Documentação da Aplicação Front-end: Educação Financeira

Tobias Alberto Paulo Nambane – 706230204

706230204@ucm.ac.mz

1. Introdução

Esta documentação descreve a aplicação Front-end do sistema de Educação Financeira, uma plataforma web desenvolvida para auxiliar usuários na gestão de suas finanças pessoais, oferecendo ferramentas como cálculo de poupança, relatórios financeiros, chat com bot e administrador, e um painel administrativo para gerenciamento de usuários e dúvidas. A aplicação foi projetada para ser intuitiva, responsiva e funcional, com integração com um backend Node.js hospedado no Render. Este documento cobre as funcionalidades, estrutura, tecnologias, instalação, hospedagem e manutenção da aplicação.

2. Informações Gerais

- **Titulo do Projeto**: Desenvolvimento de Página Web com Integração de Tecnologias Web Modernas
- **Nome da aplicação**: Educação Financeira
- **Objetivo principal**: Fornecer uma plataforma acessível para educação financeira, permitindo aos usuários gerenciar finanças, interagir com um bot financeiro, conversar com administradores, e aos administradores gerenciar usuários e responder dúvidas.
- **Público-alvo**: Indivíduos interessados em educação financeira, desde iniciantes até usuários avançados, e administradores responsáveis pela manutenção da plataforma.
- **Tecnologias utilizadas**:
    - **Front-end**: HTML5, CSS3, JavaScript, Bootstrap 5.3.0 (via CDN)
    - **Back-end**: Node.js, Express.js
    - **WebSocket**: Socket.IO 4.5.0 para comunicação em tempo real (chat e notificações)
    - **Outras bibliotecas**: UUID para geração de IDs únicos

- **Hospedagem**: Render (Web Service com volume persistente)

3. Instalação e Configuração

Pré-requisitos

- **Node.js**: Versão 18.x ou superior
- **npm**: Versão 8.x ou superior
- **Git**: Para clonar o repositório
- **Navegador moderno**: Chrome, Firefox, ou Edge para testes locais
- **Render Account**: Para hospedagem (opcional, se for implantar)

Comandos de Instalação

1. Instalação das dependências:
2. `npm install`
3. Criação o arquivo `server/data/database.json` para armazenar os dados com a estrutura inicial:

```
4. {
5. "users": [],
6. "financialData": [],
7. "savings": [],
8. "reports": [],
9. "definitions": [],
10.  "queries": []
11.  }
```

Como Iniciar o Projeto

- Execute o servidor localmente:
- `npm start`
- Acesse a aplicação em `http://localhost:3000`.

Configuração de Variáveis de Ambiente

Crie um arquivo `.env` na raiz do projeto com as seguintes variáveis:

```
PORT=3000
```

Render atribuir uma porta automatica

4. Estrutura do Projeto

A aplicação segue uma estrutura organizada para separar o front-end (arquivos estáticos) do back-end (lógica do servidor). Abaixo está a organização das pastas e arquivos:

```
Projeto-pagina-web/
├── public/
│   ├── index.html            Página inicial
│   ├── savings.html          Página de cálculo de poupança
│   ├── reports.html          Página de relatórios financeiros
│   ├── chat.html             Página de chat com bot e administrador
│   ├── admin.html            Painel administrativo
│   ├── login.html            Página de login
│   ├── register.html         Página de cadastro
│   └── estilo.css            Estilos personalizados
├── server/
│   ├── app.js              Configuração do servidor Express e Socket.IO
│   ├── routes.js           Definição das rotas da API (login, registro,
admin, etc.)
│   └── data/
│       └── database.json     Banco de dados JSON para persistência
├── package.json              Configuração do projeto e dependências
└── README.md                 Instruções gerais do projeto
```

