ISCTE-IUL
Licenciatura em Ciência de Dados

TODO TITULO

Projeto realizado no âmbito da Unidade Curricular de
Projetos em Ambientes Web e Cloud

Allan Kardec Rodrigues, 103380
aksrs@iscte-iul.pt

André Plancha, 105289
Andre_Plancha@iscte-iul.pt

Abril de 2024
Versão 1.0.0

Índice

1. Introdução

React é uma *framework* de *javascript* que ajuda no desenvolvimento de websites, auxiliando na criação de interface e interatividade entre o utilizador e a *backend*. A biblioteca põe à disposição várias ferramentas para reutilização de partes de HTML (chamadas de componentes), e em conjunto com a biblioteca **React Router** (uma ferramenta para navegação rápida entre páginas do website), possibilita a criação de websites completos e eficientes. Neste projeto, foi criado um website de venda de livros com estas bibliotecas, em conjunto com um conjunto de outras. Para isso, foi-nos dado um pequeno conjunto de informações sobre livros para simular a venda deles, e o nosso objetivo foi implementar a *frontend* do website.

Este relatório serve de documentação do projeto, incluindo explicação das várias funções criadas, decisões tomadas, e observações feitas. O *source code* encontra-se disponível no GitHub[1].

2. Bibliotecas Usadas

Para além do `react-dom` e do `react-browser-router`, um conjunto de outras bibliotecas e ferramentas foram usadas na elaboração do projeto. Estas vão sendo apesentadas ao longo do relatório quando apropriado, mas há algumas principais importantes de notar, sendo que são bases para o nosso projeto.

- Vite <vitejs.dev>
  Vite é uma ferramenta de *frontend* com o propósito de compilar, agrupar e servir o nosso código para que este seja utilizável na web. Neste projeto ele está empregado de tranformar o nosso código e servi-lo para um url destino. Este contém também um conjunto extensivo de extensões (*plugins*) e configuração que pudemos usar. Este apresenta também um bom conjunto de documentação no seu website de como começar um projeto com react[2], e é recomendado pelos *MDN Web docs*[3].
- Bulma <bulma.io>
  Bulma é uma *framework* de css que contém um conjunto de componentes compatível com qualquer ferramenta, pois apenas contém classes pre-feitas de css. Esta tornou-se uma biblioteca principal devido à sua flexibilidade e facilidade de implementação, estando presente na maioria dos componentes criados. No entanto, os componentes e classes que provém desta biblioteca são limitados a css devido à sua natureza flexível, e os componentes criados precisaram de ajustes.
- shadcn/ui <ui.shadcn.com>
  shadcn/ui tem uma coleção de componentes de componentes reusáveis que usámos para várias partes do nosso interface. Estes componente fazem parte de outras bibliotecas, juntando-as em apenas uma fonte pesquisável. Estes componentes, ao contrário dos compoentes do Bulma, não estão limitados a css, e de facto tem os seus componentes preparados para usar em React. No entanto, o uso desta biblioteca tornou-se complicado porque apenas começámos a usar a meio do projeto, e nem tudo presente revelou-se compatível com o Bulma. Ainda assim, alguns componentes são usados no website.
- UnoCSS <unocss.dev>
  UnoCSS é um motor de CSS Atómico; ou seja, a biblioteca gera um conjunto de classes de css programaticamente, cada uma delas definindo poucas regras de css. Há um conjunto grande de vantagens sobre css direto, como por exemplo o encapsulamento que vem com CSS *inline*, em conjunto com uma facibilidade de escrita e revisão dos estilos aplicados[4].

[1] https://github.com/boladouro/web_e_cloud/

```html
<p class="w-full h-4 text-orange">Hello</p>
<!-- Equivalente a -->
<p style="width: 100%; height: 4em; color: rgb(251 146 60)">Hello</p>
```

Listing 1: Demostração do UnoCSS

- styled-components <styled-components.com>

  styled-components, semelhante ao UnoCSS, serve para encapsular estilos a certos elementos. A diferença é que estes são mais esplícitos e mais verbosos, o que é útil em certos casos, como por exemplo para fazer componentes pequenos rapidamente com estilos específicos, replicar dentro de um componente varios estilos sem expor esses estilos para o resto do projeto, ou para diagnosticar um problema de estilo qualquer[4].

```jsx
const StyledDiv = styled.div`
  display: flex;
  flex-direction: column;
  gap: 10px;
  justify-content: start;
  z-index: 5;
  background-color: rgb(37, 68, 86);
  margin: 1em;
  padding: 2em;
`;
export const T = () => {
  return <>
    <StyledDiv><p>Isto está dentro de um div com estilos!</p></StyledDiv>
    <StyledDiv><p>Isto também!</p></StyledDiv>
  </>
}
```

Listing 2: Demostração do styled-components

- TypeScript <typescriptlang.org>

  TypeScript é uma linguagem de programação construida em cima de JavaScript, com o propósito de dar "tipos" ao javascript. Esta ajudou bastante no projeto, pois conseguimos encontrar rapidamente erros durante a escrita do programa, sendo uma ferramenta usada principalmente como preventiva. A adoptação abrangente desta ferramenta pelas outras bibliotecas ajuda também na nossa escrita do código, pois pudemos imediatamente saber o que certas funções desejam receber, ou o que é suposto uma variável ser ou não ser.

