Documentação Desafio Handtalk

Como rodar o desafio:

- Versão do node utilizada para desenvolvimento: v22.11.0
- Recomendo (principalmente se for a primeira vez rodando o projeto) que siga a ordem Server → Plugin → Page

Server

O que você vai precisar:

- Configurar seu `.env` (pode utilizar o modelo que está no `.env.example`

Credenciais para fins de teste:

```
DB_USERNAME='eireneof'
DB_PASSWORD='kpVMMjJrgzQRPPmi'
DB_NAME='pageanalytics'
DB_HOST='clusterpageanalyticsplu.prfs7.mongodb.net'
```

Com seu `.env` configurado na raiz do projeto `Server` . Pode seguir para a instalação do projeto em si:

```
# entrar do diretório do projeto:
cd PageAnalyticsPlugin
```

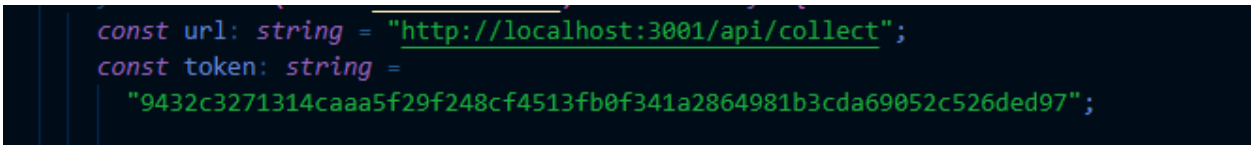
```
# entrar na pasta do servidor
cd Server

# instalar as dependências
npm i

# rodar a aplicação
npm run dev
```

Plugin

Seu `.env` para o plugin pode ficar exatamente igual ao `.env.example`

```
SERVER_PORT = 3000
```

⚠ Atenção: O Plugin já está configurado para rodar com um token de teste e espera que o **servidor** esteja na porta `3001` . Caso você queira utilizar outra porta para rodar o **servidor** ou desejar alterar o token de teste. Vá até `Plugin\src\services\collect-data.service.ts` e modifique esse trecho para a porta do **servidor** e/ou outro **token.**



Com as configurações em mãos:

```
# entrar do diretório do projeto:
cd PageAnalyticsPlugin

# entrar na pasta do plugin
cd Plugin

# instalar as dependências
npm i

# rodar a aplicação
npm run dev
```

Tokens para teste:

```
# TOKEN_1 já está configurado no código do plugin
TOKEN_1: 9432c3271314caaa5f29f248cf4513fb0f341a2864981b3cda6905

# caso queira mudar para o TOKEN_2 ou outro gerado pela api,
#lembre do pass-a-passo mencionado acima
TOKEN_2: 9f96006aff29c22de5c8240fa51e811c46d387aa1f3fa0bf410f8b
```

⚠ Atenção: Caso alguma alteração seja feita no **plugin**, ele deve ser reinicializado com `npm run dev`

Front

```
# entrar do diretório do projeto:
cd PageAnalyticsPlugin

# entrar na pasta do front
cd Page
```

```
# instalar as dependências
npm i

# rodar a aplicação
npm run dev
```

Arquitetura da Aplicação

A arquitetura da aplicação segue o padrão
MonoRepo, como foi sugerido no desafio, organizando três módulos principais: **Page**, **Plugin** e **Server.**

Segue o diagrama da aplicação:

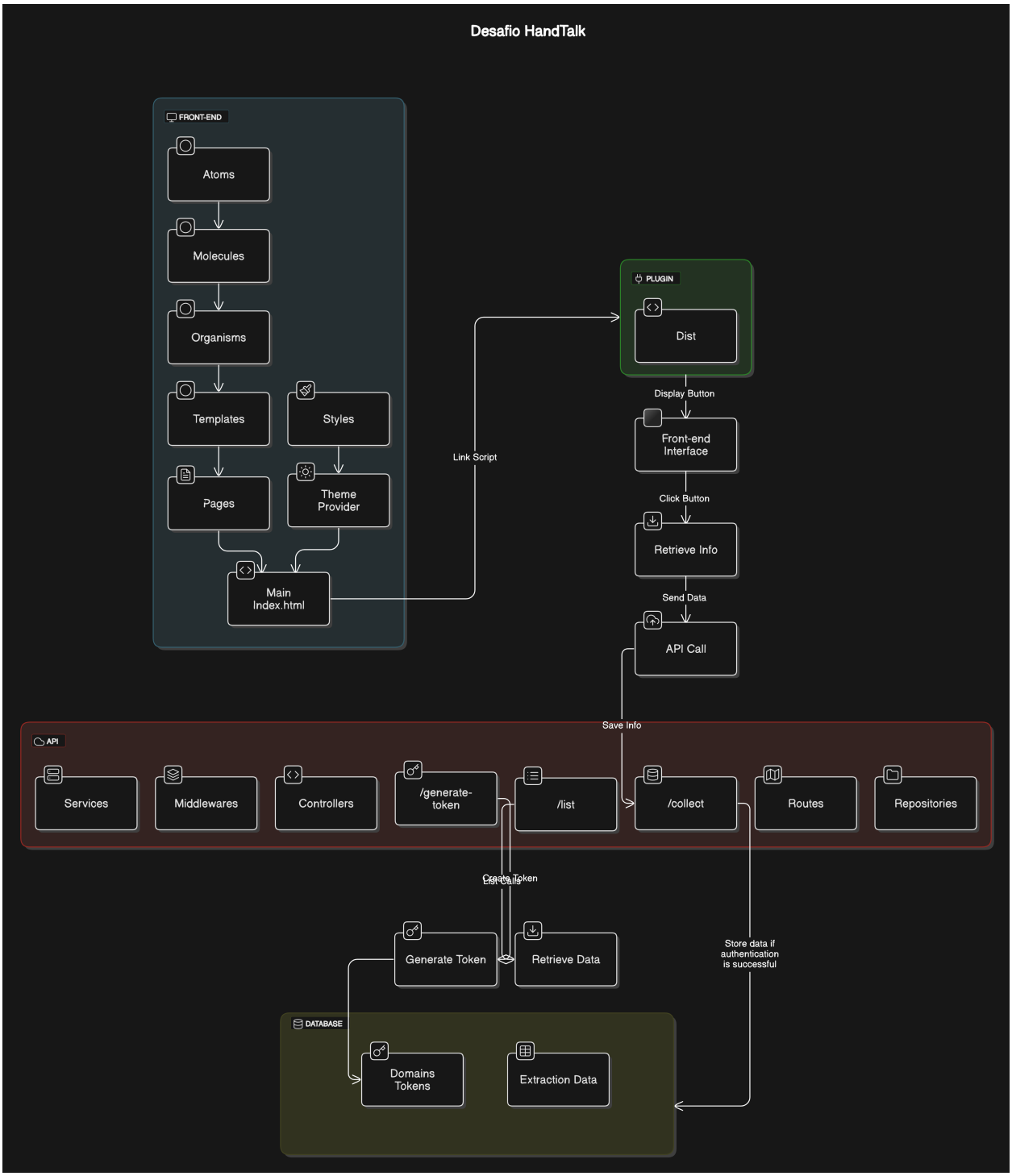
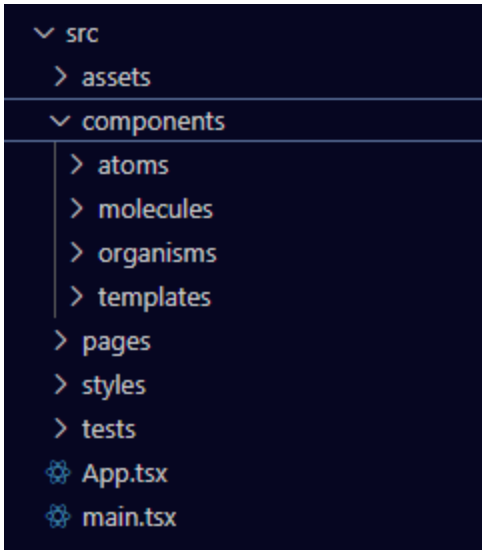


Imagem para download:

diagrama_desafio_handtalk.png

A motivação do porquê dessa arquitetura, vai ser detalhado a frente mas o foco das escolhas se centralizou no: tempo que eu tive para desenvolver o desafio, familiaridade que eu tinha para desenvolver 'X' coisa, mas ainda assim pensando em flexibilidade e escalabilidade.

Front



- **Pattern:** Atomic Design

Escolhi esse desing pattern porque ele organiza os componentes em **átomos**, **moléculas**, **organismos**, **templates** e **páginas**, facilitando a escalabilidade e manutenção.

