



Next.js

Prof. Vicente de Araujo Calfo

Descrição

Construção de aplicações SPA (single page application) com os principais fundamentos e conceitos do framework Next.js.

Propósito

O crescente avanço do JavaScript como linguagem de programação dominante da camada de front-end estimula a utilização de frameworks e bibliotecas para garantir escalabilidade e qualidade do código. Nesse cenário, o framework Next.js agiliza o processo de construção de aplicações que utilizam a biblioteca ReactJS, se tornando uma das tecnologias mais empregadas atualmente no desenvolvimento de aplicações front-end.

Preparação

Antes de começar, certifique-se de que esteja utilizando um computador com no mínimo 4GB de RAM, 15GB de espaço em disco disponível, sistema operacional Windows, MacOS ou Linux e um editor de código (preferencialmente o VSCode). Tenha acesso ainda a uma Internet de banda larga para que você possa realizar os exercícios sem problema.

Objetivos

Módulo 1

Fundamentos de Next.js

Construir uma aplicação mediante o uso do Next.js.

Módulo 2

Módulos de pré-renderização do Next.js

Aplicar a funcionalidade do Next.js de renderização estática.

Módulo 3

Deploy na nuvem

Desenvolver uma aplicação desenvolvida em Next.js em uma infraestrutura de nuvem.

Módulo 4

Migrando o Next.js para TypeScript

Transferir a aplicação Next.js para TypeScript com as funcionalidades do JavaScript.



Introdução

O JavaScript é certamente a linguagem de programação mais importante na atualidade quando se fala de aplicações modernas, compondo, assim, a tríade do desenvolvimento Web front-end junto a HTML e CSS. Por isso, a busca por qualidade de código e escalabilidade estimulou a criação de inúmeros frameworks e bibliotecas que usam o JavaScript como base.

Neste conteúdo, você vai aprender os fundamentos e as principais funcionalidades do framework Next.js, além de relembrar alguns conceitos importantes da biblioteca usada como base: o ReactJS.

Ao longo dos tópicos, você acompanhará o processo de construção de uma aplicação front-end capaz de se manter simples para manutenção e flexível para escalar quando o tamanho do projeto aumenta. Tudo isso será feito utilizando componentes de rota e suporte à renderização do lado do servidor (SSR), aplicando o TypeScript em seu código-fonte e, por fim, implantando-a em um servidor na nuvem.



1 - Fundamentos de Next.js

Ao final deste módulo, você será capaz de construir uma aplicação mediante o uso do Next.js.

Por que usar o Next.js

Quando pensamos em tecnologia, não devemos acreditar que existe a chamada “bala de prata”, isto é, uma solução perfeita e que sirva para solucionar qualquer problema. Por causa disso, é importante avaliar minuciosamente o problema e as soluções disponíveis a fim de que, dentro das justificativas técnicas, se possa escolher a melhor opção.

O **ReactJS** é a **biblioteca de base usada pelo Next.js** para construir toda interface de usuário (UI), como mostraremos na imagem a seguir. Essa biblioteca fornece inúmeras funções que ajudam na construção da UI da aplicação, deixando o desenvolvedor mais livre para construir apenas as funções pertinentes à aplicação.

No entanto, construir uma aplicação inteira no ReactJS requer uma série de configurações e serviços, os quais, apesar de serem necessários, consomem tempo de desenvolvimento.



O emprego do Next.js em seu projeto não difere dessa regra. Ele é um framework que vem ganhando o mercado, sendo adotado por inúmeras empresas.

Mas exatamente em que essa solução ajuda no desenvolvimento de aplicações front-end? Para podermos responder a essa questão, vamos conhecer adiante as **principais funcionalidades do Next.js**, as quais, aliás, são as principais responsáveis por seu atual sucesso:



Renderização no servidor (SSR)

As telas são construídas no lado do servidor, entregando todos os arquivos prontos para o cliente (navegador);



Geração de site estático (SSG)

Os dados de uma API ou de documentos de texto são pré-processados, criando, com isso, os arquivos estáticos (HTML, CSS e JavaScript) a serem entregues para o cliente (navegador);



CSS-in-JS

Estrutura em Styled-JSX pronta para estilizar a marcação de HTML, embora ela seja flexível para a utilização de outras soluções, como, por exemplo, styled components;



Configuração mínima

O framework já está praticamente configurado para utilização de rotas e hot reloading, reduzindo quase a zero a intervenção do desenvolvedor;



Suporte ao TypeScript.

SSR

Server side rendering

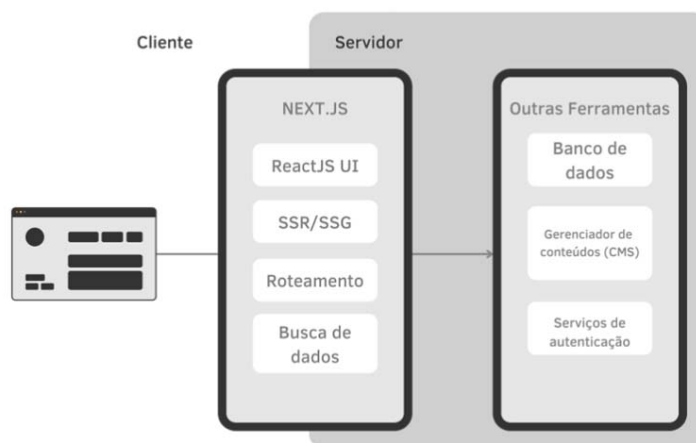
SSG

Static site generation

API

Application programming interface

O NEXT.JS usa as funcionalidades anteriores para fornecer blocos de construção que lidam com as configurações necessárias para o ReactJS, criando uma estrutura predefinida, os recursos e as otimizações para a aplicação em desenvolvimento.



