ISCTE-IUL
Licenciatura em Ciência de Dados

React + React Router + json-server: Um Website Sobre Venda de Livros

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. A biblioteca deixou também possível fazer temas de noite e de dia, baseado no sistema do utilizador.

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

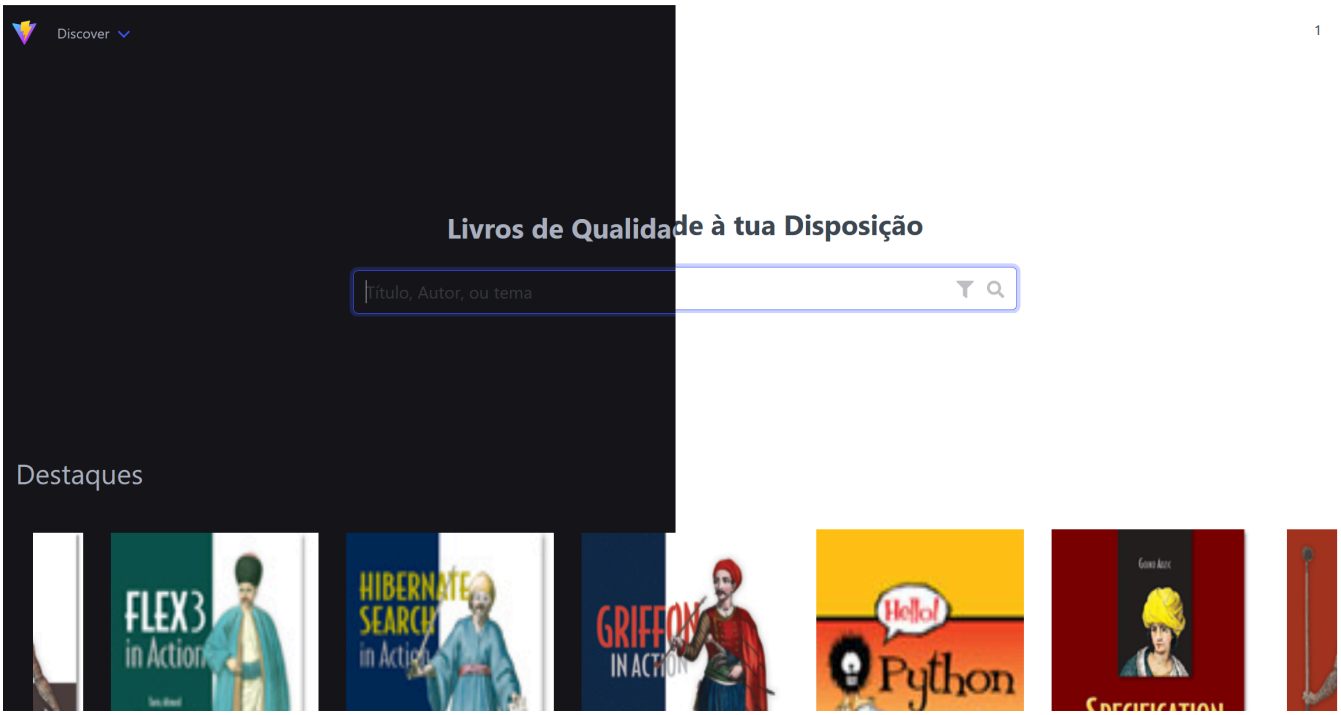

Figura 1: Modo noite (esquerda) e modo dia (direita), baseado no sistema

- 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].

```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.

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.

[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

- `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, com ajuda do evento onError. 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. Este Browser Router vai ter o componente `<Root />` como base para poder ter sempre presente o `<Header />` e o `<Footer />`, em conjunto com o componente ligado a *route* específica, com ajuda do `<Outlet />`[8]. Algumas *routes* têm funções *loaders*, específicamente o `book/` e o `search/`; estas vão ser apresentadas de seguida. Adicionalmente, o Router vai para uma página de erro `<ErrorPage />` quando há um erro, como por exemplo uma navegação para um livro que não existe.

O utilizador pode navegar para as seguintes Routes:

- `/home`

  Esta é a página principal, onde se encontram uma página inicial minimalista, e os destaques. Se um utilizador navegar para o \, será redirecionado para aqui. Esta *route* é navegada para quando é clicado o icone do canto da página.
- `book/:bookId/:bookTitle?`

  Esta é a página de um livro com id `bookId`. Em conjunto com o `shouldRevalidate` e o `<Navigate/>` dentro do `BookPage.tsx`, quando uam pessoa navega para `book/:bookId/` será redirecionado para `book/:bookId/:bookTitle?`, com o título do livro apropriado. Isto está desenhado para deixar claro qual livro o utilizador está a navegar para quando vê este link, e o `shouldRevalidade` faz com que o componente não seja recriado quando o utilizador é redirecionado. Esta *route* é navegada para quando é clicado qualquer livro (a partir do `<BookComponent/>`).
- `search/?q=...&page=...` Esta é a página de resultados de uma pesquisa qualquer feita. Esta *route* é navegada para quando é feito *submit* numa `<SearchBar />`.
- `cart/` Esta é a página do carrinho/cesto de compras. Esta *route* é navegada para quando é clicada no cesto, no canto superior direito.

4.1. Funções loaders

Funções loaders são funções que correm antes da renderização do componente, com o propósito de ir buscar dados. No nosso caso, estas vão fazer *api requests* ao `json-server`. Há algumas vantagens de usar isto invés de usar o `useEffect`, uma delas sendo por exemplo o use do `shouldRevalidate`[9].

No nosso Browser Router, há 2 loaders: `searchLoader()` e `bookLoader()`. A segunda é mais simples que a primeira, e faz um request para `GET /books/:bookId`, obtendo o bookId do parametro da *route*. O primeiro precisa de uma explicação extensiva.

4.1.1. Pesquisa de livros

O nosso website tem uma extensa funcionalidade de pesquisa, incorporando *full-text search*, filtros extensos e ordenações. Para este fim, decidimos incorporar estes filtros diretamente na pesquisa usando sintaxe especial, para esta ser escrita em conjunto com *queries* gerais.

[6] https://tailwindcss.com/

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

[8] https://reactrouter.com/en/main/components/outlet

[9] Este forum post fala mais sobre o assunto: https://forum.freecodecamp.org/t/react-router-loaders-vs-useeffect/589483

As nossas *queries* gerais são aquelas que vão incorporar a pesquisa em vários campos: O título, a data de publicação, as descrições do livro, os autores, as categorias, e o isbn. Isto significa que o utilizador pode por exemplo introduzir um isbn (ou parte de um isbn) para pesquisar por esse isbn, sem interações adicionais. Para poder usar esta funcionalidade, tivémos que fazer *downgrade* do `json-server` e adicionar uma funcionalidade adicional a tal. Essa funcionalidade adicional encontra-se em github.com/notPlancha/json-server, e essa *fork* foi introduzida no nosso projeto usando `npm i notPlancha/json-server#v0`.

