

NEXTjs

Vimos react, bom, o next basicamente ‘nasceu’ do react, porém, criando paginas dinamicas e renderização de paginas ao lado do servidor. Uma das principais diferenças que eu vi quando comparei com react, é que não tem index.html e onde o index.js pega o id e carrega o JSX a partir daquele id, no nextjs na hora de compilar tudo, ele gera um arquivo html. Outra coisa:

```
layout.js M x layout.jsx U page.jsx U header.js U page.js .../app M page.js .../cat
aula > src > app > layout.js > ...
9   });
10
11  const geistMono = Geist_Mono({
12    variable: "--font-geist-mono",
13    subsets: ["latin"],
14  });
15
16  export const metadata = {
17    title: "Create Next App",
18    description: "Generated by create next app",
19  };
20
21  export default function RootLayout({ children }) {
22    return (
23      <html lang="en">
24        <body
25          className={` ${geistSans.variable} ${geistMono.variable} antialiased`} >
26          <Header/>
27          {children}
28        </body>
29      </html>
30    );
31  }
```

esse é o index que carrega o children, que é tem que ter o nome de *page*. O index importa de forma automatica sem importar no arquivo js o componente, ele faz isso automaticamente graças aos nomes (*layout* → carrega o *page*)

OBS: *layout* é um nome obrigatório também

Navegação entre páginas



Assim como no react, usamos o Link, mas do next, invés do “to=”, usamos o “href=”

```
src > app > page.js > ...
import Image from "next/image";
import Link from "next/link";

export default function Home() {
  return (
    <div>
      <Link href="/posts">Posts</Link>
      <Link href="/param?q=">Parametros</Link>
      <Link href="/posts/categoria">categoria</Link>
    </div>
  );
}
```

Pode ver que temos uma query string bem diferente, não é com `/?nome=`, com next é de forma mais direta

“E como pegamos o valor passado na query string?” – simples!

```
aula > src > app > param >  page.js >  PageParamns
1  "use client";
2  import { useSearchParams } from "next/navigation";
3  //para query
4  export default function PageParamns(){
5      const search = useSearchParams().get("q");
6      return(
7          <div>
8              <h2>Parametros</h2>
9              <p>Resultado: {search}</p>
10         </div>
11     );
12 }
13 }
```

Usamos o `useSearchParams()` serve pra **acessar os parâmetros da query string** na URL. Agora para pegar o valor, usamos o `.get()`

“O que é o `use client`?”

Para tornar um componente client-side no Next.js


No Next.js com o **App Router** (`/app`), por padrão, **todos os componentes são Server Components**. Isso significa que eles **rodam no servidor**, não no navegador.

Mas tem situações em que você **precisa rodar algo no cliente** (navegador), como:

✚ **Quando usar `use client`:**

- Hooks como `useState`, `useEffect`, `useSearchParams`, `useRouter`, etc.
- Eventos de clique, mudança de input, animações no DOM
- Acessar `localStorage`, `window`, `document`
- **Renderizar componentes que precisam de interatividade**

"`use client`" diz pro Next.js:

“ Ei Next.js, **esse componente precisa rodar no navegador** (client-side), porque ele **vai interagir com o usuário** ou usar coisas que **só existem no navegador**.”

Por que "roda no navegador" o `useSearchParams`?

1. Interação com a URL:

Quando você usa `useSearchParams()`, você está basicamente acessando os parâmetros na **URL da página atual**. Isso acontece de forma **dinâmica**. Ou seja, a URL do navegador pode mudar sem que a página inteira seja recarregada, por exemplo, quando você navega de forma interna no Next.js usando `router.push()` ou quando o usuário interage com os parâmetros da URL diretamente.

Esse comportamento depende de APIs como `window.location` e outras propriedades do navegador, que **não existem no servidor**. Assim, o `useSearchParams()` só pode "**ver**" a **URL atual** quando o código estiver rodando no **navegador**.

