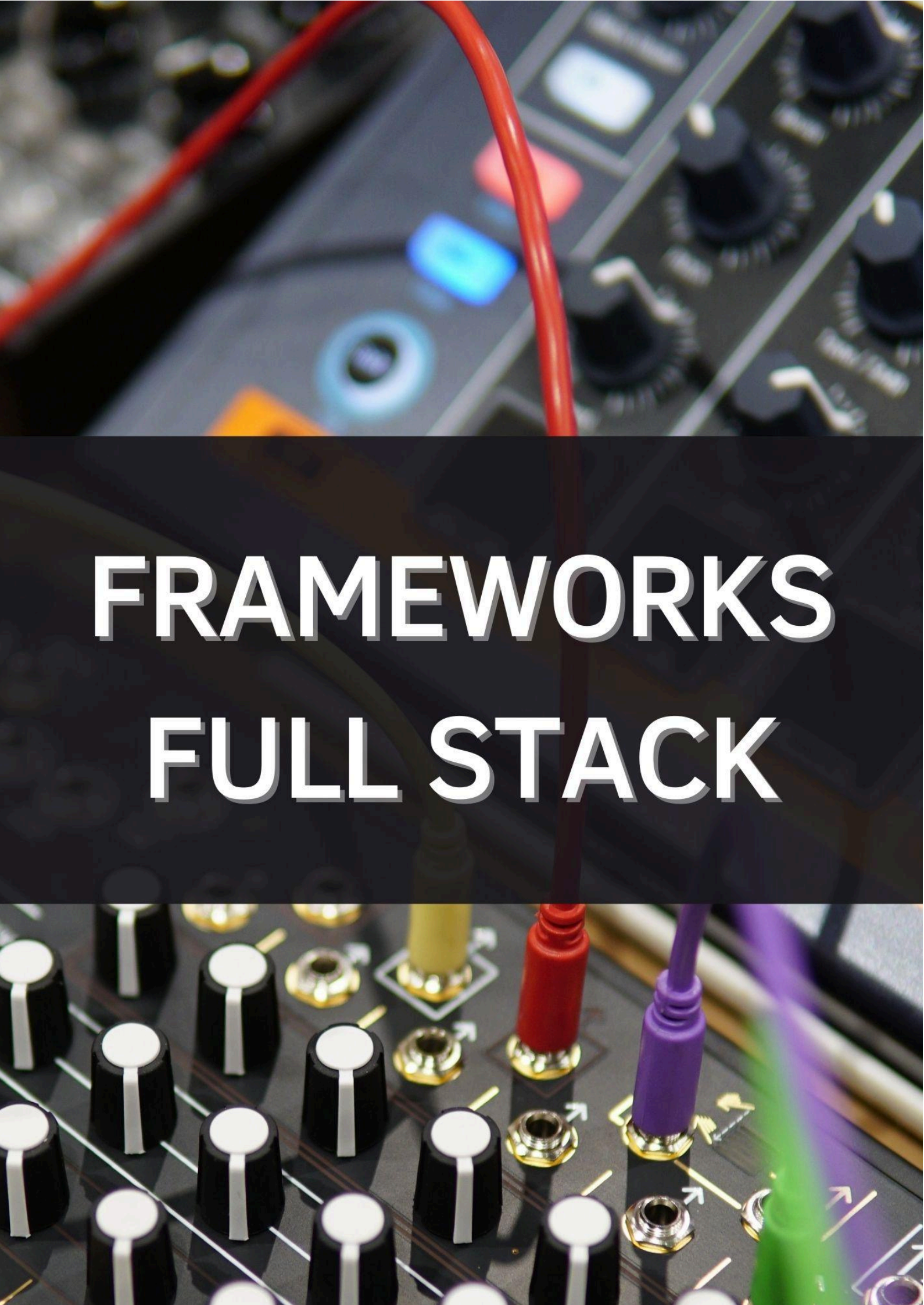
FRAMEWORKS
FULL STACK

Texto base

6

Realização de uma venda

Prof. André Nascimento Maia
Prof. Caio Nascimento Maia

Resumo

Uma web application é composta de diversas páginas, visões, componentes e possibilidades de manipulação de informação. Dificilmente uma única visão resolve problemas complexos. A navegação utilizando o componente Link do framework Next.js melhora a experiência do usuário e performance da aplicação. Além do componente de navegação, as formas permitidas para busca de dados pelo Next.js são importantes para garantir produtividade e eficiência no desenvolvimento.

