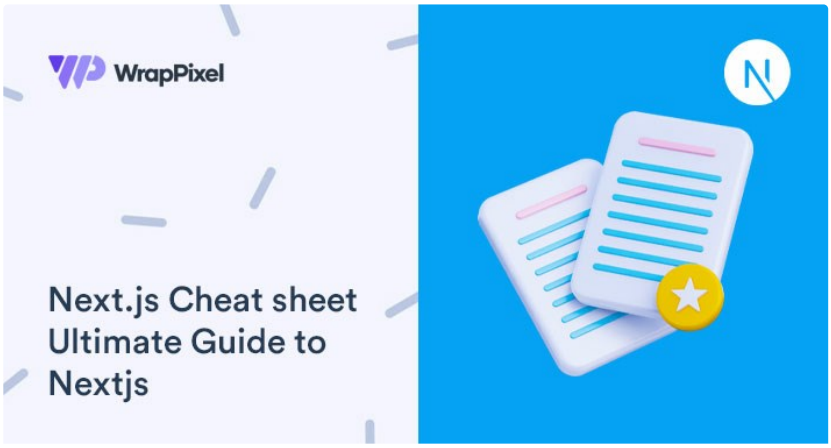
Folha de dicas NextJS: Guia final para NextJs

Por Sunil Joshi em 12 de julho de 2023 Folha de dicas, NextJs



Com um aumento cada vez mais crescente da popularidade dos NextJs, tornou-se um padrão bastante padrão para aplicativos da Web do React do lado do servidor. Então, isso significa que a folha de dicas do NextJS será seu principal ativo daqui para frente.

Ele obtém todos os recursos do React.js - a biblioteca de interface do usuário do JavaScript para criar componentes e adiciona tantos recursos adicionais que, às vezes, como desenvolvedor do NextJS, é difícil acompanhar os diferentes trechos de código, comandos para executar, pacotes para instalar.

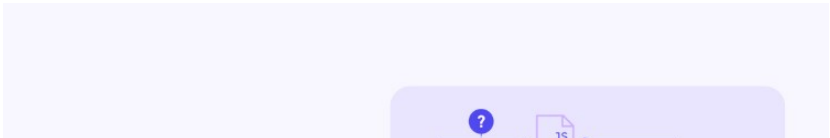
Seja você iniciante ou profissional experiente, esta folha de dicas servirá como um recurso valioso. Agora, vamos explorar os detalhes que podem aprimorar muito seus projetos NextJS. incorporados Esta folha de dicas do NextJS o ajudará se você estiver trabalhando em modelos de NextJS . Esta folha de dicas do NextJS será uma solução para todos os seus próximos problemas de codificação JS.

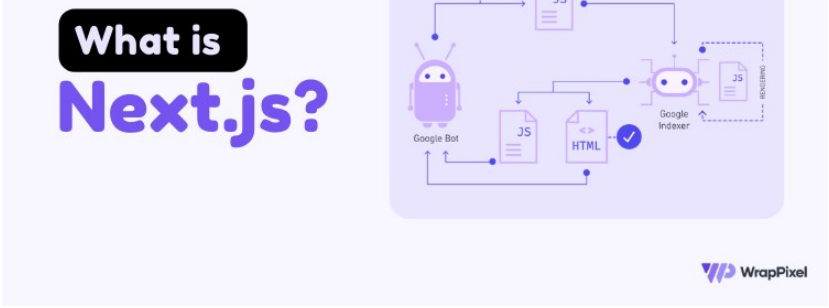
Para resolver esse problema, criamos uma folha de dicas do NextJS de ponto de todos os recursos que beneficiarão todos os desenvolvedores-sejam eles iniciantes ou profissionais. Vamos mergulhar!

O que é o NextJs?

Se não sabem o que é o próximojs . Então deixe -me dizer a você. De acordo com sua definição nos documentos oficiais:

- NextJS é uma **estrutura de reação** flexível que oferece blocos de construção para criar **aplicativos da Web** rápidos .





Basicamente, ele fornece alguns dos recursos cruciais e os blocos de construção em cima de um aplicativo Standard React.js para criar um site e aplicativos modernos.

Aqui estão algumas dessas características importantes (entre outras) que discutiremos:

Índice

Folha de dicas NextJs

Modernize Nuxt JS Admin Dashboard

⭐ 4.88 / 5.00

Categorias

Dado que abaixo está a próxima folha de dicas que você pode usar no seu próximo projeto.

Crie aplicativo usando o NextJS Chep Sheet

Nesta primeira etapa da folha de dicas do NextJS, criaremos um aplicativo NextJS, o processo recomendado é usar o comando Create-Next-App oficial que configura todos os arquivos, pastas e configuração necessários automaticamente.

```
npx create-next-app@latest
# OR
yarn create next-app
```

Em seguida, execute o NPM Run Dev ou Yarn Dev para iniciar o servidor de desenvolvimento local em http: // localhost: 3000.

