Universidade Catolica De Moçambique e Faculdade de Gestao de Turismos e informatica

Documentação da Aplicação Front-end - Agenda de Contatos

Jose Maria Dos Santos Celso-706230146

Documentação do Projeto Agenda de Contatos

1. Introdução

Esta documentação descreve todos os aspectos relevantes da aplicação "Agenda de Contatos", uma aplicação web full-stack desenvolvida para gerenciar contatos e fornecer funcionalidades de autenticação e chat em tempo real. O objetivo é detalhar as funcionalidades, estrutura, tecnologias utilizadas, instruções de uso e manutenção, além de fornecer um guia para desenvolvedores que desejem contribuir ou manter o projeto. A aplicação foi projetada para ser simples, funcional e escalável, com foco em uma experiência de usuário fluida.

2. Informações Gerais

- **Nome da aplicação**: Agenda de Contatos
- **Objetivo principal**: Gerenciar contatos de forma segura, permitindo autenticação de usuários, armazenamento de informações em um banco de dados JSON, e comunicação em tempo real por meio de um chat.
- **Público-alvo**: Usuários individuais ou pequenas equipes que precisam de uma solução simples para organizar contatos e interagir em tempo real, como em cenários de suporte ou colaboração.
- **Tecnologias utilizadas**:
  - **Back-end**: Node.js (v22.15.0), Express (v4.19.2), WebSocket (ws v8.18.0), JSON Web Token (jsonwebtoken v9.0.2)
  - **Front-end**: HTML, CSS, JavaScript
  - **Gerenciamento de dependências**: npm
  - **Desenvolvimento**: Nodemon (v3.1.7) para reinicialização automática do servidor
  - **Armazenamento**: Arquivo JSON (`database.json`) como banco de dados leve

3. Instalação e Configuração

Pré-requisitos

- **Node.js**: Versão 22.15.0 ou superior instalada.
- **npm**: Incluído com o Node.js, usado para gerenciar dependências.
- **Sistema operacional**: Windows, macOS ou Linux.
- **Navegador web**: Qualquer navegador moderno (Chrome, Firefox, Edge, etc.).

- **Git** (opcional): Para clonar o repositório, se hospedado em um controle de versão.

Comandos de Instalação

1. Clone o repositório (se aplicável) ou navegue até o diretório do projeto:
2. `cd C:\Users\Jose Santos\Desktop\Yummy\ProjetoWeb`
3. Instale as dependências listadas em `package.json`:
4. `npm install`

Como Iniciar o Projeto

- Execute o servidor em modo de desenvolvimento com Nodemon:
- `npm start`

  Isso inicia o servidor definido em `server/app.js`, que estará acessível em `http://localhost:3000` (ou outra porta configurada).

Configuração de Variáveis de Ambiente

- Atualmente, não há variáveis de ambiente configuradas explicitamente. Caso seja necessário adicionar, crie um arquivo `.env` na raiz do projeto e configure variáveis como:
- `PORT=3000`
- `JWT_SECRET=sua_chave_secreta`
- Use a biblioteca `dotenv` (não incluída atualmente) para carregar essas variáveis em `app.js`:
- `require('dotenv').config();`

4. Estrutura do Projeto

A estrutura do projeto é organizada para separar o back-end (servidor Node.js) do front-end (páginas HTML e assets). Abaixo está a organização dos arquivos e pastas conhecidos:

```
C:\Users\Jose Santos\Desktop\Yummy\ProjetoWeb
├── /server
│   ├── app.js          # Configuração do servidor Express e WebSocket
│   ├── routes.js       # Definição das rotas da API (login, contatos,
etc.)
├── /public             # (Suposto, com base em práticas comuns)
│   ├── index.html      # Página principal da aplicação
│   ├── chat.html       # Página para funcionalidade de chat
│   ├── /css            # Estilos CSS
│   ├── /js             # Scripts JavaScript do front-end
├── database.json       # Banco de dados JSON para armazenar contatos e
usuários
├── package.json        # Configuração do projeto e dependências
├── package-lock.json   # Versões exatas das dependências
```

Descrição dos Arquivos

- **`server/app.js`**: Configura o servidor Express, serve arquivos estáticos (como `index.html` e `chat.html`), e inicializa o WebSocket para o chat.
- **`server/routes.js`**: Define as rotas da API, incluindo endpoints para autenticação (`/login`), gerenciamento de contatos (`/contacts`), e middleware para verificar tokens JWT.
- **`public/index.html`**: Página inicial da aplicação, provavelmente com um formulário de login e listagem de contatos.
- **`public/chat.html`**: Interface para o chat em tempo real, conectada ao WebSocket.
- **`database.json`**: Armazena dados de usuários e contatos em formato JSON, usado como um banco de dados leve.
- **`package.json`** e **`package-lock.json`**: Gerenciam dependências e configurações do projeto.

