicie o servidor backend:
- npm start

O servidor será executado em http://localhost:3000, servindo os arquivos da pasta public

Acesse a aplicação no navegador em http://localhost:3000.

Configuração de Variáveis de Ambiente

- Não há variáveis de ambiente explícitas no projeto atual.
- Caso o backend seja hospedado separadamente (ex.: Render), configure a URL do backend no JavaScript do frontend (ex.: substitua http://localhost:3000 pela URL do Render no cliente Socket.IO).

4. Estrutura do Projeto

A aplicação frontend está localizada na pasta public, com a seguinte organização:

/Estudo

/Public

-admin.html -- Painel administrativo

-chat.html--- Página de chat para dúvidas

-estilo.html -- Estilos globais (variáveis, responsividade, animações)

-index.html -- Página inicial

-lessons.html -- Seleção de tópicos por nível

-login.html -- Autenticação e cadastro de usuários

-materials.html -- Exibição de materiais e chat

/Server

-app.js -- Servidor Express e Socket.IO

-data.json -- Armazenamento de dados

-routes.js -- Rotas da API

Na pasta raiz (Estudo)

-package.json -- Configuração do projeto

-package-lock.json -- Dependências exatas

public/: Contém os arquivos HTML e CSS que formam a interface.

server/: Backend que serve os arquivos estáticos e gerencia a lógica.

5. Funcionalidades

A aplicação frontend possui as seguintes funcionalidades:

Página Inicial (index.html)

Apresenta a plataforma com uma mensagem de boas-vindas.

Links de navegação para Início, Materiais, Admin e Login.

Call-to-action para login ou cadastro.

Autenticação e Cadastro (login.html)

Login: Formulário para nome de usuário e senha.

Cadastro: Formulário para nome de usuário, senha, tipo (estudante/professor), nível (inicial/intermédio/avançado) e localização.

Integração com a API (/api/login, /api/users) para autenticação e registro.

Materiais (materials.html)

Exibe vídeos, áudios e PDFs, organizados por tipo.

Filtros implícitos por nível e tópico (via API /api/materials?level=<nivel>&topic=<topico>).

Chat de dúvidas integrado, com envio de mensagens via Socket.IO.

Modal de feedback para interação com o usuário.

Tópicos (lessons.html)

Lista tópicos divididos por nível (inicial, intermédio, avançado).

Integração com a API para carregar tópicos dinamicamente.

Modal de feedback.

Chat de Dúvidas (chat.html)

Interface dedicada para chat relacionado a um material específico.

Envio de mensagens via Socket.IO (userMessage).

Exibição de respostas administrativas em tempo real.

Painel Administrativo (admin.html)

Autenticação de Admin: Campo para chave de administrador (validação via /api/admin/auth).

Gerenciamento de Usuários: Adição, edição e exclusão de usuários (API /api/users).

Gerenciamento de Administradores: Adição e exclusão de administradores (API /api/admins).

Usuários Online: Monitoramento em tempo real via Socket.IO (userStatus).Monitoramento do Sistema: Logs de atividades (ex.: logins, acessos) via Socket.IO (systemActivity).

Conversas do Chat: Visualização e resposta a dúvidas via Socket.IO (adminMessage).

Relatórios:

Temas com mais dúvidas (API /api/reports/doubts).

Utilização da plataforma (acessos por dia, tempo médio de sessão, materiais mais visualizados) via /api/reports/usage.

Responsividade

Design responsivo implementado em estilo.css com media queries para telas de 1024px, 768px e 480px.

Ajustes em fontes, padding e layout para dispositivos móveis e tablets.

Integração com APIs

Consumo de endpoints REST (/api/users, /api/materials, /api/doubts, etc.) para dados dinâmicos.

Comunicação em tempo real via Socket.IO para chat, notificações e monitoramento.

6. Documentação de Código

Comentários: O arquivo estilo.css inclui comentários explicativos para seções (ex.: /* Navbar */, /* Cards */) e variáveis CSS.

Boas Práticas:

Uso de variáveis CSS (--primary, --secondary) para consistência.

Estrutura modular no CSS com animações e transições suaves.

HTML semântico com classes descritivas (ex.: .btn-primary, .card).

JavaScript: Não fornecido, mas uso de funções comentadas para integração com API e Socket.IO. Recomenda-se adotar JSDoc para futuras implementações.

7. Testes

Testes Realizados: Não há testes automatizados implementados no frontend atual.

Testes Manuais:

Navegação entre páginas (index.html, login.html, etc.).

Submissão de formulários (login, cadastro, chat).

Responsividade em diferentes dispositivos (desktop, tablet, celular).

Integração com backend (carregamento de materiais, envio de mensagens).

Recomendações:

Implementar testes unitários com Jest para funções JavaScript.

Usar Cypress para testes end-to-end (ex.: fluxo de login, envio de mensagens).

8. Hospedagem

A aplicação será hospedada no Render, uma plataforma de hospedagem que suporta aplicações Node.js e arquivos estáticos.

Passos para o Deploy

Preparação:

Certifique-se de que o repositório está no GitHub, GitLab ou Bitbucket.

Crie um arquivo .gitignore para excluir node_modules e arquivos sensíveis.

Configuração no Render:

Acesse render.com e crie uma conta.

Crie um novo Web Service e conecte ao repositório do projeto.

Configure as opções:

Runtime: Node.

Build Command: npm install.

Start Command: npm start.

Pasta Raiz: / (ou ajuste se o projeto estiver em uma subpasta).

Adicione variáveis de ambiente (se necessário, ex.: PORT=3000).

Deploy:

Clique em "Deploy" no Render. O serviço construirá e executará o servidor.

O Render servirá os arquivos estáticos da pasta public e o backend em uma única URL (ex.: https://estudo-portugues.onrender.com).

Configuração de Domínio (opcional):

Adicione um domínio personalizado no Render (ex.: www.estudoportugues.com).

Configure registros DNS no provedor de domínio para apontar para o Render.

Link de Acesso:

Após o deploy, o Render fornecerá uma URL (ex.: https://estudo-portugues.onrender.com).

Acesse a aplicação diretamente no navegador.

Comandos Usados

Build: npm install

Start: npm start

9. Conclusão

A aplicação frontend "Estudo de Português" oferece uma interface intuitiva e responsiva para aprendizado de português, com funcionalidades como acesso a materiais, chat de dúvidas e painel administrativo robusto. A integração com o backend via Express e Socket.IO garante dinamismo e interatividade.

Design Responsivo: A importância de media queries para garantir acessibilidade em dispositivos variados.

Integração Frontend-Backend: A necessidade de sincronizar chamadas à API e eventos Socket.IO para uma experiência fluida.

Documentação: A relevância de manter uma documentação clara para facilitar manutenção e colaboração.