Além disso, o escolhi por ter familiaridade com o padrão e pelas vantagens que ele oferece no desenvolvimento, como maior clareza e organização no fluxo de trabalho.

- React com Vite

- Styled-components

Foi utilizado para estilização dos componentes, aproveitando a familiaridade com a ferramenta para desenvolver de forma mais ágil e eficiente. Ele Permite estilização dinâmica com base em props, alinhando-se bem às necessidades do projeto.

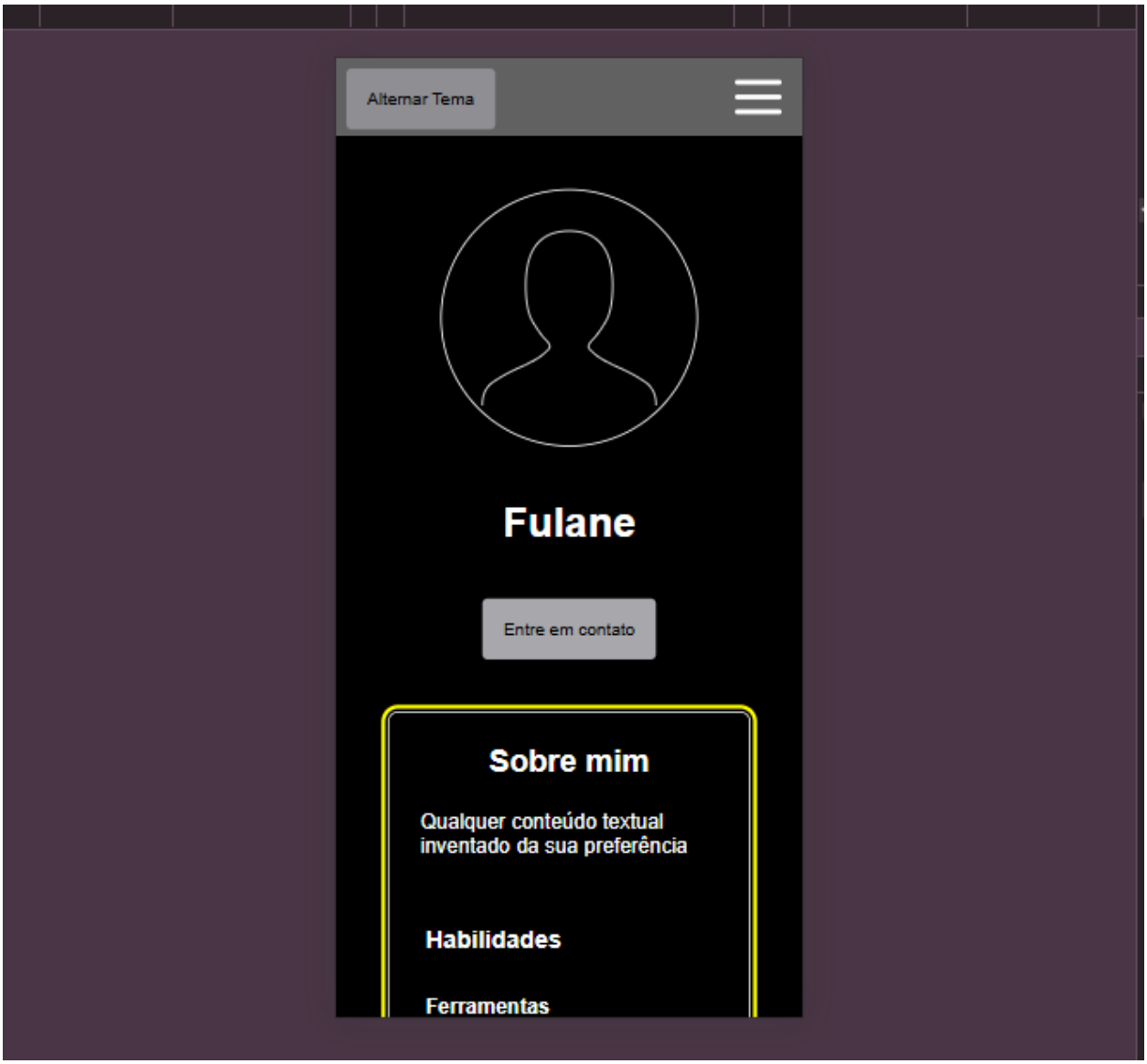
- Theme Provider Pattern

Aplicado para gerenciar valores de estilo de forma centralizada, garantindo consistência no tema da aplicação. Como um dos requisitos era ter um dark theme e um light theme, escolhi esse método para facilitar a atualização e refatoração de elementos de design, atendendo aos requisitos do desafio.