Como alternativa, se você quiser instalar manualmente o NextJs, primeiro você deve instalar a seguir, reagir e reagir-dom em seu projeto como:

```
npm install next react react-dom
# OR
yarn add next react react-dom
```

Dentro do seu *arquivo package.json* , adicione os seguintes scripts:

```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint"
}
```

Use TypeScript, Eslint e NPM.

```
npx create-next-app --typeScript --eslint --use-npm
```

Criar páginas

Para criar uma página estática simples, no diretório do Páges, crie um arquivo chamado *Demo.js* que exporte um componente React:

```
function Demo() {
  return <h1>Demo</h1>
}
export default Demo
```

Esta página estará disponível em http: // localhost: 3000/demonstração do

seu ambiente local.

Roteador de aplicativos

Um componente-chave do Next.js foi o roteamento baseado no sistema de arquivos. Fornecemos esta ilustração de construir uma rota a partir de um único componente de reação em nossa postagem inicial:

```
// Pages Router

import React from 'react';
export default () => <h1>About us</h1>;
```

Nada mais necessário para ser configurado. Coloque um arquivo nas páginas e clique em Avançar. O restante seria tratado pelo roteador JS. Continuamos a adorar a simplicidade da rota. No entanto, à medida que a popularidade da estrutura aumentava, o mesmo aconteceu com os tipos de interfaces que os programadores queriam criar com ela.

Os desenvolvedores solicitaram melhor suporte de definição de layout, a capacidade de caminhar componentes da interface do usuário como layouts e aumentar a flexibilidade ao especificar os estados de carregamento e erro.

O roteador deve ser o ponto focal de toda a estrutura. Alterações de página, recuperação de dados, armazenamento em cache, modificação e revalidação de dados, streaming, estilo de conteúdo e muito mais.

```
// New: App Router 
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}
```

```
export default function Page() {
  return <h1>Hello, Next.js!</h1>;
}
```

```
// Pages Router

// This "global layout" wraps all routes. There's no way to
// compose other layout components, and you cannot fetch global
// data from this file.
export default function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />;
}
```

```
// New: App Router 
```

```
// The root layout is shared for the entire application
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}
```

```
// Layouts can be nested and composed
export default function DashboardLayout({ children }) {
  return (
    <section>
      <h1>Dashboard</h1>
      {children}
    </section>
  );
}
```

```
// Pages Router

// This file allows you to customize the <html> and <body> tags
// for the server request, but adds framework-specific features
// rather than writing HTML elements.
import { Html, Head, Main, NextScript } from 'next/document';

export default function Document() {
  return (
    <Html>
      <Head />
      <body>
        <Main />
        <NextScript />
      </body>
    </Html>
  );
}
```

```
// New: App Router 
// The root layout is shared for the entire application
export default function RootLayout({ children }) {
  return (
    <html lang="en">
      <body>{children}</body>
    </html>
  );
}
```

Muitas outras solicitações de recursos relevantes para o nosso sistema de

roteamento podem ser abordadas ao mesmo tempo em que tivemos a chance de projetar um novo roteador de sistema de arquivos. Por exemplo:

No passado, _app.js só podia importar folhas de estilo globais de pacotes externos do NPM (como bibliotecas de componentes). Não foi a melhor experiência de desenvolvimento. Qualquer arquivo CSS pode ser importado (e colocado) em qualquer componente usando o roteador de aplicativos.

Anteriormente, o uso do seu aplicativo foi restrito até que a página inteira fosse hidratada se você optar por usar a renderização do lado do servidor com o Next.js (através do GetServersideProps). Refatoramos a arquitetura com o roteador de aplicativos para ser totalmente integrado ao suspense do React, permitindo -nos hidratar seletivamente partes da página sem impedir que outros componentes da interface do usuário sejam interativos.

Turbopack

Através *do próximo dev - Turbo* e em breve, sua produção constrói (*Próxima compilação –Turbo*), Turbopack , nosso novo empuxo que estamos testando e estabilizando no Next.js, ajuda a acelerar as iterações locais enquanto trabalhamos no seu próximo.js.

Testemunhamos um aumento constante de popularidade desde a liberação alfa do Next.js 13, pois trabalhamos para corrigir problemas e adicionar suporte por falta de funcionalidade. Para obter feedback e aumentar a estabilidade, estamos testando o Turbopack em vercel.com e com vários clientes da Vercel, executando sites substanciais do Next.js. Agradecemos a assistência da comunidade em testar e alertar nossa equipe sobre bugs.

Agora estamos preparados para avançar para a fase beta após seis meses.