Algumas funcionalidades do Next.js serão percebidas apenas em produção, pois ele ajuda na performance da aplicação e na indexação de conteúdo pelos motores de busca, o que sempre foi um dos principais problemas das SPAs.

Instalando o Next.js

Todo conteúdo apresentado a partir de agora assume que você tenha conhecimento de HTML, CSS, JavaScript e ReactJS.

Dica

Para começar a desenvolver uma aplicação em Next.js, é preciso ter instalado o Node.js versão 12.22.0 ou posterior. Você provavelmente já deve ter feito essa instalação, pois ela foi necessária no seu curso de ReactJS; no entanto, pode seguir o passo a passo da instalação do Node.js disponível em seu site oficial: <https://nodejs.org/en/download/>

A maneira mais indicada para se começar uma aplicação é por meio da configuração automática, pois, com apenas uma linha de comando, o Next.js realizará todas as configurações necessárias para você. No seu terminal, digite “npx create-next-app@latest” para criar uma aplicação usando a última versão estável disponível do Next.js.



```
usuario@usuario:~/Área de Trabalho$ npx create-next-app
Need to install the following packages:
  create-next-app
Ok to proceed? (y) y
✓ What is your project named? _ nome-da-app
Creating a new Next.js app in /home/Área de Trabalho/nome-da-app.

Using yarn.

Installing dependencies:
- react
- react-dom
- next

yarn add v1.22.17
info No lockfile found.
[1/4] Resolving packages...
[2/4] Fetching packages...
[3/4] Linking dependencies...
[4/4] Building fresh packages...

success Saved lockfile.
info To upgrade, run the following command:
$ curl --compressed -o- -L https://yarnpkg.com/install.sh | bash
success Saved 14 new dependencies.
info Direct dependencies
├─ next@12.1.5
├─ react-dom@18.0.0
└─ react@18.0.0
info All dependencies
├─ @next/env@12.1.5
├─ @next/swc-linux-x64-gnu@12.1.5
├─ @next/swc-linux-x64-musl@12.1.5
├─ caniuse-lite@1.0.30001332
├─ js-tokens@4.0.0
├─ nanoid@3.3.3
├─ next@12.1.5
├─ picocolors@1.0.0
├─ postcss@8.4.5
├─ eslint
└─ eslint-config-next
.....
```

Após a execução do comando da figura, você precisa apenas informar o nome da aplicação. Com isso, o próprio Next.js se encarregará de instalar todas as dependências necessárias para o funcionamento – inclusive o próprio ReactJS.

Ao término da instalação, você tem de seguir os seguintes passos:

- Execute “npm run dev” para iniciar o servidor de desenvolvimento em <http://localhost:3000>;

- Visite <http://localhost:3000> para ver a tela inicial padrão do projeto Next.js.

O Next.js vai criar uma estrutura de diretórios básica para você começar a desenvolver a aplicação. A instalação básica será criada com três importantes diretórios:




Um arquivo importante que o Next.js gera é o **componente MyAPP**, cuja função é inicializar as páginas da aplicação. Esse arquivo é encontrado em "pages/_app.js".

É possível sobrescrever esse arquivo para controlar a inicialização das páginas dependendo das suas necessidades. Isso permite:

- Que partes do layout persistam entre mudanças de páginas;

- Manter o estado da aplicação quando ocorrer a navegação entre páginas;
- Capturar e tratar de erros e exceções;
- Injetar dados adicionais nas páginas;
- Declarar estilos (CSS) de maneira global.

No pedaço de código adiante, podemos observar o conteúdo padrão do arquivo `"/pages/app.js"` gerado pelo Next.js durante a criação do projeto:

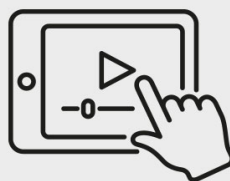
```
JavaScript   
  
1 import '../styles/globals.css'  
2 function MyApp({ Component, pageProps }) {  
3   return <Component {...pageProps} />  
4 }  
5 export default MyApp
```

pages/app.js



Instalando o Next.js

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Páginas

Conceitos básicos

O **Next.js** utiliza o **conceito de páginas**. Cada página é um componente ReactJs que fica armazenado na pasta "pages", onde se pode até mesmo adicionar parâmetros para criar rotas dinâmicas. Para a criação de uma página estática, basta, dentro dessa pasta, criar um arquivo com o nome que se deseja mapear no link.

Seguindo tal lógica, vamos supor que queiramos criar um link "/cadastro" para acessar um formulário. A lógica desse formulário deve estar na pasta "/pages/cadastro.js".

Podemos ainda criar uma rota dinâmica. Para isso, devemos, na pasta "pages", criar outro diretório. Aí sim, dentro dele, podemos criar um arquivo, o qual, por sua vez, precisa seguir o seguinte padrão: [NOME DO PARÂMETRO].js.

Comentário

Uma rota dinâmica também é chamada de URL slugs.

Vamos considerar a seguinte página: "pages/noticias/[id].js". Qualquer link, como "/notícias/1" ou "/notícias/xpto", por exemplo, está sendo mapeado para o arquivo "pages/noticias/[id].js".