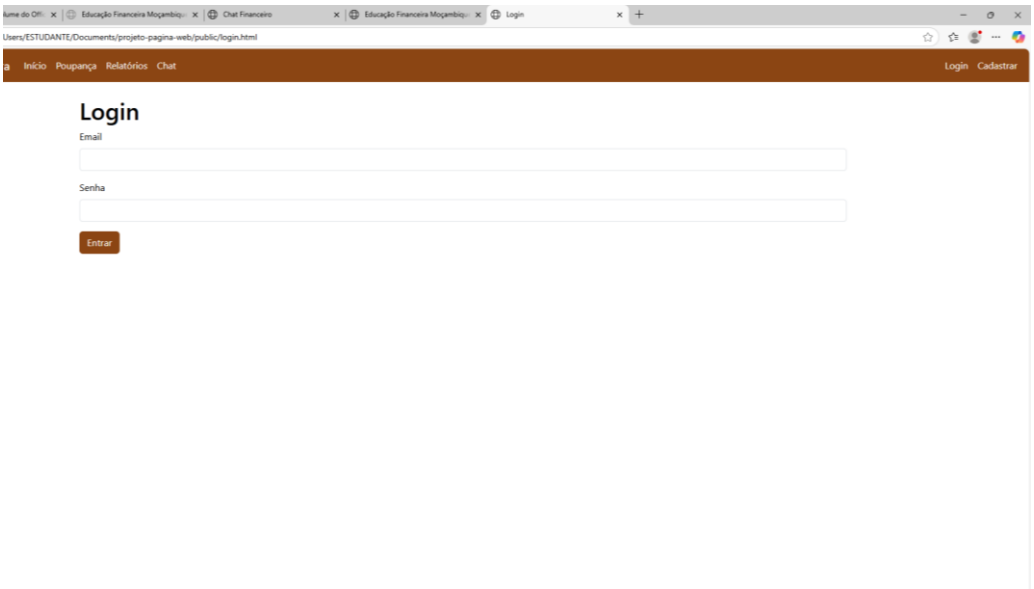
- **/server**: Contém a lógica do back-end, incluindo o servidor Express (`app.js`), rotas da API (`routes.js`), e o arquivo de dados (`database.json`).
- **/public**: Contém os arquivos estáticos do front-end, servidos pelo Express.
- **package.json**: Define scripts, dependências e metadados do projeto.

5. Funcionalidades

A aplicação possui as seguintes funcionalidades principais, implementadas com foco em usabilidade e robustez:

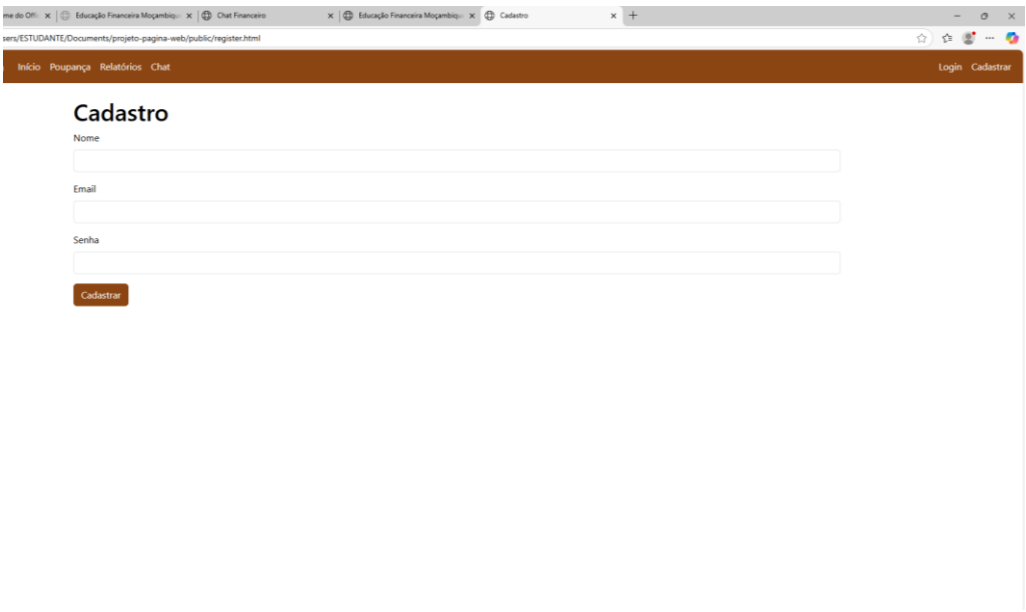
Página de Login

- **Descrição**: Permite que usuários façam login com email e senha. Após login bem-sucedido, administradores são redirecionados para `admin.html`, e usuários comuns para `index.html`.
- **Funcionalidades**:
    - Validação de campos obrigatórios (email e senha).
    - Mensagens de erro detalhadas para credenciais inválidas.
    - Armazenamento de informações do usuário (`user`, `userId`, `isAdmin`) no `localStorage`.
- **Captura de Tela**:



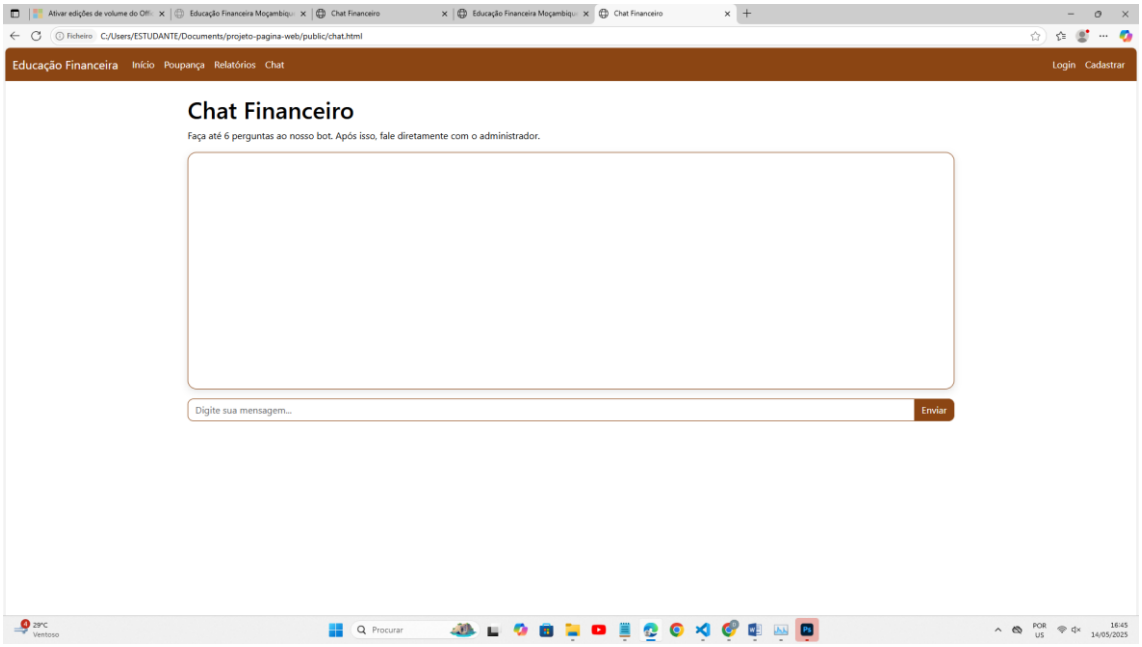
Página de Cadastro

- **Descrição**: Permite o registro de novos usuários com nome, email e senha.
- **Funcionalidades**:
    - Validação de campos obrigatórios e formato de email.
    - Verificação de email único no banco de dados.
    - Redirecionamento para a página de login após cadastro bem-sucedido.
- **Captura de Tela**:



Página de Chat

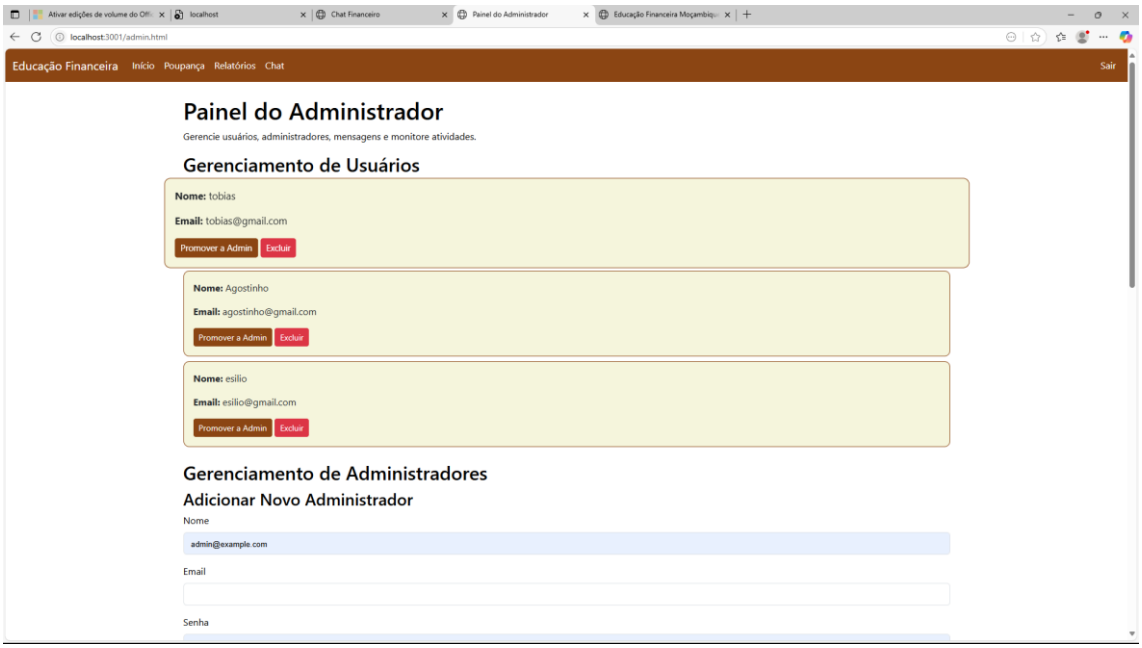
- **Descrição**: Interface para interação com um bot financeiro (até 6 mensagens) e, posteriormente, com o administrador.
- **Funcionalidades**:
    - Usuários não logados inserem um nome para iniciar o chat.
    - Bot responde com base em palavras-chave financeiras ou respostas predefinidas.
    - Após 6 mensagens, usuários logados conversam com o administrador; não logados veem uma mensagem de login/cadastro.
    - Persistência do contador de mensagens (`replyCount`) no `localStorage` para evitar reinício do bot ao navegar.
    - Mensagens do administrador e do bot aparecem à esquerda (classe `bot`), e do usuário à direita (classe `user`).
    - Persistência de mensagens no `localStorage` para continuidade do chat.
- **Captura de Tela**:

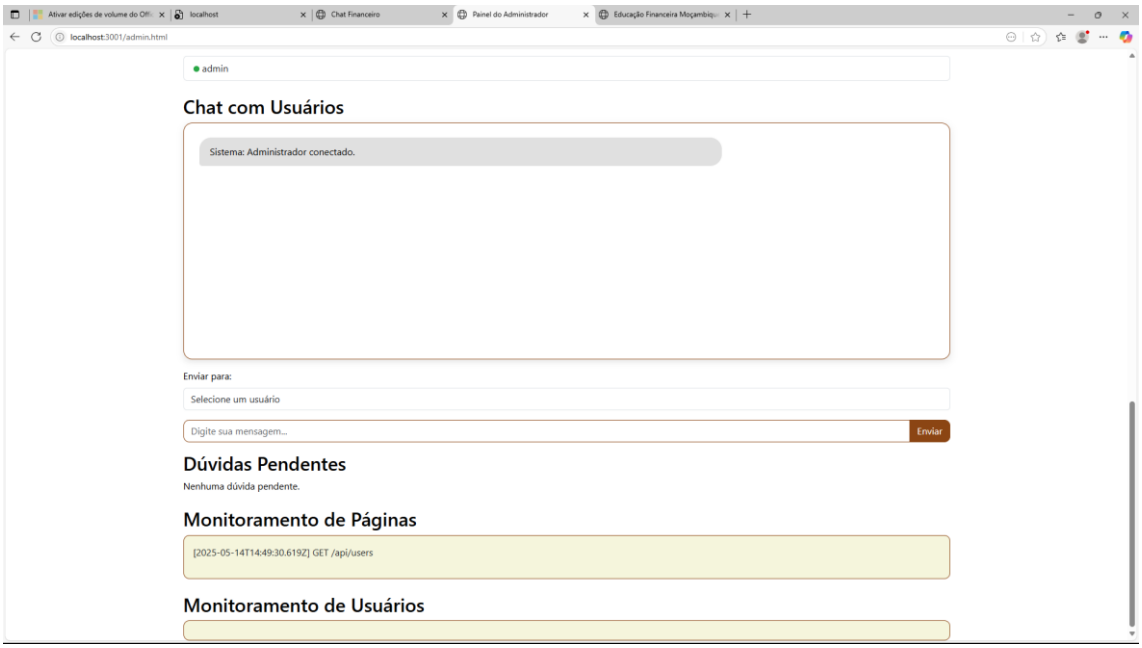


Painel Administrativo

- **Descrição**: Interface para administradores gerenciarem usuários, administradores, dúvidas e atividades.
- **Funcionalidades**:
  - Mensagem de boas-vindas com o nome do administrador exibida por 5 segundos ao acessar a página.
  - Gerenciamento de usuários (promover/excluir).
  - Gerenciamento de administradores (adicionar/editar/excluir).
  - Resposta a dúvidas pendentes dos usuários.
  - Monitoramento de atividades de páginas e mensagens de usuários.
  - Chat em tempo real com usuários.

- **Captura de Tela**:

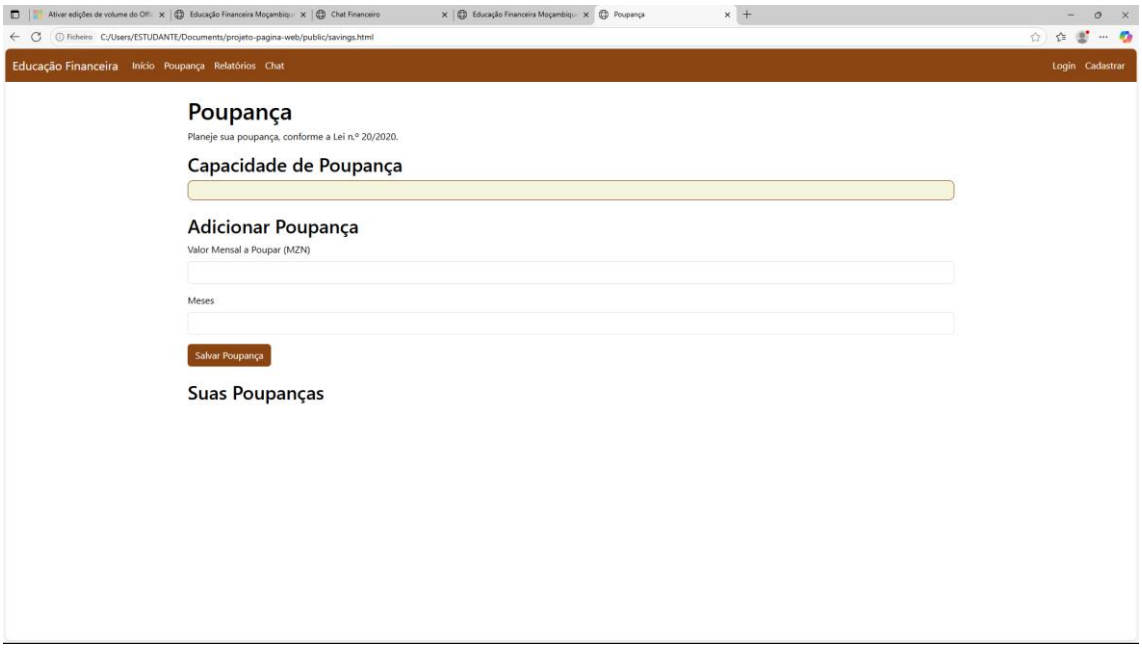




Página de Poupança

- **Descrição**: Permite aos usuários calcular e salvar planos de poupança.
- **Funcionalidades**:
  - Formulário para inserir valor mensal e período (em meses).
  - Cálculo do total poupado.
  - Salvamento dos dados no `database.json`.