Webpack e Next.js atualmente têm mais recursos do que o Turbopack. O suporte a essas funcionalidades está sendo rastreado nesta edição . A maioria dos casos de uso deve ser apoiada hoje, no entanto. Para continuar abordando bugs de maior adoção e se preparam para a estabilidade em uma versão futura, estamos lançando esta versão beta.

O motor incremental e a camada de cache do Turbopack estão sendo aprimorados, e isso acabará acelerando não apenas o desenvolvimento local, mas também as construções de produção. Fique de olho em um lançamento futuro do Next.js que permitirá que você execute *a próxima compilação –Turbo* para construções instantâneas.

Use *o próximo dev –turbo* para testar o Turbopack Beta em Next.js 13.4.

API de metadados baseada em arquivo

Ao exportar um objeto de metadados a partir de um layout ou página, você pode definir metadados (como *título* , *meta* e *link* tags dentro do seu *elemento de cabeça* HTML) usando a nova API de metadados que foi introduzida no Next.js 13.2.

```
// either Static metadata
export const metadata = {
  title: 'Home',
};
// Output:
// <head>
//  <title>Home</title>
// </head>

// or Dynamic metadata
export async function generateMetadata({ params, searchParams }) {
  const product = await getProduct(params.id);
  return { title: product.title };
}
// Output:
// <head>
//  <title>My Unique Product</title>
// </head>

export default function Page() {}
```

A API de metadados agora suporta novas convenções de arquivos, além dos metadados baseados em configuração, simplificando alterar rapidamente suas páginas para melhor SEO e compartilhamento on-line:

OpenGraph-Image. (jpg | png | svg)
Imagem do Twitter. (jpg | png | svg)
ícone Favicon.ICO. (ico | jpg | png | svg)
Sitemap. (xml | js | jsx | ts | tsx)
robôs. (txt | js | jsx | ts | tsx)
manifesto. (JSON | JS | JSX | TS | TSX)

Imagens de gráficos abertos dinâmicos

Na conferência Next.JS, @Vercel/OG foi posta à prova, criando mais de 100.000 fotos dinâmicas para cada hóspede. Estamos entusiasmados em introduzir imagens geradas dinamicamente em todos os aplicativos Next.js sem o requisito de um pacote externo, graças à adoção generalizada entre clientes da Vercel e mais de 900.000 downloads desde o lançamento.

Para criar imagens, agora você pode importar imagens -resposta do próximo/servidor:

```
import { ImageResponse } from 'next/server';

export const size = { width: 1200, height: 600 };
export const alt = 'About Acme';
export const contentType = 'image/png';
export const runtime = 'edge';
```

```
export default function og() {
  return new ImageResponse();
  // ...
}
```

Os manipuladores de rota e os metadados baseados em arquivos são apenas duas das próximas *APIs* . Por exemplo, você pode criar fotos abertas e twitter no horário de construção ou dinamicamente no tempo de solicitação usando *o ImageResponse* em um *arquivo openGraph-image.tsx* .

Exportação estática para roteador de aplicativos

As exportações totalmente estáticas agora são suportadas pelo roteador App Next.js.js.

Um site estático ou um aplicativo de página única (SPA) pode ser usada como ponto de partida e você pode mais tarde atualizar para empregar os recursos do Next.js que exigem um servidor.

Como parte do próximo processo de compilação, o Next.js cria um arquivo HTML para cada rota. Um spa rigoroso pode ser dividido em arquivos HTML separados com a ajuda do Next.js para impedir que o código JavaScript extra seja carregado no lado do cliente, reduzindo assim o tamanho do pacote e permitindo que a página mais rápida seja carregada.

```
/**
 * @type {import('next').NextConfig}
 */
const nextConfig = {
  output: 'export',
};

module.exports = nextConfig;
```

Os *novos recursos do roteador de aplicativos* , como manipuladores de rota estáticos, imagens gráficas abertas e componentes do servidor React, são compatíveis com a exportação estática.

Como a geração tradicional do site estático, os componentes do servidor, por exemplo, serão executados durante toda a compilação e renderizarão os componentes em HTML estático para a carga inicial da página e uma carga estática para o movimento do cliente nas rotas.

Antes, você precisava executar a próxima exportação antes de usar a exportação estática no *diretório de páginas* . No entanto, quando a saída: ' *exportação* ' estiver definida, a próxima compilação produzirá um diretório out graças à *opção Next.config.js* . O diretório de roteador e páginas do aplicativo pode ser configurado da mesma maneira. Portanto, a exportação subsequente não é mais necessária.

Rotas e interceptação paralelos

Rotas paralelas e rotas de interceptação são duas novas convenções dinâmicas que o Next.js 13.3 apresenta para ajudá-lo a criar cenários de roteamento complexos. Com a ajuda desses recursos, você pode exibir várias páginas na mesma visão, como painéis ou modais complexos.