Para acessarmos o valor do parâmetro passado, precisamos usar uma função fornecida pelo próprio Next.js chamada de router, como podemos ver no pedaço de código a seguir.

Documentação oficial do router: <https://nextjs.org/docs/api-reference/next/router>

JavaScript

```
1 import { useRouter } from 'next/router'
2 const Noticias = () => {
3   const router = useRouter()
4   const { id } = router.query
5   return <p>ID da Notícia: {id}</p>
6 }
```

```
7 export default Noticias
```

Atenção:

Algumas vezes, será necessário enviar mais de um parâmetro para uma página.

Vamos pensar agora no exemplo de um site de notícias. Suponha que criemos uma página para acessar todas as notícias de determinado ano por categoria.

Para resolver tais situações, é possível criar vários segmentos de rotas dinâmicas. Com base no exemplo anterior, pode-se acessar o link `"/noticias/2022/esportes"` e `"/noticias?ano=2022&categoria=esportes"`, que está armazenado na estrutura da seguinte forma: `"/pages/noticias/[ano]/[categoria].js"`.

Links

Do lado do cliente (navegador), utiliza-se o componente `"link"` fornecido pelo próprio Next.js para acessar as rotas dinâmicas. É possível criar links usando os exemplos anteriores no pedaço de código adiante.

```
JavaScript
```

```
1 import Link from 'next/link'
2 function Home() {
3   return (
4     <ul>
5       <li>
6         <Link href="/noticias/xpto">
7           <a>vai para pages/noticias/[id]
8         </Link>
9       </li>
10      <li>
11        <Link href="/noticias?ano=2022&cat
12          <a>Vai para pages/noticias/[ar
13        </Link>
14      </li>
```

Documentação oficial do link: <https://nextjs.org/docs/api-reference/next/link>

Suporte integrado de folhas de estilo (CSS)

O **Next.js** permite importar folhas de estilo (CSS) dentro de um arquivo JavaScript de maneira global ou por meio de módulos, evitando, assim, colisões de classes e estilos dentro dos componentes de interface. Porém, quando a aplicação for preparada para produção, todos os arquivos CSS serão concatenados em um único arquivo CSS minificado.

Adicionando uma folha de estilo global

Normalmente, os estilos aplicados de maneira global são declarados no arquivo “styles.css” e importados dentro do arquivo “pages/_app.js”. Os estilos declarados em “_app.js” estarão disponíveis em toda aplicação por sua natureza global.

Considere, por exemplo, a folha de estilos com a declaração da tag body no arquivo denominado “globals.css”:

```
CSS
1 body {
2   background: #333333;
3   padding: 10px;
4   max-width: 880px;
5   margin: 0 auto;
6 }
```

globals.css

Dentro do arquivo “pages/_app.js”, faremos a importação do arquivo de estilos mostrado anteriormente. Dessa forma, toda declaração de estilos feita no arquivo “globals.css” estará disponível para todas as páginas do projeto.

JavaScript



```
1 import '../globals.css'
2 export default function MyApp({ Component, pageProps
3   return <Component {...pageProps} />
4 }
```

pages/_app.js

Importando uma folha de estilo de "node module"

A pasta `node_modules` armazena todas as dependências instaladas para o seu projeto funcionar. Dessa maneira, muitas vezes durante o desenvolvimento, será preciso utilizar componentes desenvolvidos por terceiros.

Vamos imaginar um cenário no qual você queira utilizar um framework de CSS, como, por exemplo, o Bootstrap, para ajudar na estilização dos seus componentes de interface.

Ao ser adicionado na aplicação, o Bootstrap armazena seus arquivos de estilo dentro da pasta "node module". Cabe a você importar esses arquivos.

Saiba mais

Atendendo a esse cenário, o Next.js, desde a versão 9.5.4, passou a permitir essa importação de estilos em qualquer lugar da aplicação.

Aplicando estilos a componentes

O Next.js oferece suporte a **módulos de CSS** que criam um escopo local dos estilos declarados, permitindo que você use o mesmo nome de classe em arquivos diferentes sem correr o risco de haver colisões e, consequentemente, de uma sobrescrita das declarações. A

funcionalidade de criação de um escopo local torna a importação de módulos de CSS a melhor maneira de declarar estilos a um componente.

Imagine que você queira criar um componente de tela reutilizável para representar um botão. O arquivo desse componente será armazenado em uma pasta própria chamada de “/ componentes”.

Dica

O módulo de CSS deve sempre usar a mesma convenção de nomenclatura em seu nome: [NOME DO COMPONENTE].module.css.

Para criarmos o componente de botão ilustrado no parágrafo anterior, geraremos um arquivo com o seguinte nome: /components/Button.module.css. Nesse arquivo de CSS, vamos criar dois estilos para o botão, um deles representando o status de sucesso e erro, como mostra o pedaço de código adiante.

Uma das vantagens dessa abordagem é que as classes declaradas “.erro” e “.sucesso” podem ser repetidas em outros componentes sem que ocorram colisões de declaração, pois elas só fazem efeito no escopo local do componente. O Next.js se encarregará de criar identificadores únicos quando o projeto for preparado para produção.


```
CSS
```

```
1 .erro {  
2   color: white;  
3   background-color: red;  
4 }  
5 .sucesso {  
6   color: white;  
7   background-color: green;  
8 }
```

/components/Button.module.css

Seguindo as convenções proposta pelo Next.js, como o arquivo do módulo de CSS se chama “/components/Button.module.css”, temos de criar o arquivo de componente desta forma: “/components/Button.js”.