- **Jest** para testes unitários
    - OBS: decidi criar um teste unitário para o botão de troca de tema, pois o resto da página não tem muita funcionalidade, e ele é importante para a troca de tema e para um dos dados que é coletado pelo Plugin e enviado para a API.

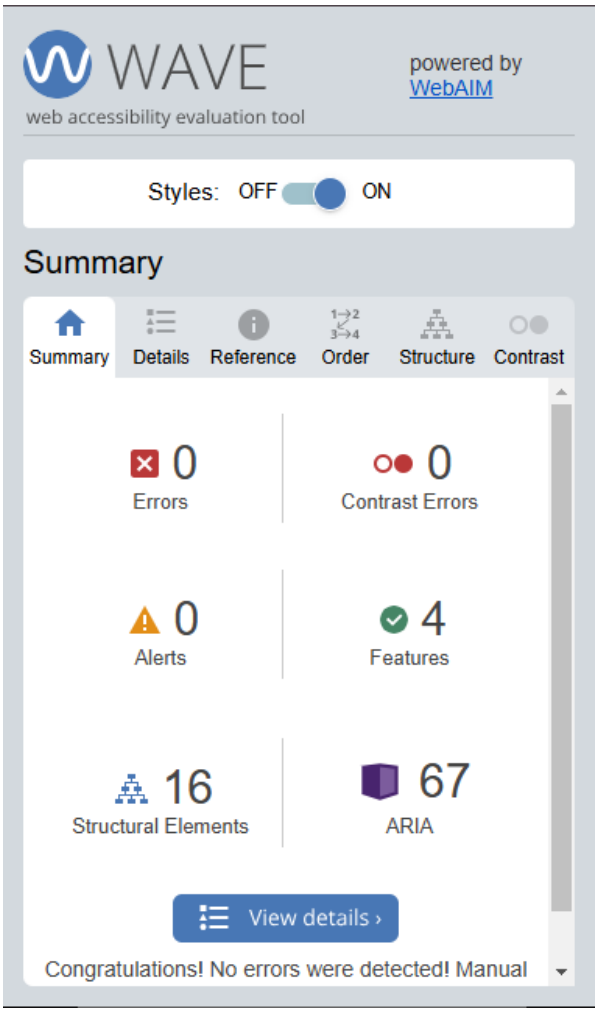
Versão desktop ligh theme:

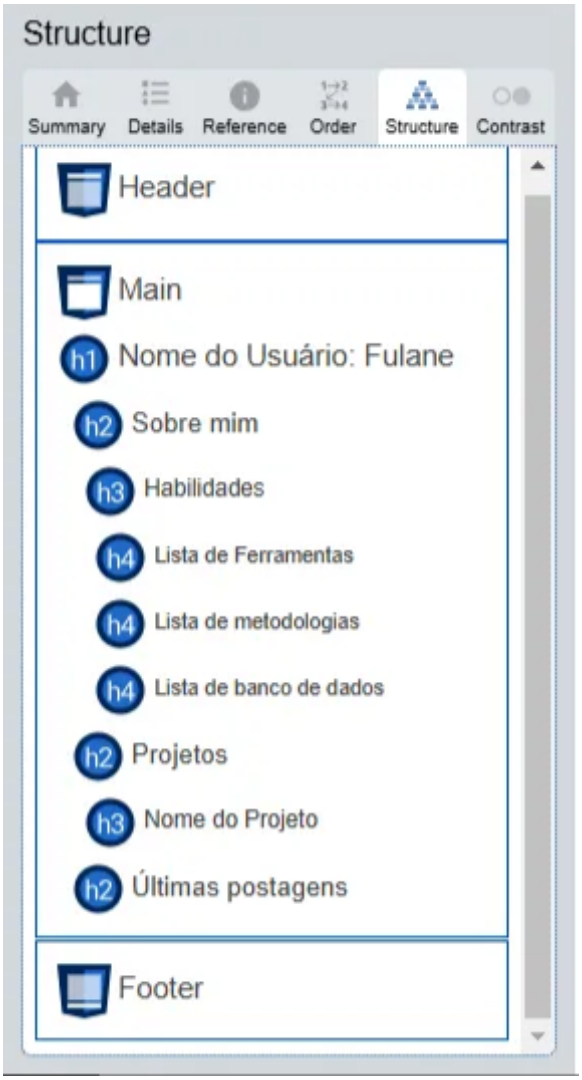
Versão mobile dark theme:



OBS: uso do foco em amarelo nos elementos para destacar a navegação por teclado

- Ferramentas que utilizei para testar a acessibilidade da página:
  - **Wave** (extensão web) - validar o padrão wacg, contraste e estrutura semântica da página
  - **Google Lighthouse**
  - **Screen Reader** (extensão web) **-** testar navegação por leitor de tela


WAVE
web accessibility evaluation tool
powered by
WebAIM
Styles: OFF ON
Summary
Summary
Details
Reference
Order
Structure
Contrast
0
Errors
0
Contrast Errors
0
Alerts
4
Features
16
Structural Elements
67
ARIA
View details ›
Congratulations! No errors were detected! Manual


Structure
Summary
Details
Reference
Order
Structure
Contrast
Header
Main
h1 Nome do Usuário: Fulane
h2 Sobre mim
h3 Habilidades
h4 Lista de Ferramentas
h4 Lista de metodologias
h4 Lista de banco de dados
h2 Projetos
h3 Nome do Projeto
h2 Últimas postagens
Footer

Plugin

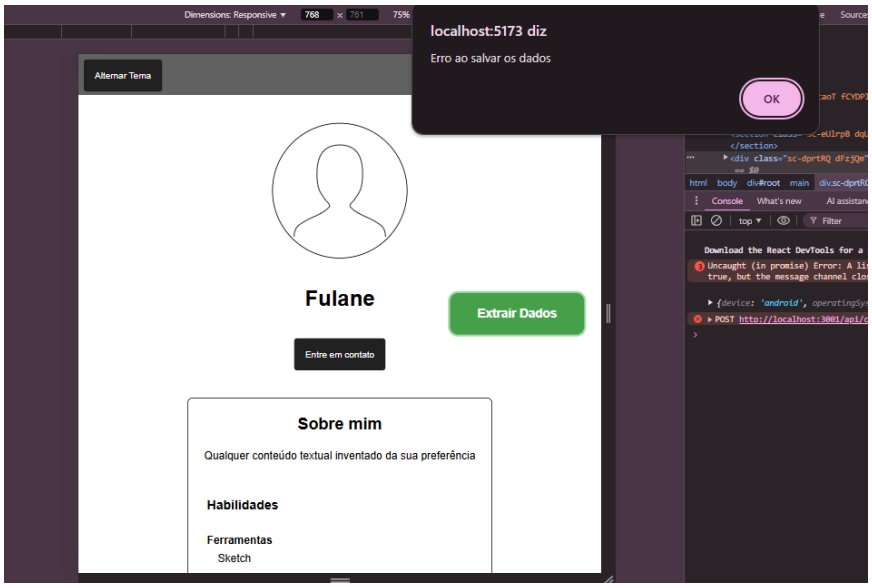

src
config
interfaces
public
services
TS server.ts

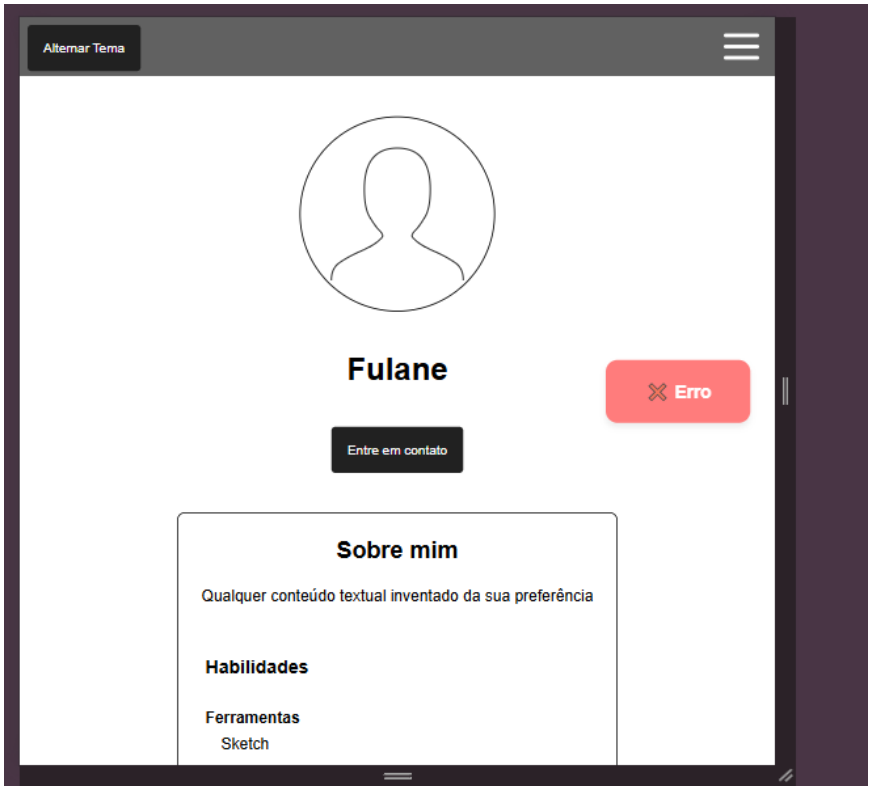
- **Pattern:** Observer Pattern - minha motivação em utilizar esse pattern foi reagir a mudanças da DOM. Eu precisava ficar observando o número de troca de temas para poder mandar para a API quando a rota `/collect` fosse chamada

```
class ThemeSwitchService {
    private setupClickListener(): void {
        if (this.button) {
            this.button.addEventListener('click', () => {
                // Handle click events
            });
        }
    }
}
```

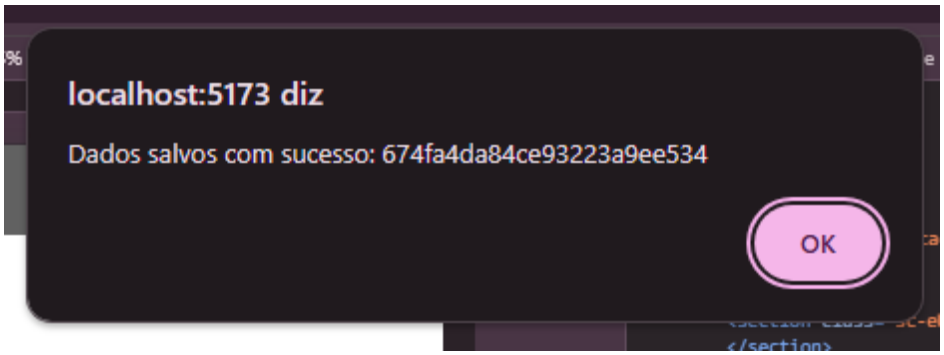