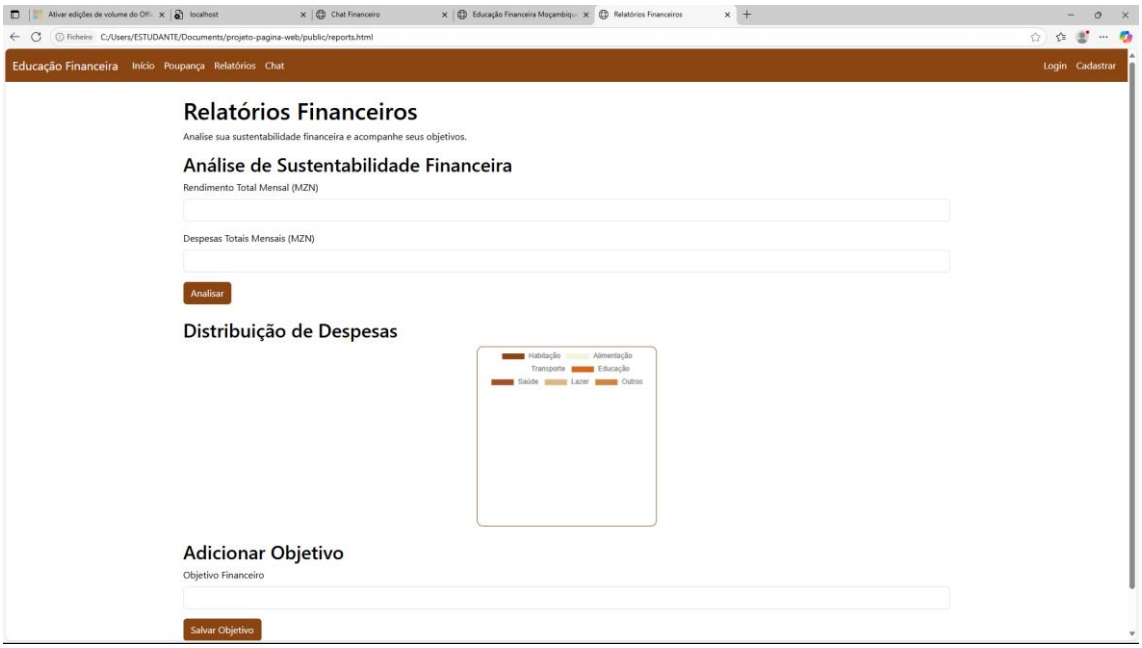
- **Captura de Tela**:



Página de Relatórios

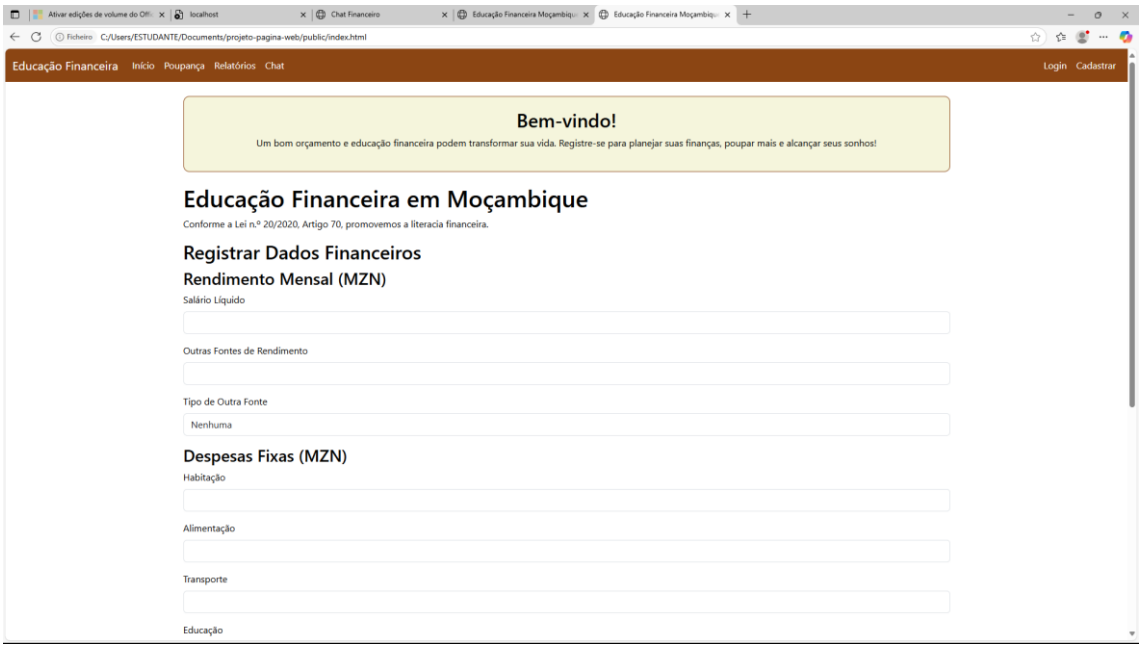
- **Descrição**: Interface para criar e visualizar relatórios financeiros.
- **Funcionalidades**:
    - Formulário para definir metas financeiras.
    - Listagem de relatórios salvos.
    - Integração com a API para persistência.