- json-server <github.com/typicode/json-server>

  Para simular uma backend, um dos requerimentos foi usar o *json-server*. Esta ferramenta simula uma REST API perante um ficheiro json, para podermos fazer pedidos para essa api e termos respostas. Esta ferramente revelou-se muito limitada, ao ponto de nos levar a fazer *downgrade* da biblioteca para a sua versão 0.* e adicionar um método de filtro adicional, através de uma *fork* da biblioteca[5]. Mais informações sobre isto encontra-se no capítulo

A quantidade de bibliotecas usadas para componentes e css foram um resultado de um conjunto de curiosidade de usar ferramentas novas, e um mau planeamento de estrutura de estilos. Achamos que para projetos futuros que necessitem de ser mantidos a longo-termo, uma escolha de bibliotecas de estilos e uma idealização da estrutura destes irá trazer melhores resultados de organização. Ainda assim, a quantidade de opções que podiamos tirar proveito de foi de grande parte uma vantagem na elaboração do projeto.

[2]Foi-nos recomendado usar o `create-react-app` para criar o projeto, mas achámos que o Vite trazia um conjunto de vantagens apropriadas para o projeto, como a velocidade, custumizabilidade e a intergração com as outras ferramentas

[3]https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started

[4]Andrei Pfeiffer descreve com mais detalhe as vantagens e desvantagens de CSS atómico e CSS-em-JS no seu blog: https://andreipfeiffer.dev/blog/2022/scalable-css-evolution/part6-atomic-css

[5]disponível em github.com/notPlancha/json-server

3. Estrutura do código

O projeto encontra-se organizado da seguinte forma:

- `dist/`

  Aqui encontra-se a compilação resultada, gerada automaticamente com o Vite, optimizada para produção, a partir de `npm run build`. Para visualizar esta *build*, é recomendado correr `npm run preview`.

- `node_modules/`

  Aqui encontra-se as bibliotecas usadas e as suas dependências, gerada automaticamente com `npm install`.

- `public/`

  Aqui encontram-se documentos servidos pelo vite, disponíveis globalmente no projeto diretamente.

  - `public/global.css`

    Aqui encontram-se os estilos globais, disponiveis em qualquer lugar, através de classes. A maioria deste documento contém mudanças gerais feitas ao Bulma, e os estilos + animações para o componente `<PrettyBook />`.

  - `public/nocover.png`

    Aqui encontra-se a imagem usada para substituir quando um livro não tem capa. Foi criada manualmente.

  - `public/vite.svg` O svg daqui veio no template inicial do projeto, durante a criação dele. Neste momento serve de icone para o website, sendo que não nos foi fornecido um, e o icone é vibrante e está de acordo com o tema do website.

- `src/`

  Esta pasta contém o código desenvolvido, cujo vai ser o alvo da compilação.

  - `src/Components/`

  Esta pasta contém os vários componentes elaborados ao longo do projeto, excluindo aqueles que compões páginas.

  - `src/Components/ui`

  Esta pasta contém os vários

  - `src/routes/`

  Esta pasta contem os vários componentes que compões páginas, sendo elas os resultados de navegação.

  - `src/entry.tsx`

  Este ficheiro é o principal do projeto; neste,é importado todo o CSS necessário (excepto o `public/global.css`, este é importado no `index.html`), gerido o router (mais detalhes à frente), onde é criado os *Contexts* (mais à frente também), e onde o React se insere no `index.html`.

  - `src/loaders.ts`

  Este ficheiro contém a grande parte das requests feitas à base de dados, a partir de funções *loaders* (mais à frente).

  - `src/lib/utils.ts`

  Este ficheiro foi criado automaticamente na configuração do shadcn/ui, e depois foi aproveitado para outras funções de ajuda.

  - `src/types.ts`

  Neste ficheiro encontram-se definições de tipos e interfaces (de TypeScript) que usámos, incluindo o interface para a definição de um livro como se encontra em `db.json`.

  - `src/tailwind.css`

  Os componentes de shadcn/ui precisam duma ferramente de *Atomic CSS* chamada *Tailwind CSS*[6], e este documento é importado no `src/entry.tsx`, sendo possível a utilização deste.

- `db.json` Esta e a base de dados, servida com o json-server (`npm run dev-back`).
- `index.html` Esta é a base do ficheiro que o React se vai inserir em, sendo esta a base da página. Só tem as tags necessárias do HTML, uma referência para o `global.css`, e uma chamada do `entry.tsx`.
- `*config.*` e `components.json`

  Estes ficheiros contém configurações para várias partes do projeto, e a maioria delas não têm mais do que o predefinido. As diferenças principais são que foi adicionado o processador de UnoCSS em `vite.config.ts`, e foi definido em `tailwind.config.js` o prefixo "tw-" (para não haver problemas de compatibilidade entre o UnoCSS e o TailWind quando usado nos componentes do shadcn/ui).
- `package.json` Este ficheiro inclui várias definições do projeto, incluindo as dependências e *scripts*. O script principal é o `npm run dev`, que corre ambos o *json-server* e o *vite*, deixando o website a funcionar completamete e com live-reload automatico, embora não esteja optimizado para a web.

4. Routes

A navegação, como referido anteriormente, é feita com o React Router. Para esse fim, foi criado um *BrowserRouter*[7], o que define os vários caminhos que o utilizador pode navegar para. O utilizador pode navegar para as seguintes Routes:

- `/home` Esta é a página principal, onde se encontram uma página inicial minimalista, e os destaques.

[6]https://tailwindcss.com/

[7]https://reactrouter.com/en/main/routers/create-browser-router