Você pode renderizar simultaneamente uma ou mais páginas na mesma visão que podem ser acessadas separadamente usando rotas paralelas. Também pode ser usado para renderizar páginas condicionalmente.

Nomeados "slots" são usados para criar rotas paralelas. De acordo com a convenção @Folder, os slots são definidos:


```
dashboard
├── @user
│   └── page.js
├── @team
│   └── page.js
├── layout.js
└── page.js
```

```
export default async function Layout({ children, user, team }) {
  const userType = getCurrentUserType();

  return (
    <>
      {userType === 'user' ? user : team}
      {children}
    </>
  );
}
```

Os *slots de rota paralela @User* e *@Team* (explícitos) no exemplo acima mencionado são produzidos condicionalmente com base no seu raciocínio. As crianças implícitas de slot de rota não requerem mapeamento para uma *@Folder* . Dashboard/Page.js, por exemplo, é idêntico ao painel/ *@children* /page.js.

Ao "mascarar" o URL do navegador, você pode carregar uma nova rota no layout existente interceptando rotas. Quando o contexto da página atual deve ser preservado, como ao expandir uma foto em um feed através de um modal enquanto o feed é mantido no fundo do modal, isso é útil.

Estilize seu aplicativo NextJS

Existem muitas maneiras de estilizar seus aplicativos nesta etapa da folha de dicas do NextJS, alguns dos métodos comuns são:

- **Estilo global** : importe os *estilos* globais.css em suas *páginas/_app.js,* onde esses estilos se aplicarão a todas as páginas e componentes do seu aplicativo como:

```
import '../styles.css'
```

```
export default function MyApp({ Component, pageProps }) {
  return <Component {...pageProps} />
}
```

- **CSS em nível de componente** : o NextJS suporta módulos CSS , onde você pode nomear os arquivos como *[nome] .module.css* e depois importá-los em um componente específico. Aqui está um exemplo:

```
// Button.module.css
.error {
  color: white;
  background-color: red;
}
// Button.jsx
import styles from './Button.module.css'
export function Button() {
  return (
    <button
      type="button"
      className={styles.error}
    >
      Cancel
    </button>
  )
}
```

- **Usando SASS** : Primeiro, você precisa instalar o pacote SASS em seu aplicativo NextJS:

```
npm install --save-dev sass
```

Em seguida, você pode configurar as opções do compilador SASS no arquivo next.config.js:

```
const path = require('path')
module.exports = {
  sassOptions: {
    includePaths: [path.join(__dirname, 'styles')],
  },
}
```

Otimize imagens e fontes

Nesta etapa da folha de dicas do NextJS, para otimizar as imagens, você deve usar o componente de imagem embutido. Instale -o em seu projeto como:

```
import Image from 'next/image'
```

Em seguida, dê um atributo SRC, como mostrado no exemplo a seguir:

```
import Image from 'next/image'
const myLoader = ({ src, width, quality }) => {
  return `https://example.com/${src}?w=${width}&q=${quality || 7
}
const MyImage = (props) => {
  return (
    <Image
      loader={myLoader}
      src="me.png"
      alt="Picture of the author"
      width={500}
      height={500}
    />
  )
}
```

404 páginas personalizadas

Para criar 404 página personalizada, crie um `404.js` arquivo no `pages` pasta

```
export default function Custom404() {
  return <h1>404 - Page Not Found</h1>
}
```

Você também pode criar um `500.js` Arquivo para o erro do servidor 500 página

SWR

É uma biblioteca React Hooks para busca de dados remotos no cliente

Você pode usá -lo no local de `useEffect`

```
import useSWR from 'swr'

export default function Home() {
  const { data, error } = useSWR('api/user', fetch)

  if (error) return <div>failed to load</div>
  if (!data) return <div>loading...</div>

  return (
    <>
      {data.map((post) => (
        <h3 key={post.id}>{post.title}</h3>
      ))}
    </>
  )
}
```

Buscar dados

Na folha de dicas NextJS, há muitas maneiras de buscar dados de fontes externas para o seu aplicativo NextJS, aqui estão alguns:

- **GetServersideProps** : se você deseja que seu aplicativo prenda uma página em cada solicitação, a função GetServerSideProps deve ser exportada assim:

```
export async function getServerSideProps(context) {
  return {
    props: {},
  }
}
```

Aqui está um exemplo para buscar dados no horário da solicitação, que pré-renderize o resultado que ele retira da fonte de dados:

```
function Page({ data }) {
  // Code to render the `data`
}
export async function getServerSideProps() {
  const res = await fetch(`https://.../data`)
  const data = await res.json()
  return { props: { data } }
}
export default Page
```