Os nossos filtros foram implementados com os seguintes sintaxes:

- `title:word`
- `author:word`
- `category:cat1,cat2, ...`
- `sort:field`

Estes na pesquisa encontram-se na mesma no parâmetro `q`, portanto o utilizador pode escrever estes diretamente; alternativamente, podem usar o painel disponível na barra de pesquisa (dentro de `<SearchBar />`). Achamos que esta funcionalidade é flexível para os utilizadores, e facilita a implementação de novos filtros que queiramos adicionar, principalmente a elaborar o interface, sendo que apenas tem que se adicionar ao parametro `q` invés de criar um novo parâmetro.

Essencialmente, `searchLoader()` vai pegar no `q`, tirar os filtros de lá, e fazer um `GET /books?q=:q&attr= ...` . Notamos que a versão atual de `json-server` (v1) não contem um parâmetro de `q`, estando ela apenas disponível na versão anterior (v0). Os outros parâmetros também mudam, a documentação destas está disponível em github.com/notPlancha/json-server.

O `searchLoader()` também tem outra funcionalidade: paginação da pesquisa. A página atual do utilizador encontra-se no parâmetro `page` (da *route* `/search?q= ... &page= ...`), sendo o predefinido a primeira página. Sendo que o `json-server` devolve também os links das páginas no *header* "link" da resposta, nós aproveitamos a resposta e devolvemos esta informação para o componente.

4.2. Routes Componentes

As seguintes componentes de *route* encontram-se dentro de `src/routes`, com o seu nome como nome de ficheiro.

4.2.1. `<Header />`

A componente `<Header />` vai ser renderizada em todas as páginas (semelhante ao `<Footer />`), pois ela encontra-se chamada no `<Root />`. Tem três partes principais: o icone do website, a parte de pesquisa, e o cesto. Ela está formatada de acordo com a navbar do Bulma[10].

- Na parte do icone, foi usada o `<Link />`[11] para voltar para o `/home` quando o utilzador desejar;
- Na parte de pesquisa, há duas partes:
  - Um *dropdown* de *discover*;
  - Uma barra de pesquisa. A barra de pesquisa encontra-se escondida até a barra de pesquisa de `<Home/>` sair do ecrã, seja por navegar ou seja por scroll. Isto é implementado com ajuda do `<Waypoint />`[12], e deixa possibilidade de ter uma barra de pesquisa grande para o `<Home/>`, para incentivar a pesquisa de livros e não haver confusão de qual barra usar: se houvesse duas, iria estar confuso.
- Na parte do cesto, está um icone do cesto para navegação e um número, indicando o número de livros dentro do cesto, guardando esse número através do Context[13] do React, que nos deixa ter uma interação com as várias

[10] https://bulma.io/documentation/components/navbar/

páginas e o cesto, sem necessidade de ter essa *prop* declarada em quase todos os componentes. A desvantagem de usar o Context aqui é que na saida do website (ou num *reload*), esse context é perdido.

4.2.1.1. `<SearchBar/>`

A barra de pesquisa é um componente `<Form />`[14], com input com dois butões, continuando a ter o estilo provenciado pelo Bulma[15], modificado levemente para ter 2 butões adicionais: um botão de filtro, e outro de pesquisa. O segundo apenas é um botão que submete o *form*, mas o de filtro abre uma caixa de diálogo `<Dialog />`[16], contendo os vários filtros mencionados anteriormente. A seleção dos filtros irá adicionar ao input o filtro com ajuda do `useState` e dos eventos `onChange` (no caso das categorias, sendo que esta é uma dropdown), `onBlur` e `onKeyDown`. Adicionar os filtros desta forma irá também fazer uma notificação, para ajudar o utilizador com o processo dos filtros de sintaxe, com ajuda do `<Toaster />`[17] (presente no `<Root />`).

Para ajudar nos filtros das categorias e na ordernação, decidimos fazer um *dropdown*[18]. No caso das categorias, foi feita uma API call para obter as categorias, para `GET http://localhost:3030/books` (com ajuda do `useEffect`), e depois foi tirada as categorias daí. Em retrospectiva, podiamos ter implementado routes adicionais com o *json-server*[19], no entanto achamos que para estes pedidos mais avançados, já seria uma mais valia uma backend mais completa, e portanto achámos que seria demasiado para o nosso projeto.