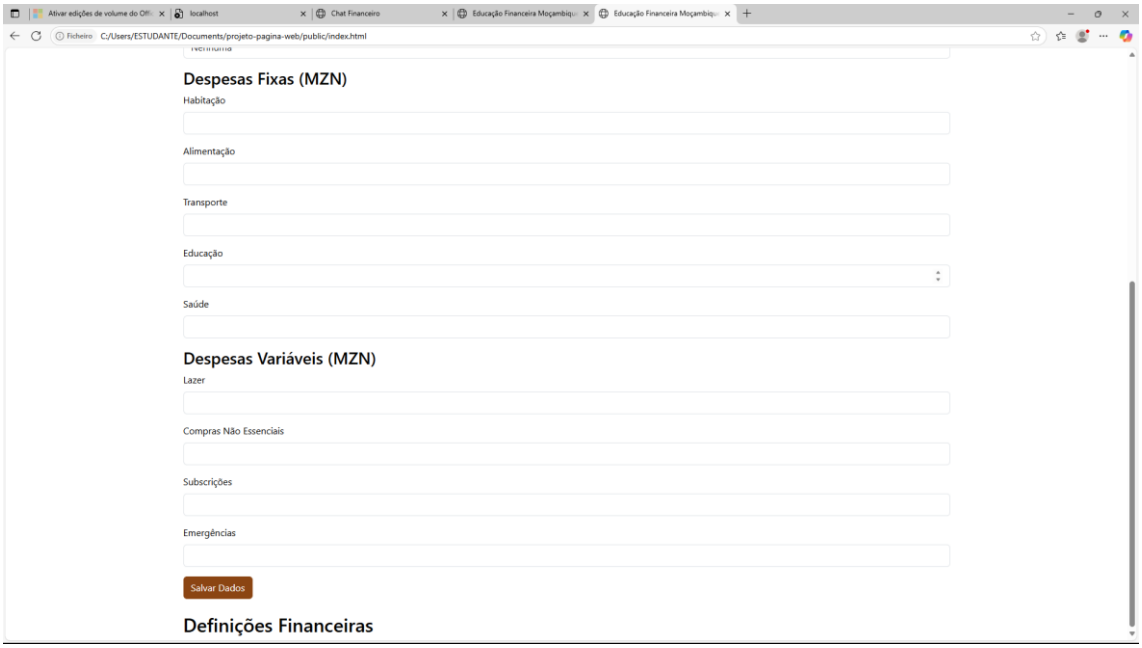
- **Captura de Tela**:



Página Inicial

- **Descrição**: Página de boas-vindas com visão geral da plataforma.
- **Funcionalidades**:
    - Navegação para outras seções (poupança, relatórios, chat, login/cadastro).
    - Design responsivo com Bootstrap.
- **Captura de Tela**:





Responsividade

- Todas as páginas utilizam Bootstrap 5.3.0 para garantir compatibilidade com dispositivos móveis, tablets e desktops.

Integração com APIs

- **Rotas da API** (definidas em `routes.js`):
  - `/api/login`: Autenticação de usuários.
  - `/api/register`: Registro de novos usuários.
  - `/api/admin/:userId`: Recupera informações do administrador.
  - `/api/users`: Lista todos os usuários.
  - `/api/queries`: Gerencia dúvidas dos usuários.
  - `/api/savings`: Gerencia dados de poupança.
  - `/api/reports`: Gerencia relatórios financeiros.
- **WebSocket** (Socket.IO):
  - Comunicação em tempo real para chat, notificações e monitoramento de atividades.