6.1. Introdução

O *framework* Next.js tem um jeito particular de navegação entre as páginas. Utilizando o componente Link do Next.js, internamente o *framework* lida com tarefas difíceis como *pre-fetch* e *cache* de consultas de dados.

Uma particularidade do *framework* Next.js é resolver os problemas que *Single Pages Application* causam para o Search Engine Optimization (SEO) (Google, 2022). Em Next.js é possível renderizar o conteúdo no servidor em tempo de execução ou em tempo de compilação, utilizando dados vindos de uma API e entregar apenas a versão estática da página. Isso também facilita a utilização de *Content Delivery Networks* (CDN) (Akamai, 2022), aumentando a eficiência e entrega praticamente instantânea do conteúdo e visão para o usuário, o que garante uma ótima experiência para o cliente.

Neste texto, é utilizado o contexto de um e-commerce para a realização de uma venda. Nesse contexto, comentaremos sobre como a navegação entre as páginas deve acontecer e como utilizar as técnicas de busca de dados do *framework* Next.js para garantir uma ótima experiência ao usuário.

6.2. Navegação entre páginas

O Next.js utiliza o sistema de arquivos para criar todo o roteamento entre páginas. É possível navegar para qualquer rota diretamente informando a URL do navegador. Contudo, a forma mais eficiente de navegação é utilizando o componente Link do *framework* (Vercel, 2022).

O Código 6.1 mostra um exemplo de uso do componente Link para a navegação da página de listagem de produtos para a página de detalhes de produtos.

Código 6.1: Navegação utilizando o componente Link.

```
01.        <Link href={`/products/${props.product.code}`}>
02.          <a className="stretched-link">
03.            <h3 className="mt-0">{props.product.title}</h3>
04.          </a>
05.        </Link>
```

Fonte: autores, 2022.

Na linha 01 do Código 6.1 temos o componente Link declarado e uma propriedade chamada *href* que contém a rota de destino da página de detalhes do produto. Perceba que a nossa rota é dinâmica, dessa forma, concatenamos o código do produto diretamente à URL. Minimamente, o filho direto do componente Link, deve ser uma âncora HTML (MDN Web Docs, 2022) com o seu texto interno (linha 02 até 04).

Para que a navegação sobre as rotas geradas automaticamente no Next.js sejam eficientes, utilizem sempre o componente Link.

6.3. Busca de dados com Next.js

O *framework* Next.js permite algumas formas de busca de dados para exibição em *client-side*. As formas principais são: Server-side rendering (SSR), Static-site generation (SSG) e Client-side rendering (CSR) (Vercel, 2022).

A busca de dados é um dos aspectos mais importantes para uma boa experiência do usuário que utiliza a *web application*. Diversas técnicas de *cache* de dados e estratégias de pré-renderização de conteúdos devem ser aplicadas para garantir maior fluidez durante a utilização da aplicação. A boa notícia é que o *framework* Next.js implementa as diversas técnicas utilizando as suas formas de buscas de dados.

6.3.1. Server-side rendering

Server-side rendering é uma estratégia de busca de dados utilizada para pré-renderizar os dados buscados em tempo de execução. Isso significa que em cada *request* para a página que utiliza *server-side rendering* o *framework* Next.js irá pré-renderizar os dados retornados de uma função específica da página, chamada

getServerSideProps. O Código 6.2 mostra como exportar a função *getServerSideProps* em uma página para que essa estratégia seja utilizada.

Código 6.2: Utilizando Server-Side Rendering (SSR)

```
01. function Page({ data }) {
02.   // Render data...
03. }
04.
05. // This gets called on every request
06. export async function getServerSideProps() {
07.   // Fetch data from external API
08.   const res = await fetch(`https://.../data`)
09.   const data = await res.json()
10.
11.   // Pass data to the page via props
12.   return { props: { data } }
13. }
14.
15. export default Page
```

Fonte: Vercel, 2022.

O Código 6.2 exibe uma página chamada Page (linha 01) utilizando a estratégia de SSR para obter dados de uma API fictícia. Os dados retornados pela API externa, são passados para a página via *props*.