Em seguida, vamos importar o arquivo CSS, como demonstra este pedaço de código:

```
JavaScript   
  
1 import styles from './Button.module.css'  
2 export function Button() {  
3   return (  
4     <button  
5       type="button"  
6       className={styles.sucesso}  
7     >  
8       Salvar  
9     </button>  
10  )  
11 }
```

/components/Button.module.css

No código da imagem, podemos ver a importação do módulo de CSS e como é feita a referência dele no HTML do botão. Nesse caso, o botão que será mostrado na tela tem a cor verde, pois a classe referenciada é a “sucesso”, que está descrita no respectivo arquivo de módulo de CSS.

Atenção!

Os módulos de CSS são recursos opcionais e só funcionam para os arquivos com a nomenclatura “.module.css”.

Otimização e instalação de fontes


Como acabamos de ver, é possível criar módulos de estilo (CSS) para cada página. Um estilo comum de ser customizado em sites e aplicações é a fonte.

O Next.js permite que haja uma otimização de fontes integradas. Desse modo, no momento da construção da aplicação, é inserido o CSS referente à fonte escolhida.

Comentário

Atualmente, o Next.js trabalha otimizando fontes da Web do Google Fonts e do Typekit, embora seus desenvolvedores estejam trabalhando para suportar outros provedores de fontes.

Observe no código a seguir o exemplo de como se deve importar sua fonte de maneira adequada: primeiramente, é necessário criar um arquivo dentro do diretório “pages” chamado “_document.js” com toda a estrutura necessária para montar a página.

```
JavaScript 
```

```
1 // pages/_document.js
2 import Document, { html, Head, Main, NextScript }
3 class MyDocument extends Document {
4   render() {
5     return (
6       <Html>
7       <Head>
8         <link href="https://fonts.googleapis.com/c
9       </Head>
10      <body>
11        <Main />
12        <NextScript />
13      </body>
14    </Html>
```


pages/_document.js

Personalizando a estrutura de uma página

Como pudemos observar no exemplo anterior, nós inserimos a tag de HTML “link” para realizar a importação de um servidor Web fonte. Essa é uma situação na qual precisamos customizar uma estrutura de página.

O Next.js permite que você crie uma estrutura de página personalizada.

Para isso, é preciso apenas criar um arquivo denominado “_document.js” dentro do diretório “pages”, como fizemos anteriormente, e utilizar componentes específicos do Next.js para arquitetar a estrutura.

```
JavaScript 
```

```
1 // pages/_document.js
2 import { Html, Head, Main, NextScript } from 'next/
3 export default function Document() {
4   return (
```



```
5      <Html>
6        <Head />
7        <body>
8          <Main />
9          <NextScript />
10       </body>
11     </Html>
12   )
13 }
```

pages/_document.js

O pedaço de código dessa imagem é composto pela marcação mínima de HTML e componentes do Next.js para que uma página seja renderizada corretamente. Os componentes fora do componente “<Main />” serão inicializados pelo navegador; portanto, não é preciso haver lógicas de programação nem propriedades de estilo (CSS).

Aplicação passo a passo

Criando o projeto

Agora você vai criar um projeto para consolidar em uma pequena aplicação todos os conceitos do Next.js apresentados anteriormente. A aplicação-conceito em questão é composta de duas páginas: uma mostra uma **lista de usuários do nosso sistema hipotético**; a outra, mais detalhes desse usuário clicado.

Usando o mínimo de funcionalidades, começaremos com a criação de um projeto Next.js, passando pelo entendimento da estrutura de diretórios e, por fim, integrando componentes de páginas às rotas dinâmicas e às estáticas de navegação. No terminal, digite “npx create-next-app” e aperte “enter” para criar o projeto.

A primeira pergunta que o Next.js fará é sobre o nome do projeto. Para fins didáticos, usaremos o nome “lista-usuarios-app”:

Terminal

```
1 nome@maquina:~/Área de Trabalho$ npx create-next-app
2 ? What is your project named? > lista-usuarios-app
```



Assim que você digitar esse nome, o Next.js começará a instalar todas as dependências para a criação do projeto. Tal procedimento pode demorar mais ou menos dependendo da velocidade da sua conexão.

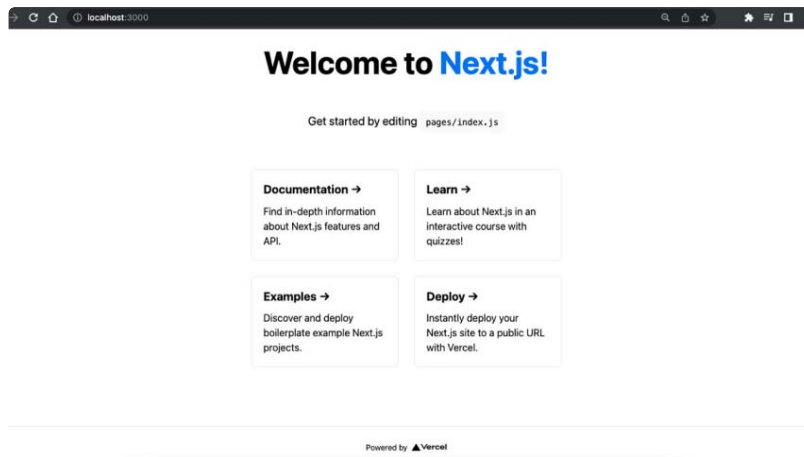
Terminada a instalação, o Next.js mostrará no terminal o procedimento para iniciar a aplicação:

```
Terminal
```

```
1 Success! Created lista-usuarios-app at /home/Area d
2 Inside that directory, you can run several commands
3   yarn dev
4     Starts the development server
5   yarn build
6     Builds the app for production.
7   yarn start
8     Runs the built app in production mode.
9 We suggest that you begin by typing:
10   cd lista-usuarios-app
11   yarn dev
```

O Next.js criará um diretório com o nome do projeto. Você deve entrar nesse diretório e inicializar a aplicação digitando “yarn dev”, como está instruído nas mensagens finais de criação do projeto.