4.2.2. `<Home/>`

A *route* `/home` vai renderizar o componente `<Home/>`, que inclui duas partes principais: a primeira visualização do website (sendo que isto e a página inicial), e os destaques. Na elaboração desta página tirámos inspiração da página atual de zillow.com, e portanto esta segue um layout parecido, adotando uma página minimalista.

Os destaques são renderizados num carrossel[20], com ajuda do `<BookComponents />`. Este carrossel vai mostrar os ultimos livros publicados com maior pontuação, usando `GET books?_limit=8&_sort=score,publishedDate.$date&_order=desc,desc`, com ajuda do `useEffect`.

4.2.3. `<Search />`

A página de pesquisa está dividida em duas partes: um *grid* de `<BookComponents/>` usando o Grid esperto do Bulma[21] porque tem boa responsividade; e a paginação no fim da página, se houver mais do que uma página.

A existência de paginação leva ajuda do `useEffect`, depois da função loader devovler os livros da pesquisa e dados sobre a paginação. Estes dados são os seguintes:

- O número da página anterior, se houver[22]
- O número da página seguinte, se houver;
- O número da primeira página;
- O número da última página.

[11] https://reactrouter.com/en/main/components/link
[12] https://react-restart.github.io/ui/Waypoint/
[13] https://react.dev/learn/passing-data-deeply-with-context
[14] https://reactrouter.com/en/main/components/form
[15] https://bulma.io/documentation/form/input/
[16] https://ui.shadcn.com/docs/components/dialog
[17] https://ui.shadcn.com/docs/components/toast
[18] Com ajuda do Bulma: https://bulma.io/documentation/form/select/
[19] https://github.com/notPlancha/json-server#add-custom-routes
[20] Foi usado o slick (https://kenwheeler.github.io/slick/) e o React Slick (https://react-slick.neostack.com/) para isso
[21] https://bulma.io/documentation/grid/smart-grid/
[22] Só não há quando é a primeira página

Com isto, e com ajuda do `<Pagination />`[23], a paginação é dinâmica, tendo sempre visível a primeira e última página, a página atual, a seguinta e a anterior, tendo condições para a não aparência da proxima e da anterior se elas foram a primeira ou a última, respetivamente, e condições sobre a distância à ultima e primeira.

4.2.4. `<BookPage />`

A página do livro teve como base uma *codepen* de Fivera[24]. Foram adicionados informações sobre o livro dentro da animação, tornando a esperência da página única e interativa.

4.2.5. `<Cart />`

A página de cesto é outro carousel, deixando o elemento responsivo no em ecrãs pequenos, de `<CartComponent />`s. Estes componentes são baseados no `<BookComponent />`, mas tem novos botões para a funcionalidade de tirar e por no cesto. Para saber os livros que foram postos no cesto, usámos o `useContext`, em que nele estão os próprios objetos dos livros. Alternativamente, podiamos ter apenas gravado os Ids num array, mas isso precisava de mais requests ao api. Essa estratégia tem a vantagem de deixar sincronizado o livro (como por exemplo uma mudança repentina no preço), no entanto como os nossos dados são estáticos de momento isso não é necessário.

5. Conclusão

Na execução deste trabalho, aprendemos várias ferramentas novas, muitas lecionadas em aula, mas muitas não. Este projeto foi uma demonstração de curiosidade e serviu de instrumento de aprendizagem, não só das possibilidades do React e do React Router, mas também das suas (poucas) limitações. O uso de novas bibliotecas, incluindo não lecionadas como o UnoCSS e o styed-components, revelou novas técnicas de resolução de problemas que nós ainda não tinhamos visitado previamente. Aprendemos também por experiência como organizar um projeto para a web, e temos a certeza que o próximo terá uma preparação melhorada devido a isto.

[23] https://ui.shadcn.com/docs/components/pagination
[24] https://codepen.io/fivera/pen/kQJzxP