- **GetStaticpaths** : se você deseja gerar rotas dinamicamente em seu aplicativo, juntamente com o GetStaticProps, o GetStaticSpaths irá renderizar todos os caminhos fornecidos a ele como:

```
export async function getStaticPaths() {
  return {
    paths: [
      { params: { ... } }
    ],
    fallback: true
  };
}
```

- **GetStaticProps** : Se você deseja que o NextJS gere uma página no horário de construção usando os adereços passados, o GetStaticProps deve ser exportado como:

```
export async function getStaticProps(context) {
  return {
    props: {},
  }
```

```
}
```

Observe que os adereços aqui devem ser passados para o componente da página como adereços. Um exemplo de seu uso quando você deseja que os dados busquem de um CMS é o seguinte:

```
function BlogPosts ({ posts }) {
  return (
    <>
      {posts.map((post) => (
        <h1>{post.title}</h1>
        <p>{post.summary}</p>
      ))}
    </>
  )
}
export async function getStaticProps() {
  const res = await fetch('https://.../posts')
  const posts = await res.json()
  return {
    props: {
      posts,
    },
  }
}
export default BlogPosts
```

- **Regeneração estática incremental (ISR)** : Se você deseja criar ou atualizar páginas estáticas existentes *após* a criação do seu site, o ISR permite gerar estaticamente por página por página. Isso significa que agora você não precisa reconstruir todo o site do zero.

Para que isso funcione, você só precisa adicionar o suporte do Revalidate ao método GetStaticProps:

```
export async function getStaticProps(context) {
  return {
    props: {},
    revalidate: 5 // this means the request to re-generate the p
  }
}
```

- **Busque dados no lado do cliente** : isso pode ser feito de duas maneiras diferentes-seja através do gancho de uso do uso como:

```
function User() {
  const [data, setData] = useState(null)
  const [isLoading, setLoading] = useState(false)
  useEffect(() => {
    setLoading(true)
```

```
    fetch('api/user-data')
      .then((res) => res.json())
      .then((data) => {
        setData(data)
        setLoading(false)
      })
  }, [])
  if (isLoading) return <p>Loading user data...</p>
  if (!data) return <p>No user data found</p>
  return (
    <div>
      <h1>{data.name}</h1>
      <p>{data.bio}</p>
    </div>
  )
}
```

Ou através da biblioteca SWR , que lida com o cache, a reavalidação, o rastreamento de foco, a retirada de intervalos e mais como:

```
import useSWR from 'swr'
const fetcher = (...args) => fetch(...args).then((res) => res.js
function User() {
  const { data, error } = useSWR('/api/user-data', fetcher)
  if (error) return <div>Failed to load user data</div>
  if (!data) return <div>Loading user data...</div>
  return (
    <div>
      <h1>{data.name}</h1>
      <p>{data.bio}</p>
    </div>
  )
}
```

Linhando seu código

Você pode usar o ESLINT pronto para uso para linha. Basta adicionar o seguinte script ao *arquivo package.json* :

```
"scripts": {
  "lint": "next lint"
}
```

Agora você pode executar fiapos de fiagem ou fios do NPM para iniciar o linhador. Se você estiver usando o Eslint em um monorepo, onde o NextJS não está instalado no seu diretório raiz, basta adicionar o rootdir ao seu *arquivo .eslinTrc* :

```
{
```

```
  "extends": "next",
  "settings": {
    "next": {
      "rootDir": "packages/my-app/"
    }
  }
}
```

Para usar mais bonito com configurações de ESLint, primeiro instale a dependência:

```
npm install --save-dev eslint-config-prettier
# OR
yarn add --dev eslint-config-prettier
```

E, em seguida, adicione mais bonito ao seu arquivo de configuração de ESLint existente:

```
{
  "extends": ["next", "prettier"]
}
```

Suporte TypeScript

O suporte ao TypeScript também é uma das próximas folhas de trapaça. Para usar o TypeScript com o NextJS ao iniciar um aplicativo, use o comando Create-Next-App junto com o sinalizador –TS ou –TypeScript:

```
npx create-next-app@latest --ts
# or
yarn create next-app --typescript
```

Isso aumentará um novo projeto NextJS com todos os arquivos e componentes do TypeScript sem nenhuma configuração extra.

Mas se você deseja integrar o TypeScript em um projeto existente, crie um novo *arquivo tsconfig.json* na raiz do diretório do projeto. Em seguida, execute o NPM Run Dev ou o Yarn Dev, com este NextJS o guiará através da instalação dos pacotes necessários para concluir a configuração do TypeScript Integration.