A aplicação vai ser inicializada em <http://localhost:3000> . Acessando esse endereço no navegador, você terá acesso à página inicial padrão de um projeto Next.js como o mostrado a seguir:



Preparando a aplicação para o desenvolvimento

Abra o seu editor de código preferido (preferencialmente, o VSCode) e verifique a estrutura de diretórios criada pelo Next.js. Nós agora vamos limpar os códigos-padrão que montam a página inicial do projeto como a da imagem anterior.

O Next.js trabalha com o conceito de páginas, segundo o qual cada página é um componente ReactJs. Abra a pasta “pages” e acesse o arquivo “index.js”.


O conteúdo desse arquivo constitui exatamente o que precisamos retirar para criar nossa primeira página do projeto. Sua página deve estar semelhante ao código a seguir:

```
JavaScript

1 import Head from 'next/head'
2 import Image from 'next/image'
3 import styles from '../styles/Home.module.css'
4
5 export default function Home() {
6   return (
7     <div className={styles.container}>
8       <Head>
9         <title>Create Next App</title>
10        <meta name='description' content='
11        <link rel='icon' href='/favicon.ic
12      </Head>
13      <main className={styles.main}>
14        <h1 className={styles.title}>
```

index.js


Agora vamos retirar o conteúdo e deixar apenas um link para a página na qual listaremos os usuários do nosso sistema hipotético. Para criarmos esse link, usaremos um componente próprio do Next.js a exemplo do que vimos ao longo deste módulo. Seu código deve estar assim:

```
JavaScript 
```

```
1 import Link from 'next/link'
2 import styles from '../styles/Home.module.css'
3 export default function Home() {
4   return (
5     <>
6       <Link href="/usuarios">
7         <a className={styles.button}>Lista de l
8       </Link>
9     </>
10  )
11 }
```

index.js

É preciso prestar atenção no módulo de folha de estilos (CSS) para se poder estilizar o botão. Abra o arquivo “styles/Home.module.css”, limpe todo o código e acrescente o estilo mínimo do novo botão, como aponta o código adiante:

```
CSS 
```

```
1 .button{
2   padding: 20px;
3   background-color: cadetblue;
4 }
```


styles/Home.module.css

Criando a página com a lista de usuários

Vamos criar uma página que listará todos os usuários do nosso sistema. Como o Next.js determina que as páginas fiquem em um diretório denominado “pages”, devemos criar um arquivo chamado “usuarios.js” dentro dele.

Sabendo que o framework já cria automaticamente as rotas com base nos nomes das páginas, o link de acesso, ao criar esse arquivo, será “/usuarios”, tal como informamos anteriormente no caso do link do arquivo “index.js”. No arquivo “pages/usuarios.js”, vamos usar (apenas para simplificar) o projeto um array com os usuários da lista, mas esses dados poderiam vir de diferentes lugares, tal como uma API de terceiros.

Seu código deve estar parecido com este:

```
JavaScript   
  
1 import Link from 'next/link'  
2 import styles from '../styles/Usuarios.module.css'  
3 const usuarios = [  
4   {  
5     nome: "André Eppinghaus",  
6     id: 1  
7   },  
8   {  
9     nome: "Luiz Estevão",  
10    id: 2  
11  },  
12  {  
13    nome: "Vicente Calfo",  
14    id: 3
```

pages/usuarios.js

Nessa página, podemos observar que listamos apenas os usuários e direcionamos cada botão para outra página dinâmica que receberá o id do usuário. Além disso, criamos um módulo de estilos CSS mínimo apenas para estilizar a lista. Esse módulo está armazenado em “styles/Usuarios.module.css”. Observe este código:

CSS



```
1 .card{
2   display: inline-flex;
3   padding: 40px;
4   border: 1px solid gray;
5 }
```

styles/Usuarios.module.css

Atenção:

Lembre-se sempre da nomenclatura para a criação de módulos de CSS, em que o arquivo deve ser nomeado seguindo o padrão: NOME DA PAGINA.module.css.

No exemplo do nosso projeto, acabamos de criar uma página chamada Usuarios. Sendo assim, seu arquivo de módulo CSS deve se chamar: Usuarios.module.css.

Criando a página de usuário

Na etapa anterior, criamos a lista de usuários com links de rotas dinâmicas para acessar uma página com mais detalhes do usuário escolhido. Agora criaremos essa página, a qual, por sua vez, mostrará mais detalhes do usuário. O link criado na etapa anterior segue o seguinte padrão: "/usuario/ID DO USUARIO".

Para que esse link seja mapeado corretamente pelo Next.js, temos de seguir uma estrutura preestabelecida pelo framework: criar um diretório chamado "usuario" e, dentro dele, um arquivo com a variável desejada entre colchetes. Dessa maneira, o arquivo para a página de detalhes do usuário precisa estar armazenada em "/pages/usuario/[id].js".