6. Documentação de Código

- **Comentários no Código**:
  - Todos os arquivos JavaScript (e.g., `routes.js`, scripts em `chat.html`, `admin.html`, `login.html`, `savings.html`, `register.html`, `report.html`, `esstilo.css`) incluem comentários explicando a funcionalidade de cada seção (ex.: "Inicializa conexão WebSocket", "Manipula envio de mensagens").
  - Os arquivos HTML contêm comentários para identificar seções (ex.: "Barra de navegação", "Conteúdo Principal").
- **Boas Práticas**:
  - Uso de `async/await` para chamadas assíncronas, com tratamento de erros via `try/catch`.
  - Validação de entradas do usuário antes de enviar para a API.
  - Persistência de dados no `localStorage` para continuidade do estado (mensagens, contador do chat).
  - Modularidade nas rotas da API, separando lógica por funcionalidade.
- **Ferramentas de Documentação**:
  - Não foi utilizado JSDoc formalmente, mas os comentários no código são detalhados para facilitar manutenção.
  - Recomenda-se adotar JSDoc para projetos maiores no futuro.

7. Testes

- **Tipos de Testes**:
  - **Manuais**: Testes manuais foram realizados para validar:
    - Fluxo de login e cadastro.
    - Persistência do contador de mensagens do chat.
    - Exibição da mensagem de boas-vindas no painel do admin.
    - Comunicação em tempo real no chat (bot e administrador).
    - Salvamento de dados no `database.json`.
  - **Unitários/Integração**: Não foram implementados testes automatizados devido ao escopo do projeto.
- **Ferramentas Recomendadas para Futuro**:

- **Jest**: Para testes unitários de funções JavaScript.
- **Cypress**: Para testes end-to-end da interface.
- **Postman**: Para testar as rotas da API.

- **Como Executar Testes**:
  - Atualmente, os testes são manuais. Para testes automatizados futuros, configurar:
  - `npm install --save-dev jest cypress`
  - `npm test`

8. Hospedagem

- **Plataforma**: Render (Web Service)
- **Passos para Deploy**:
  1. Adicionando o projeto no repositório no GitHub:
  2. `git init`
  3. `git add .`
  4. `git commit -m "primeiro commit"`
  5. `git remote add origin https://github.com/Tobias0036/Desenvolvimento_de_pagina_Web_com_Integracao_de_Tecnologias_Web_Modernas.git`
  6. `push -u origin main`
  7. No Render, clique no new e crie um novo Web Service:
     - Conecte ao repositório GitHub.
     - Configuracao:
       - **Build Command**: `npm install`
       - **Start Command**: `npm start`
  8. Habilite Auto-Deploy para atualizações automáticas no branch `main`.
- **Link de Acesso**:
  - https://desenvolvimento-de-pagina-web-com.pe26.onrender.com

9. Conclusão

A aplicação Educação Financeira foi desenvolvida com sucesso, oferecendo uma interface amigável para gestão financeira e interação com administradores. As principais conquistas incluem:

- Implementação de um chat funcional com bot e administrador, com persistência de estado.
- Painel administrativo robusto com mensagem de boas-vindas e gerenciamento completo.
- Correção de problemas de persistência de dados no Render usando volume persistente.
- Design responsivo e integração em tempo real via Socket.IO.

Lições Aprendidas

- A importância de configurar volumes persistentes em ambientes como o Render para evitar perda de dados.
- A necessidade de validações robustas no front-end e back-end para prevenir erros de usuário.
- Benefícios do uso de WebSocket para comunicação em tempo real.

Sugestões para Melhorias Futuras

- Implementar testes automatizados com Jest e Cypress.
- Migrar o banco de dados para PostgreSQL para maior escalabilidade.
- Adicionar autenticação mais segura (ex: JWT em vez de senhas em texto puro).
- Melhorar a interface do chat com recursos como anexos ou formatação de mensagens.