6.3.2. Static-site generation

Utilizando a mesma estratégia que o SSR, a estratégia de *Static-site generation* (SSG) também utiliza um função na página para permitir a busca de dados e o resultado é passado via *props*. Porém, a principal diferença entre os dois é que neste caso os dados são obtidos em tempo de compilação. Sendo assim, não há necessidade de buscar dados a cada requisição do usuário, os dados são buscados uma única vez e são mantidos de forma estática, garantindo grande eficiência para cache e na distribuição via CDN.

O Código 6.3 mostra a utilização da estratégia SSG. Perceba que o código é igual ao utilizado na estratégia de SSR, a única diferença é que o nome da função exportada é getStaticProps ao invés de getServerSideProps.

Código 6.3: Utilizando Static-Site Generation (SSG)

```
01. function Page({ data }) {
02.   // Render data...
03. }
04.
06. export async function getStaticProps() {
07.   // Fetch data from external API
08.   const res = await fetch(`https://.../data`)
09.   const data = await res.json()
10.
11.   // Pass data to the page via props
12.   return { props: { data } }
13. }
14.
15. export default Page
```

Fonte: autores, 2022.

6.3.3. Client-side rendering

Diferentemente das outras estratégias de SSR e SSG, a estratégia de *Client-side rendering* (CSR) acontece sempre do lado do cliente e não do lado do servidor. Sempre que o cliente renderiza uma página ou componente React e durante a montagem desses for necessário obter dados de uma API REST essa estratégia pode ser utilizada.

Essa estratégia é útil quando não é necessário se importar com requisitos como *Search Engine Optimization* (SEO). Os web *crawlers* (Patel, 2022) geralmente não executam o código Javascript em sua página aguardando o retorno e renderização dos dados para indexa-las, eles geralmente indexam HTML e JSON retornados em uma requisição.

O Código 6.4 exibe um trecho de código de um componente chamado Profile que utiliza essa técnica, utilizando o *hook* chamado *useEffect* (linha 05) para que a chamada para a API REST aconteça necessariamente quando o componente é construído. A chamada para a API REST é feita utilizando a API Fetch (MDN Web Docs, 2022).

Código 6.3: Utilizando Client-Side Rendering (CSR)

```
01. function Profile() {
02.   const [data, setData] = useState(null)
03.   const [isLoading, setLoading] = useState(false)
04.
05.   useEffect(() => {
06.     setLoading(true)
07.     fetch('api/profile-data')
08.       .then((res) => res.json())
09.       .then((data) => {
10.         setData(data)
11.         setLoading(false)
12.       })
13.   }, [])
14.
15.   if (isLoading) return <p>Loading...</p>
16.   if (!data) return <p>No profile data</p>
17.
18.   return (
19.     <div>
20.       <h1>{data.name}</h1>
21.       <p>{data.bio}</p>
22.     </div>
23.   )
24. }
```

Fonte: Vercel, 2022.

Referências

AKAMAI. (2022). **O que é uma CDN?** Akamai. Disponível em: <https://www.akamai.com/pt/our-thinking/cdn/what-is-a-cdn>. Acessado em: 05 fev. 2022

GOOGLE. (2022). **SEO Starter Guide**: The Basics | Google Search Central. Google Developers. Disponível em: <https://developers.google.com/search/docs/beginner/seo-starter-guide>. Acessado em: 05 fev. 2022

MDN Web Docs. (2022). **<a> - HTML**: linguagem de marcação de hipertexto | MDN. MDN Web Docs. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/Element/a>. Acessado em: 05 fev. 2022

MDN Web Docs. (2022). **Fetch API** - APIs da Web | MDN. MDN Web Docs. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API>. Acessado em: 06 fev. 2022

PATEL, N. (2022). **Web Crawler**: entenda o que é, quando usar e como funciona. Neil Patel. Disponível em: <https://neilpatel.com/br/blog/web-crawler/>. Acessado em: 06 fev. 2022

VERCEL. (2022). **Data Fetching**: Overview. Next.js. Disponível em: <https://nextjs.org/docs/basic-features/data-fetching/overview>. Acessado em: 05 fev. 2022

VERCEL. (2022). **Routing**: Introduction. Next.js. Disponível em: <https://nextjs.org/docs/routing/introduction>. Acessado em: 05 fev. 2022