Para fins de exemplo, criaremos um objeto com o nome do usuário apenas para validar a mudança dos dados, o que depende do link

selecionado na página `/usuarios`. Para receber o parâmetro passado no link, o Next.js fornece uma função denominada `router`.

Vamos observar sua implementação no código a seguir:

```
JavaScript
1 import { useRouter } from "next/router"
2 const usuarios = [
3   {
4     nome: "André Eppinghaus",
5     id: 1
6   },
7   {
8     nome: "Luiz Estevão",
9     id: 2
10  },
11  {
12    nome: "Vicente Calfo",
13    id: 3
14  }
```

`pages/usuario/[id].js`

Finalizando esse projeto, a despeito de ele ser simples, você foi capaz de implementar algumas funcionalidades do Next.js, verificando principalmente como funciona a estrutura de páginas e mapeamento para rotas estáticas e dinâmicas, além do funcionamento do módulo de CSS.

O projeto completo está disponível neste [repositório](#).

repositório

<https://github.com/vicentecalfo/lista-usuarios-app>

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Um projeto criado com o framework Next.js possui obrigatoriamente uma dependência da biblioteca ReactJs?

A

Não. Um projeto criado com o framework Next.js foi pensado para deixar o desenvolvedor decidir qual biblioteca de base deseja utilizar.

B

Não. Um projeto criado com o framework Next.js foi pensado para deixar o desenvolvedor decidir qual biblioteca de base deseja usar, embora ele indique a utilização do ReactJs.

C

Não. Um projeto criado com o framework Next.js não tem nenhuma dependência de biblioteca de terceiros.

D

Sim. Um projeto criado com o framework Next.js precisa usar obrigatoriamente o ReactJs como biblioteca de base.

E

Não. Um projeto criado com o framework Next.js não tem nenhuma dependência de biblioteca de terceiros, embora o ReactJs possa ser usado para fornecer mais recursos à aplicação.

Parabéns! A alternativa D está correta.

O Next.js é um framework que disponibiliza um conjunto de funcionalidades desenvolvido exclusivamente para ser adicionado a uma aplicação que utiliza a biblioteca ReactJS.

Questão 2

Considere que, em um projeto Next.js, uma página também seja um componente ReactJs e que eu esteja trabalhando em um projeto no qual possa acessar uma página de notícias, filtrada, por meio de parâmetros, por ano e categoria. Onde ficam os arquivos dessa página? Qual link para listar notícias de 2019 da categoria "entretenimento" eu devo acessar?

A

O arquivo do componente da página ficará armazenado em "pages/noticias.js". O link de acesso é: "/noticias?id=2019&categoria=entretenimento".

B

O arquivo do componente da página ficará armazenado em "pages/noticias/index.js". O link de acesso é: "/noticias/2019/entretenimento".

C

O arquivo do componente da página ficará armazenado em "pages/noticias/[ano]/[categoria].js". O link de acesso é: "/noticias?id=2019&categoria=entretenimento" ou "/noticias/2019/entretenimento".

D

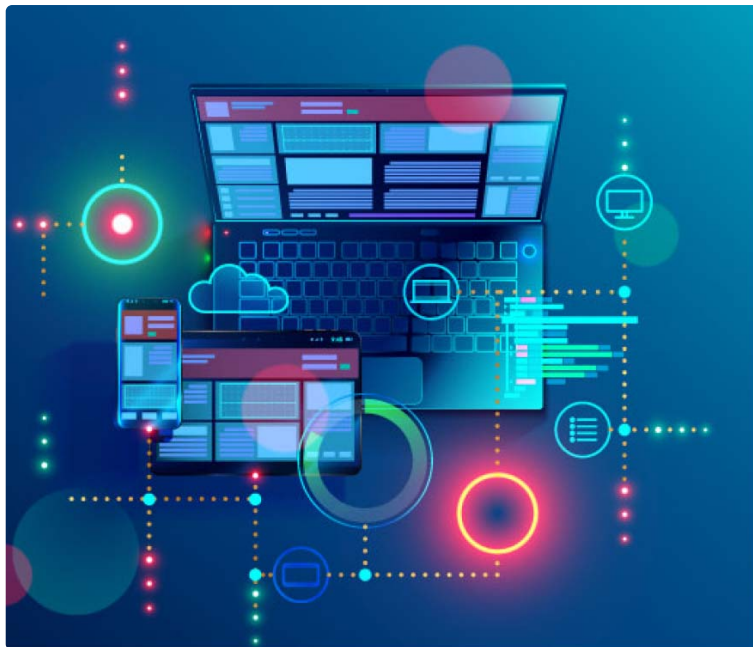
O arquivo do componente da página ficará armazenado em "pages/noticias/ano/[categoria].js". O link de acesso é: "/noticias?id=2019&categoria=entretenimento" ou "/noticias/2019/entretenimento".

E

O arquivo do componente da página ficará armazenado em "pages/noticias/index.js". O link de acesso é: "/noticias?id=2019&categoria=entretenimento".

Parabéns! A alternativa C está correta.

O Next.js é compatível com rotas dinâmicas que recebem parâmetros, mas, para isso, é necessário seguir um padrão. Por exemplo, se você criar um arquivo chamado "pages/noticias/[id].js", ele estará automaticamente acessível em "noticias/1", "noticias/2" e assim por diante. No caso das rotas dinâmicas com mais de um parâmetro, basta criar subdiretórios seguindo o mesmo padrão de colchetes. Exemplo: o arquivo "pages/noticias/[ano]/[categoria].js" estará acessível em "noticias/2019/entretenimento", "noticias/2022/política" e assim por diante.