5. Funcionalidades

A aplicação possui as seguintes funcionalidades principais:

- **Página de Login**:
  - Permite que usuários façam login com credenciais (e.g., nome de usuário e senha).
  - Usa `jsonwebtoken` para gerar tokens JWT, autenticando solicitações subsequentes.
  - Implementada em `routes.js` com o endpoint `/login`.
- **Gerenciamento de Contatos**:
  - Listagem, adição, edição e exclusão de contatos, armazenados em `database.json`.
  - Acessível via endpoints em `routes.js` (e.g., `GET /contacts`, `POST /contacts`).
  - Protegido por autenticação JWT, garantindo que apenas usuários logados acessem.
- **Chat em Tempo Real**:
  - Interface de chat em `chat.html`, permitindo comunicação instantânea entre usuários.
  - Implementada com WebSocket (`ws`) em `app.js`, suportando mensagens bidirecionais.
- **Responsividade**:
  - As páginas `index.html` e `chat.html` são projetadas com CSS para serem responsivas, adaptando-se a dispositivos móveis e desktops (suposto com base em práticas padrão).
- **Integração com APIs**:
  - A API interna, definida em `routes.js`, fornece endpoints RESTful para o front-end interagir com o back-end.
  - Exemplo: `GET /contacts` retorna a lista de contatos em formato JSON.

6. Testes

Atualmente, não há informações sobre testes implementados no projeto. Abaixo está uma sugestão para testes que poderiam ser adicionados:

- **Tipos de Testes**:
  - **Unitários**: Testar funções específicas em `routes.js`, como validação de tokens JWT.
  - **Integração**: Verificar a integração entre o front-end (`index.html`, `chat.html`) e os endpoints da API.
  - **End-to-End**: Simular o fluxo completo do usuário (login, adicionar contato, enviar mensagem no chat).
- **Ferramentas Sugeridas**:
  - **Jest**: Para testes unitários e de integração no back-end.
  - **Supertest**: Para testar endpoints HTTP em `routes.js`.
  - **Cypress**: Para testes end-to-end no front-end.
- **Como Executar Testes** (se implementados):
- `npm test`

Recomenda-se adicionar testes para aumentar a robustez do projeto, especialmente para autenticação e manipulação de dados em `database.json`.

7. Hospedagem

Atualmente, não há informações confirmadas sobre a hospedagem da aplicação. Supõe-se que o projeto está em desenvolvimento local (`C:\Users\Jose Santos\Desktop\Yummy\ProjetoWeb`). Para hospedagem, as seguintes opções são recomendadas:

- **Plataformas Sugeridas**:
  - **Render** ou **Heroku**: Para hospedar o back-end Node.js.
  - **Netlify** ou **Vercel**: Para servir os arquivos estáticos do front-end (`public`).
  - **GitHub Pages**: Para hospedagem estática simples, se o front-end for independente.
- **Passos para Deploy (exemplo com Render)**:
  1. Crie uma conta no Render e configure um novo serviço web.
  2. Conecte o repositório Git do projeto (se disponível).
  3. Configure o comando de inicialização:
  4. `npm install && npm start`
  5. Defina variáveis de ambiente (e.g., `PORT`, `JWT_SECRET`) no painel do Render.
  6. Faça o deploy e obtenha o link da aplicação (e.g., `https://agenda-de-contatos.onrender.com`).
- **Link de Acesso**: Não disponível, pois o projeto parece estar em desenvolvimento local.

8. Conclusão

A aplicação "Agenda de Contatos" é uma solução full-stack eficaz para gerenciar contatos e oferecer comunicação em tempo real. Com um back-end robusto baseado em Node.js, Express e WebSocket, e um front-end simples em HTML/CSS/JavaScript, o projeto atende às necessidades de usuários que buscam uma ferramenta prática e segura. Durante o desenvolvimento, foram utilizados padrões modernos de desenvolvimento web, como autenticação JWT e comunicação via WebSocket, garantindo segurança e interatividade.

Lições Aprendidas

- A integração de WebSocket com Express requer cuidado na configuração para evitar conflitos de portas.
- O uso de `database.json` como banco de dados é prático para protótipos, mas pode não escalar para grandes volumes de dados.
- A ausência de testes automatizados destaca a importância de planejar testes desde o início.

Sugestões para Melhorias Futuras

- Migrar o armazenamento de `database.json` para um banco de dados relacional (e.g., PostgreSQL) ou NoSQL (e.g., MongoDB) para maior escalabilidade.
- Implementar testes unitários e end-to-end para aumentar a confiabilidade.
- Adicionar mais recursos ao front-end, como um framework (e.g., React) para melhorar a interatividade.
- Configurar CI/CD para deploy automatizado em plataformas como Render ou Vercel.

Esta documentação será atualizada conforme o projeto evolui, garantindo que todas as mudanças sejam devidamente registradas.