2. Roteamento no Next.js:

O Next.js tem uma camada de **Roteamento baseado em hooks** que permite acessar e alterar a URL sem forçar o carregamento completo da página. Esse comportamento é feito no **lado do cliente**:

- O **SSR (Server-Side Rendering)** no Next.js renderiza a página no servidor, mas **não tem acesso dinâmico à URL do cliente**. Ele apenas processa os parâmetros da URL quando a página é solicitada inicialmente.
- Quando a página é carregada e o JavaScript do cliente é executado, a URL pode ser atualizada dinamicamente (exemplo: parâmetros de pesquisa podem mudar). Então, é nesse momento que o **`useSearchParams()`** entra em cena, acessando e manipulando os parâmetros da URL diretamente **no lado do cliente**.

Mas calmaaa, meu amigo!

Lembra que a gente falou que as páginas só podiam ter um arquivo com nome de page, não poderia ter outro, sabe como fazemos essa nomeação ? Na real, para ser mais exato, como funciona ?

Você cria uma pasta com o nome, tipo, “posts” e dentro dela, o .page, e ai, na hora da compilação, aquela pasta se tornará o caminho, ou seja, a página

Porque o **Next.js entende que cada pasta é uma rota**, e o `page .tsx` dentro dela é a **renderização principal daquela rota**.

- A pasta = o **caminho na URL**
- O `page .tsx` = o **conteúdo daquela página**

Se tivesse dois arquivos `page .tsx` na mesma pasta, o Next não saberia qual renderizar pra aquela rota, então ele exige **um por pasta**.

O Next.js tem um **sistema de roteamento baseado em arquivos**, e ele faz isso:

1. **Lê a pasta /app inteira**
2. Identifica **cada subpasta como um segmento da URL**
3. Procura por arquivos especiais como:
 - `page .tsx` → conteúdo da rota
 - `layout .tsx` → layout da rota ou rota pai
 - `loading .tsx` → carregamento
 - `error .tsx` → erro
4. Gera as rotas do site automaticamente com base nisso



Como o Next.js transforma pastas e arquivos em páginas?

A "mágica" não é mágica de verdade, é engenharia: o Next.js usa **file-based routing** (roteamento baseado em arquivos) e um processo de build que mapeia tudo direitinho.

💡 Etapas do que acontece nos bastidores:

1. Scan do diretório /app

Quando você roda `npm run dev` ou `next build`, o Next:

- Vasculha toda a pasta /app
- Identifica todas as pastas como **segmentos de URL**
- Procura arquivos como:
 - `page.tsx`: conteúdo da página
 - `layout.tsx`: estrutura comum para várias páginas
 - `loading.tsx`, `error.tsx`, etc.

2. Criação automática das rotas

Ele **monta uma árvore de rotas internamente** com base na estrutura de pastas. Algo tipo isso:



📦 E os arquivos `.js`, `.ts`, `.tsx`, `.jsx`?

Durante o build:

- O Next **transpila (compila)** todos esses arquivos com o Babel e o SWC (um compilador ultra rápido)
- Depois ele **gera bundles otimizados** para cada página e rota
- Esses bundles são carregados **sob demanda** no navegador