Usando scripts

Nós vamos usar scripts nesta etapa da folha de dicas do NextJS, o elemento HTML <script> nativo é substituído pelo componente Next/Script no NextJs. Aqui está um exemplo de carregar um script do Google Analytics:

```
import Script from 'next/script'
export default function Home() {
  return (
    <>
```

```
      <Script src="https://www.google-analytics.com/analytics.js
    </>
  )
}
```

Primeiro, você importa o componente de script:

```
import Script from 'next/script'
```

Em seguida, existem maneiras diferentes de lidar com scripts com este componente que pode ser definido pela propriedade da estratégia com um dos três valores a seguir:

1. antes do interativo: carregue o script antes que a página seja interativa.
2. APÓSIRATIVO: Carregue o script imediatamente após a página se tornar interativa.
3. LAZYONLOAD: Carregue o script durante o tempo ocioso.

Aqui está um exemplo:

```
<script
  src="https://cdn.jsdelivr.net/npm/cookieconsent@3/build/cookie
  strategy="beforeInteractive"
/>
```

Roteamento no nível do aplicativo

Na próxima etapa da NextJS Cheat Sheet, o NextJS possui um roteador baseado no sistema de arquivos que funciona no conceito de páginas. Você pode ter rotas de índice como *páginas/index.js* que mapeiam para/e *páginas/blog/index.js* que mapeiam para/blog.

Ou você pode ter rotas aninhadas, onde ele suporta arquivos aninhados. Por exemplo, um arquivo localizado nas *páginas/blog/my-post.js* irá percorrer/blog/my-post.

Para rotas dinâmicas, você precisa usar a sintaxe do suporte para que ela possa corresponder aos parâmetros nomeados. Por exemplo, *páginas/[nome de usuário] /settings.js* mapeará para/johndoe/configurações.

O componente <ink> é usado para fazer transições de rota do lado do cliente. Primeiro, você precisa importá -lo como:

```
import Link from 'next/link'
```

Em seguida, use -o em um componente:

```
import Link from 'next/link'
function Home() {
  return (
    <ul>
      <li>
        <Link href="/">
```

```
          <a>Home</a>
        </Link>
      </li>
      <li>
        <Link href="/about">
          <a>About Us</a>
        </Link>
      </li>
      <li>
        <Link href="/blog/hello-world">
          <a>Blog Post</a>
        </Link>
      </li>
    </ul>
  )
}
export default Home
```

Para caminhos dinâmicos, você pode usar a interpolação da string para criar o caminho desejado:

```
import Link from 'next/link'
function Posts({ posts }) {
  return (
    <ul>
      {posts.map((post) => (
        <li key={post.id}>
          <Link href={`/blog/${encodeURIComponent(post.slug)}`}>
            <a>{post.title}</a>
          </Link>
        </li>
      ))}
    </ul>
  )
}
export default Posts
```

Roteamento da API

Qualquer arquivo dentro da *pasta Páginas/API* é mapeado para/api/* que será tratado como um terminal de API. Por exemplo, para retornar uma resposta JSON com um código de status OK de 200, você pode exportar uma função de manipulador com o Req e o RES Passado como parâmetros:

```
export default function handler(req, res) {
  res.status(200).json({ name: 'John Doe' })
}
```

Para lidar com diferentes métodos HTTP, você pode usar o req.method em

seu manipulador de solicitação:

```
export default function handler(req, res) {
  if (req.method === 'POST') {
    // Process a POST request
  } else {
    // Handle any other HTTP method
  }
}
```

Middlewares

1. Para usar o Middlewares no NextJs, primeiro instale a versão mais recente do Next:

```
npm install next@latest
```

2. Em seguida, crie um arquivo _middleware.ts dentro do seu */páginas* diretório
3. Por fim, exporte uma função de middleware do mesmo arquivo:

```
import type { NextFetchEvent, NextRequest } from 'next/server'
export function middleware(req: NextRequest, ev: NextFetchEvent)
  return new Response('Hello, world!')
}
```

Por exemplo, aqui está um exemplo em que o middleware é usado para registrar:

```
import { NextRequest } from 'next/server'
// Regex for public files
const PUBLIC_FILE = /\.(.*)$/
export default function middleware(req: NextRequest) {
  // Only log for visited pages
  if (!PUBLIC_FILE.test(req.nextUrl.pathname)) {
    // We fire and forget this request to avoid blocking the req
    // and let logging occur in the background
    fetch('https://in.logtail.com', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        Authorization: `Bearer ${process.env.LOGTAIL_TOKEN}`,
      },
      body: JSON.stringify({
        message: 'Log from the edge',
        nested: {
          page: req.nextUrl.href,
          referrer: req.referrer,
          ua: req.ua?.ua,
```

```
          geo: req.geo,
        },
      }),
    })
  }
}
```

Autenticação

Existem muitas maneiras diferentes de autenticar um usuário em um aplicativo NextJS. Alguns dos comuns são:

1. **Autenticando as páginas geradas estaticamente** : Aqui, sua página pode renderizar um estado de carregamento do servidor, após o qual buscará os dados do usuário do lado do cliente. No exemplo a seguir, a página renderiza um estado de esqueleto de carregamento e, uma vez que a solicitação é atendida, ele mostra o nome do usuário:

```
import useUser from '../lib/useUser'
import Layout from '../components/Layout'
const Profile = () => {
  // Fetch the user client-side
  const { user } = useUser({ redirectTo: '/login' })
  // Server-render loading state
  if (!user || user.isLoggedIn === false) {
    return <Layout>Loading...</Layout>
  }
  // Once the user request finishes, show the user
  return (
    <Layout>
      <h1>Your Profile</h1>
      <pre>{JSON.stringify(user, null, 2)}</pre>
    </Layout>
  )
}
export default Profile
```

2. **Autenticando as páginas renderizadas pelo servidor** : Aqui você precisa exportar uma função assíncroada getServersideProps () de uma página pela qual o NextJS fará pré-renderizar esta página em cada solicitação. Aqui está um exemplo em que, se houver uma sessão, o usuário será retornado como um suporte para o componente do perfil:

```
import withSession from '../lib/session'
import Layout from '../components/Layout'
export const getServerSideProps = withSession(async function ({
  const { user } = req.session
  if (!user) {
    return {
```

```
      redirect: {
        destination: '/login',
        permanent: false,
      },
    }
  }
  return {
    props: { user },
  }
})
const Profile = ({ user }) => {
  // Show the user. No loading state is required
  return (
    <Layout>
      <h1>Your Profile</h1>
      <pre>{JSON.stringify(user, null, 2)}</pre>
    </Layout>
  )
}
export default Profile
```

3. **Autenticando com provedores de terceiros** : para provedores de autenticação comuns como Auth0 , FireBase , Supabase , etc., você pode dar uma olhada no repositório oficial do GitHub para obter exemplos de como configurar e configurar seu próprio aplicativo NextJS.

Teste

A última etapa da folha de dicas do NextJS, assim como na autenticação, os testes podem ser feitos de várias maneiras diferentes e com diferentes ferramentas de teste. Veja como configurar testes com ferramentas comuns:

1. **Testando com o Cypress** : comece com o exemplo do compressor para iniciar rapidamente um aplicativo NextJS com o Cypress como:

```
npx create-next-app@latest --example with-cypress with-cypress-a
```

Ou manualmente, instale o pacote Cypress:

```
npm install --save-dev cypress
```

Em seguida, adicione -o ao campo Scripts do seu *arquivo package.json* :

```
"scripts": {
  ...
  "cypress": "cypress open",
}
```

Finalmente, execute o cipreste com o seguinte comando:

```
npm run cypress
```

Para criar um arquivo de teste do Cypress. Basta criar um arquivo sob *Cypress/Integration/App.spec.js* como:

```
describe('Navigation', () => {
  it('should navigate to the about page', () => {
    // Start from the index page
    cy.visit('http://localhost:3000/')
    // Find a link with an href attribute containing "about" and
    cy.get('a[href*="about"]').click()
    // The new url should include "/about"
    cy.url().should('include', '/about')
    // The new page should contain an h1 with "About page"
    cy.get('h1').contains('About Page')
  })
})
```

2. **Testando com a biblioteca de testes de brincadeira e reação** : novamente você pode usá-lo rapidamente com a comunidade fornecida com exemplo de jest enquanto você gira um novo próximo projeto:

```
npx create-next-app@latest --example with-jest with-jest-app
```

Ou manualmente, você pode instalar o Jest, React Testing Library e Jest Dom Pacotes:

```
npm install --save-dev jest @testing-library/react @testing-libr
```

Em seguida, crie um *arquivo jest.config.js* no diretório raiz do projeto:

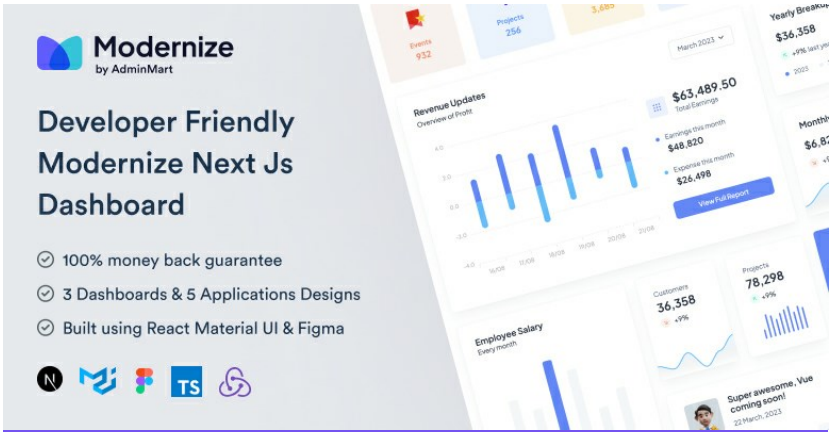
```
const nextJest = require('next/jest')
const createJestConfig = nextJest({
  // Provide the path to your NextJs app to load next.config.js
  dir: './',
})
// Add any custom config to be passed to Jest
const customJestConfig = {
  // Add more setup options before each test is run
  // setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  // if using TypeScript with a baseUrl set to the root director
  moduleDirectories: ['node_modules', '<rootDir>/'],
  testEnvironment: 'jest-environment-jsdom',
}
// createJestConfig is exported this way to ensure that next/jes
module.exports = createJestConfig(customJestConfig)
// Add a test script to the package.json file:
"scripts": {
  ...
  "test": "jest --watch"
```

```
}
// Then create a Jest test file under __tests__/index.test.jsx a
import { render, screen } from '@testing-library/react'
import Home from '../pages/index'
describe('Home', () => {
  it('renders a heading', () => {
    render(<Home />)
    const heading = screen.getByRole('heading', {
      name: /welcome to next\.js!/i,
    })
    expect(heading).toBeInTheDocument()
  })
})
```

Para executar o teste, basta executar o comando NPM Run Test.

Neste artigo, você conheceu a próxima folha de dicas do NextJS e como isso ajuda a criar sites e aplicativos modernos baseados em reagir. Em seguida, você viu como configurar um projeto NextJS, como buscar dados, otimizar ativos, adicionar suporte ao TypeScript, usar linters, integrar ferramentas de teste e muito mais!

Para aqueles que priorizam os modelos incorporados (modernize o modelo de administrador gratuito do NextJS) sobre os modelos personalizados. Esta folha de dicas é para você. NextJS GRATUITO Dado abaixo, está a imagem Modernize o modelo de administração :



Conclusão

Em conclusão, ter uma folha de dicas do NextJS pode beneficiar bastante os desenvolvedores trabalhando com essa estrutura poderosa. O fornecimento de um guia de referência rápido para os recursos e funcionalidades mais usados permite que os desenvolvedores trabalhem de maneira mais eficiente e eficaz.

Uma folha de dicas do NextJS serve como uma ferramenta útil para iniciantes e desenvolvedores experientes. Ajuda a reduzir o tempo gasto em busca de detalhes de sintaxe ou configuração, permitindo que os desenvolvedores se concentrem mais na escrita de código limpo e otimizado.

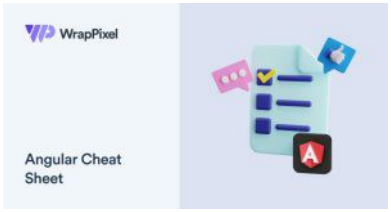
Além disso, uma folha de dicas também pode ajudar a melhorar a qualidade

do código, promovendo as melhores práticas e destacando armadilhas comuns a serem evitadas. Ele atua como um companheiro confiável, garantindo que os desenvolvedores sigam as diretrizes recomendadas ao desenvolver seus aplicativos NextJS.

No geral, ter acesso a uma folha de dicas abrangentes do NextJS é inestimável para qualquer desenvolvedor que deseje otimizar seu fluxo de trabalho e aprimorar sua produtividade. Esteja você construindo pequenos projetos ou aplicativos em larga escala, esse recurso, sem dúvida, será um ativo essencial no seu kit de ferramentas de desenvolvimento.

Postagens relacionadas

NUXT Cheat Sheet & Nuxt.js Essentials



Folha de dicas angulares 2023

Autor

Sunil Joshi
Co-fundador da Wrappixel

Sunil Joshi é um ávido desenvolvedor de designers e apaixonado por resolver desafios complexos de UX entre os negócios digitais. Ele é um criador de tendências no campo da visualização de dados e design do painel e foi elogiado por sua estética de design limpo e minimalista. Ele co-fundou o Wrappixel, o mercado de design em 2016 com o objetivo de trazer ótimo design e código limpo a um alcance fácil de todos.

Deixe uma resposta

Seu endereço de e -mail não será publicado. Os campos necessários estão marcados *

Nome*

Digite seu nome completo aqui

E-mail*

Insira o email aqui

Salve meu nome, email e site neste navegador para a próxima vez que eu comentar.

Message

Write message here

Post Comment

Company

Why Wrappixel
Affiliate Program
Blog

Help & Wpblog

Contact Us
Premium Wpblog
Custom Development

Legal Information

Licenses
Terms & Conditions
Privacy Policy

446,528
Customer

489,073
Downloads

Admin | Premium Themes | UI Kits