- Princípios SOLID - Single Responsibility Principle: Tentei ao máximo, separar as responsabilidades de cada service para facilitar ao deixar a aplicação mais modular, com seus serviços reutilizáveis e fáceis de refatorar (e manter)
- **Jest** para testes unitários
    - Obs: decidi como alvo do meu teste unitário o serviço de data-collect por acreditar que ele é a principal feature do plugin.

Sistema de feedback é feito tanto por alert quanto pela apresentação de ícones no botão do plugin (✔️ e ❌)





Quando os dados são salvos com sucesso, o id é consolado no browser e também aparece no alert.



API

- **Pattern:** Service Layer Pattern
  - Centralizar a lógica de negócios
  - Não sobrecarregar os controllers
  - Facilidade na hora de implementar testes, modularizando as camadas fica mais amigável de testar e refatorar
  - como a lógica de negócios está centralizada, ela pode ser reutilizada por diferentes partes da aplicação

- Aqui também me concentrei em implementar o princípio de responsabilidade única
- **Testes com Jest:** Foque em criar dois testes, uma para a rota `collect` e outro para a rota `list` para garantir a confiabilidade na chamada desas rotas que fazem integração com as outras partes do projeto
- Validação com `Joi`
- Rotas da API:
  - `api/collect POST` - Autenticação via token
    - salva os dados enviados pelo plugin
  - `api/list?token=token GET` - Rota pública
    - pega os últimos 20 dados extraídos pelo token informado
  - `api/generate-token POST` - Rota pública
    - gera um token para o domínio informado
- Collection Postman:

OBS: collection disponível do repositório do projeto na pasta na raiz: `assets`

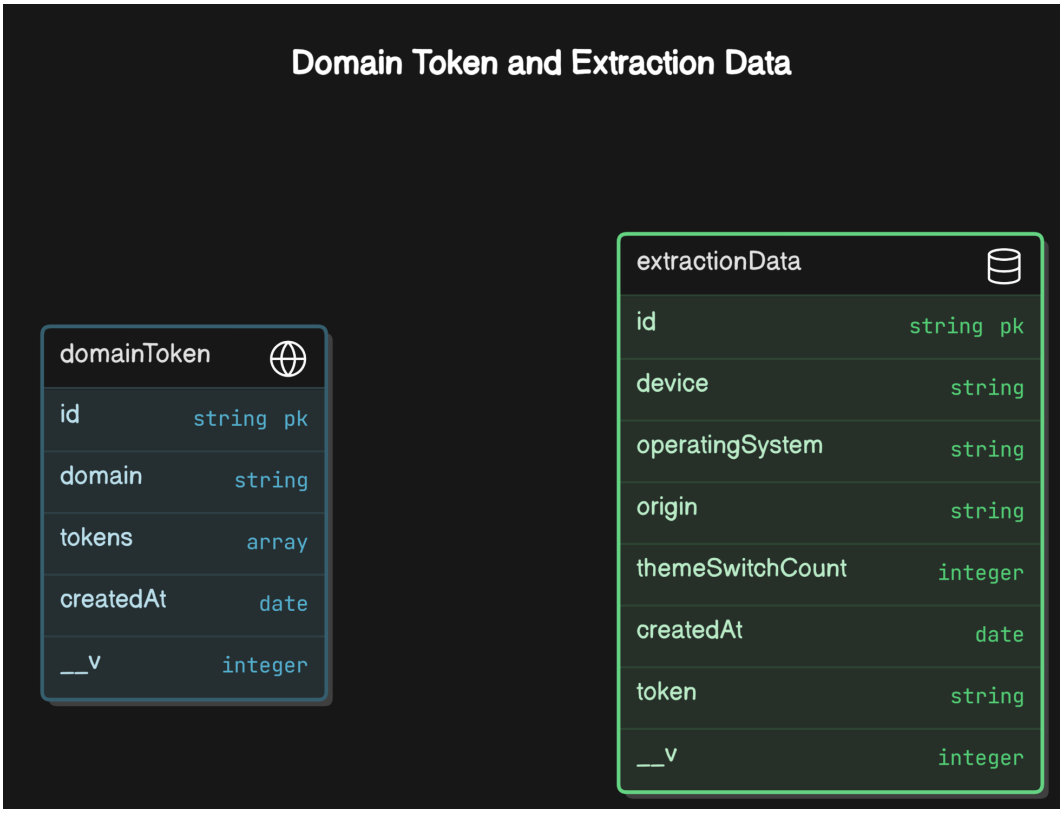
> PageAnalyticsPlugin.postman_collection.json

`{{url_base_page_analytics_plugin}}` = `http://localhost:3001/api`

`{{token}}` = `9432c3271314caaa5f29f248cf4513fb0f341a2864981b3cda69052c526ded97` ou `9f96006aff29c22de5c8240fa51e811c46d387aa1f3fa0bf410f8b03da148709`

Banco de dados

O banco utilizado foi `mongoDB` conforme solicitado



OBS: tokens está como array, mas o serviço de gerar token só gera um token de fato se o domínio requisitante não tiver nenum token, e seu limite máximo é um. Isso aconteceu, pois pensei em desenvolver uma estratégia com mais de um token por domínio, mas acabei não indo para frente com a ideia.

Outros tópicos

Ferramentas em comum que utilizei nos 3 repositórios

- Prettier
- Eslint
- Sonar (Extensão do VSCode)

Ideias de melhoria:

Ao longo do desenvolvimento tive algumas ideias de implementação para melhorias (então é possível encontrar alguns `TODO` 's por aí 🙂)

- Front:
  - Deixar tema armazenado em contexto para que seu valor atual não se perca a atualizar a página e possa ser observável
  - Deixar página mais dinãmica com dados mockados simulando uma API (ex: lista de posts)
- Plugin:
  - Aplicar variáveis de ambiente no serviço de coletar os dados
  - Encontrei um bug: Em aba anônima a estratégia de espera para carregamento dos dados da DOM aparentemente não funciona (mas só percebi depois, e precisei focar em outros requisitos de implementação)
- API:
  - Implementar uma rota para remover tokens específicos
  - Fazer com que os tokens expirem depois de um tempo