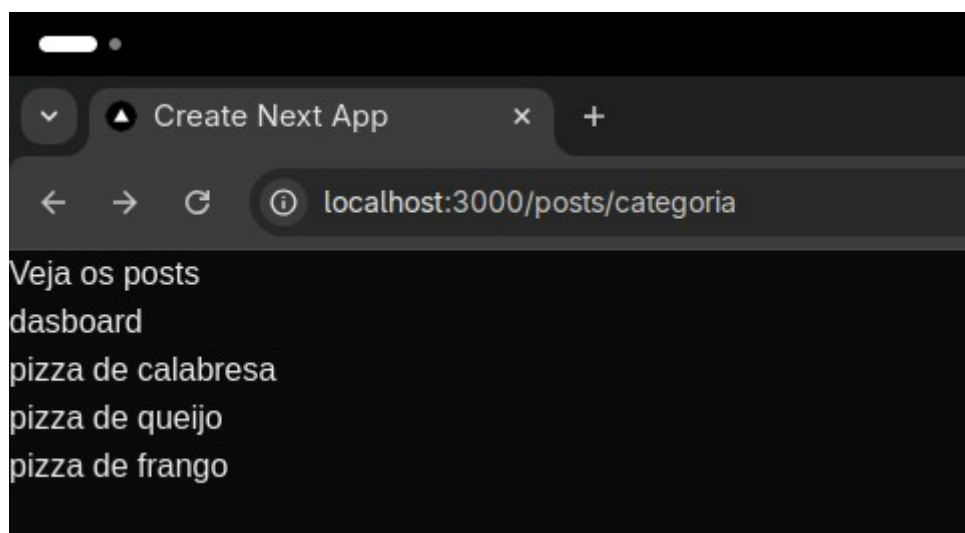
Roteamento em páginas dinâmicas

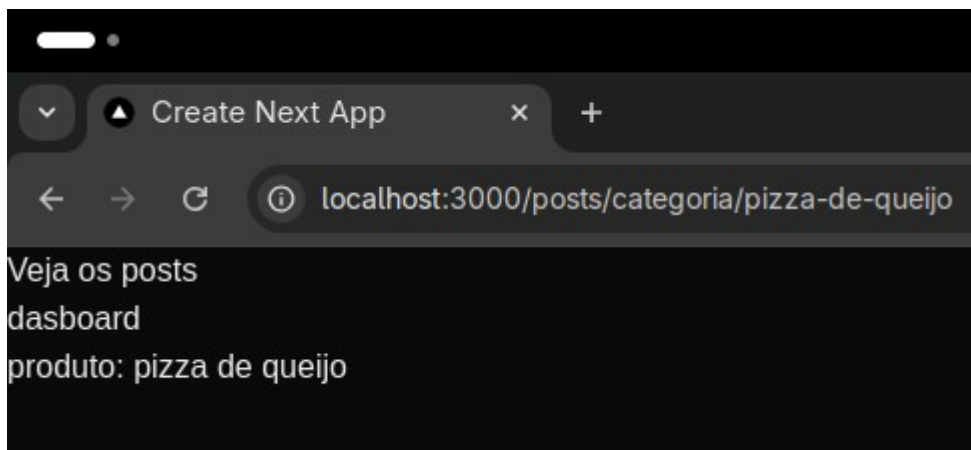
Como havia dito, as pastas viram páginas, mas existem pastas dinâmicas que o nome da pasta(página) irá ter um nome dinâmico. Para isso, você tem que nomear o nome da pasta entre colchetes

```
aula > src > app > posts > categoria > .js page.js > Categoria
1  import Link from "next/link";
2
3  export default function Categoria(){
4      const produtos = ['pizza de calabresa', 'pizza de queijo', 'pizza de frango'];
5      return(
6          <ul>
7              {produtos.map(p => (
8                  <li key={p}>
9                      <Link href={`./categoria/${p.replace(/ /g, "-")}`}>{p}</Link>
10                 </li>
11             ))}
12          </ul>
13      );
14  }
```

E para pegar o valor, você tem que usar função assíncronica(isso à partir do nextjs 15)

```
aula > src > app > posts > categoria > [produto] > .js page.js > produtos
1  export default async function produtos({params}){
2      const {produto} = await params
3      console.log(params.replace)
4      return(
5          <p>produto: {produto.replace(/-/g, " ")}</p>
6      );
7  }
```





OBS: pegar o paramentro é diferente que pegar o valor da query, você

Veja como funciona de baixo dos panos esse params:

```
✓ Compiled /posts/categoria/[produto] in 9ms
Promise {
  { produto: 'pizza-de-calabresa' },
  produto: 'pizza-de-calabresa',
  [Symbol(async_id_symbol)]: 42831,
  [Symbol(trigger_async_id_symbol)]: 42830,
  [Symbol(kResourceStore)]: {
    isStaticGeneration: false,
    page: '/posts/categoria/[produto]/page',
    fallbackRouteParams: null,
    route: '/posts/categoria/[produto]',
    incrementalCache: IncrementalCache {
      locks: Map(0) {},
      hasCustomCacheHandler: false,
      dev: true,
      disableForTestmode: false,
      minimalMode: false
    }
  }
}
```