2 - Módulos de pré-renderização do Next.js

Ao final deste módulo, você será capaz de aplicar a funcionalidade do Next.js de renderização estática.

Introdução à pré-renderização

Conceitos básicos

Você já sabe que o **Next.js oferece uma série de facilidades pré-configuradas** que permite a criação de uma aplicação ReactJs de maneira rápida e sem preocupação, por exemplo, com configurações extras, estruturas de pastas e configuração de rotas. Contudo, um dos grandes diferenciais dele é utilizar a renderização na parte do servidor para resolver um problema recorrente em aplicações e sites **SPA**, os quais, por sua vez, necessitam de **SEO** para obter uma boa indexação pelos motores de busca.

Como é comum em aplicações SPA, você perceberá que carregar todo o conteúdo no lado do cliente (navegador) pode não ser algo muito performático. Além disso, o Next.js também oferece soluções para se trabalhar a renderização de forma híbrida com SSG e SSR.

SPA

Single page application.

SEO

Search engine optimization.

Entender a importância da pré-renderização no Next.js é necessário para escolher a melhor forma de trabalhar a aplicação a ser desenvolvida. Por padrão, o framework vai pré-renderizar cada página, gerando, assim, HTMLs com antecedência, em vez de fazer tudo por meio do JavaScript no lado do cliente.

Os HTMLs gerados estão sempre associados ao código JavaScript mínimo necessário para permitir que a página funcione corretamente. Como vimos anteriormente, o Next.js trabalha com duas formas de pré-renderização: a principal diferença é quando o HTML da página é gerado.

A SSG (geração estática) é a mais recomendada, pois o HTML da página é gerado no momento da compilação, permitindo sua reutilização em cada solicitação, enquanto nos SSR (renderização do lado do servidor) o HTML da página é gerado na hora da solicitação. Embora uma geração estática seja recomendada por motivos de performance, nem sempre é possível aplicá-la dependendo do projeto.

Por isso, o Next.js possibilita a escolha do tipo de pré-renderização a ser usado em cada página, permitindo o trabalho híbrido de geração dos HTMLs.

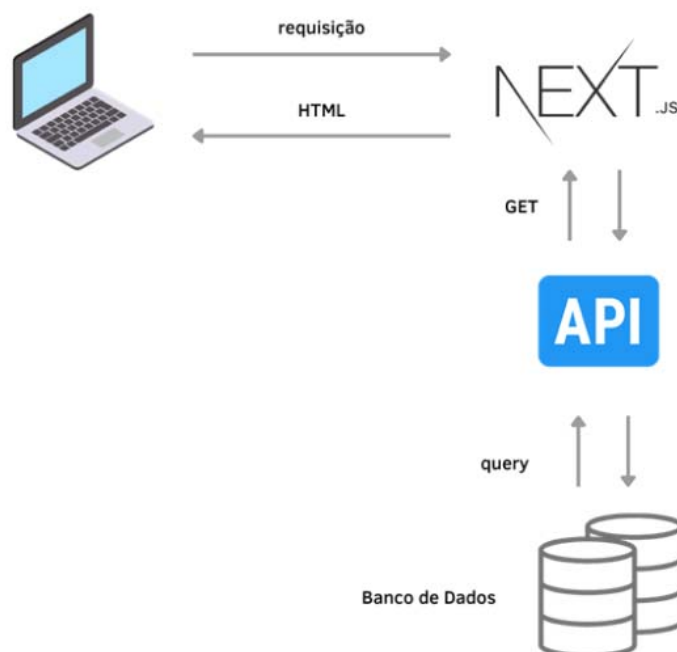
SSR (server side rendering)

A renderização no lado do servidor pode acontecer de duas maneiras. O primeiro cenário se dá ao **acessar um endereço HTTP**; o segundo, no **acesso a um link no próprio cliente** (navegador), sendo direcionado para outra página.

Conforme descrito no primeiro cenário, envia-se uma requisição **HTTP**; em seguida, tal solicitação migra para um servidor Next.js, que faz a comunicação com uma API, retornando a solicitação novamente para esse servidor. Aí sim o Next.js executa a renderização, devolvendo ao cliente os arquivos necessários para o funcionamento da tela acessada.

HTTP

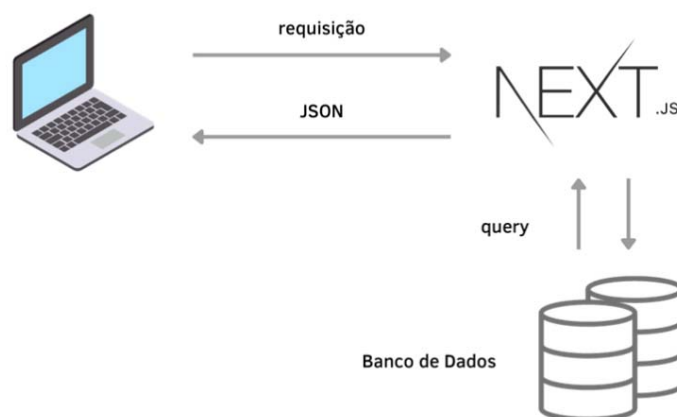
Hypertext transfer protocol.



Já na segunda situação, ocorre um redirecionamento interno para outra página dentro da aplicação. Quando essa nova página for gerada, uma requisição semelhante ao exemplo anterior ocorrerá – só que, dessa vez, o servidor Next.js poderá interagir diretamente com o banco de dados.