Link ativos

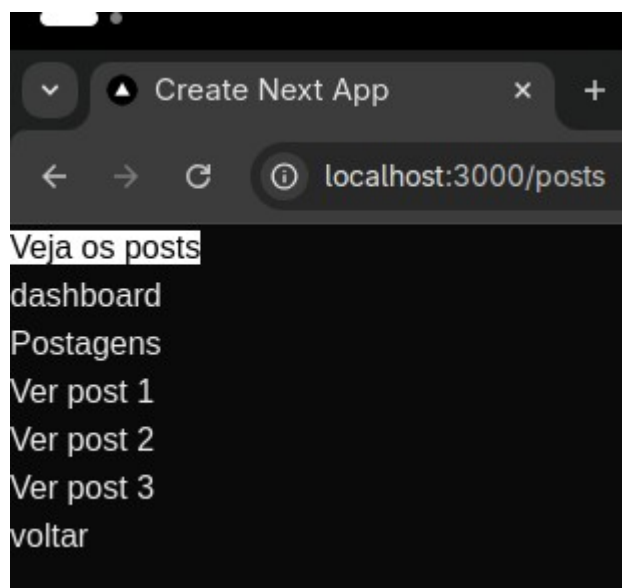
```
aula > src > components > header.js > Header
1  "use client"
2  import Link from "next/link";
3  import { usePathname, useRouter } from "next/navigation";
4
5  export default function Header() {
6    const path = usePathname();
7    const router = useRouter();
8    function click(){
9      router.push("/dashboard");
10   }
11   return (
12     <nav>
13       <ul>
14         <li>
15           <Link href="/posts" className={` ${path === "/posts" ? "active" : ""} `}>Veja os posts</Link>
16         </li>
17         <li>
18           <button onClick={click}>dashboard</button>
19         </li>
20       </ul>
21     </nav>
22   );
23 }
```


O `usePathname`, é um hook do Next.js (a partir da App Router) que **retorna o caminho atual da URL**, ou seja, a **parte da URL após o domínio, sem incluir os parâmetros de query**.

Ele é usado quando você quer saber **em que rota o usuário está atualmente**. Isso é muito útil, por exemplo, para:

- **Destacar o link do menu** que está ativo.
- Executar lógica condicional baseada no caminho da URL.
- Fazer redirecionamentos.
- Mostrar conteúdo específico dependendo da página atual.

Nesse caso, iremos usar para destacar o link do menu



Ai esse “active” é uma classe para esterelizar no CSS

useRouter

O `useRouter()` é usado **sempre que você precisa navegar entre páginas ou controlar a rota via código**, e **não** apenas com `<Link>`.

Enquanto o `<Link href="/alguma-rota">` serve para navegação **declarativa** (você clica e ele vai), o `useRouter()` é pra quando você quer fazer a navegação **programaticamente**, ou seja, via código. Ou seja, quando for fazer login

```
"use client"; // precisa disso para rodar no navegador

import { useRouter } from "next/navigation";


export default function Login() {
  const router = useRouter();

  function handleLogin() {
    // lógica de login...
    router.push("/dashboard"); // redireciona o usuário após o login
  }

  return (
    <button onClick={handleLogin}>
      Entrar
    </button>
  );
}
```


Redirect

Redirecionamento da pagina, você usa redirect

```
src > app > posts > [id] >  page.js > ...
import Link from "next/link";
import { redirect } from "next/navigation";
export default async function Page({ params }) {
  const { id } = await params
  const postsSaves = [1, 3]
  if(postsSaves.includes(Number(id))){
    return (
      <div>
        <h2>Olá</h2>
        <p>Você está na página {id}</p>
        <Link href="/posts">Voltar</Link>
      </div>
    );
  }else{
    redirect("/posts");
    return(
      <div>
        <h2>Not found</h2>
      </div>
    )
  }
};
```