Exemplo

Devolver um JSON em vez de um HTML e fazer com que as estruturas internas da página sejam alteradas sem que ocorra um recarregamento.

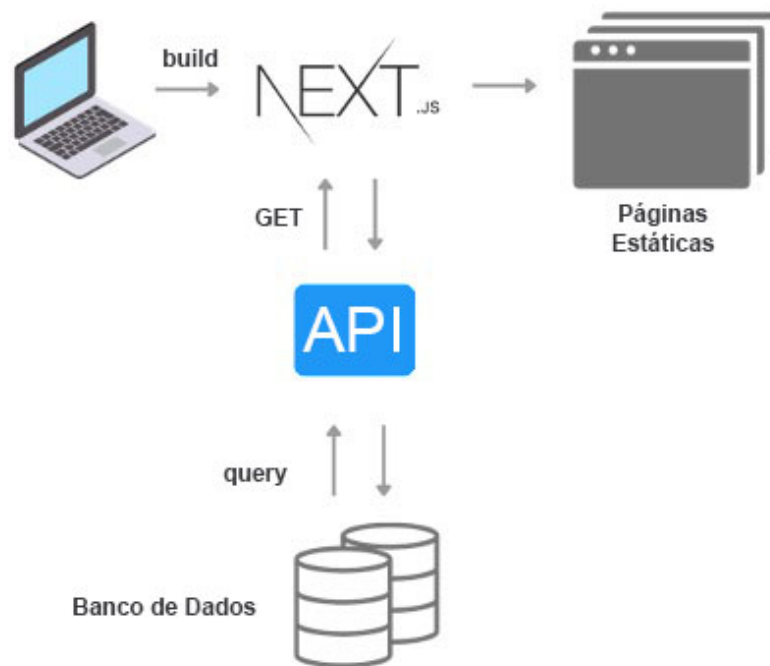


A grande vantagem das duas situações descritas anteriormente se dá pela indexação de conteúdo, pois o servidor já entrega os arquivos prontos para uso, tornando a aplicação muito mais amigável para as práticas de SEO (otimização de mecanismos de busca), diferentemente das tradicionais SPAs (aplicações de página única), nas quais o conteúdo só será montado por meio de uma chamada externa.

SSG (static site generation)

A geração de site estático acontece de modo diferente dos outros tipos de renderização, pois, em vez de acontecer em tempo de execução, ela **ocorre antes de a aplicação entrar em produção**. Tal processo é chamado de **tempo de build**.

As páginas da aplicação, portanto, serão todas geradas estaticamente para serem usadas enquanto se estiver preparando a aplicação para a produção.



Após o término do build, com as páginas estáticas já geradas, você pode armazená-las, a título de exemplo, em uma **CDN** para aumentar significativamente a performance. Um problema comum que ocorre em sites que trabalham com páginas do tipo é a necessidade de atualização com o passar do tempo.

O Next.js criou um conceito chamado de **ISR**, que consiste em gerar páginas estáticas em tempo de execução. Por meio de uma configuração mínima, o Next.js é capaz de revalidar quando uma página estática deve ser recriada, mantendo o conteúdo atualizado, embora preserve as vantagens de uma página estática.

Quando o primeiro usuário entrar na página após o período de revalidação expirado, isso fará com que o servidor Next.js recrie a página estática. A partir dali, todos os demais usuários acessarão a versão já atualizada da página.

CDN

Content delivery network.

ISR

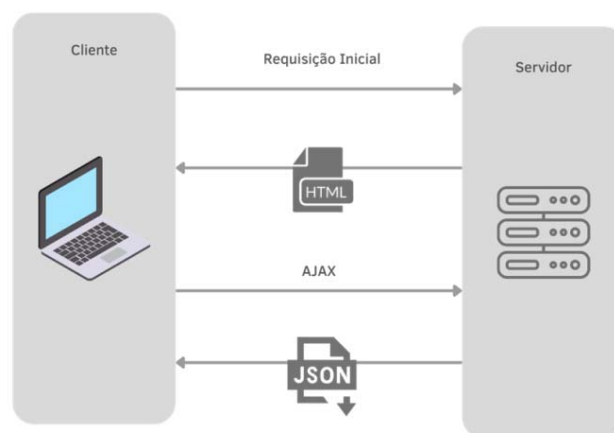
Incremental static regeneration

Uma das vantagens recorrentemente descritas do Next.js é a sua capacidade de ser amigável às práticas do SEO (otimização de mecanismos de busca). Por isso, vamos entender exatamente como ele atua e de que forma resolve o problema que as SPAs têm para obter uma indexação satisfatória nos mecanismos de busca.

Como vimos, SPA é uma sigla que vem do inglês single page application, isto é, uma aplicação de página única. Essa denominação é bem precisa, pois ela funciona exatamente dessa forma.

Em vez de devolver uma página HTML a cada requisição a exemplo das aplicações tradicionais, a SPA carrega uma única vez o HTML: cada vez que o usuário acessa um novo recurso, o JavaScript é acionado e dinamicamente carrega e manipula o HTML para gerar essa nova página.

Bibliotecas como ReactJs, Angular e VueJs utilizam exatamente esse tipo de abordagem, como expõe o ciclo de vida de uma SPA da figura a seguir:



A abordagem da SPA trouxe todo um novo conceito sobre o tempo de carregamento de uma aplicação, um melhor desempenho e o emprego de alta performance de JavaScript. Entretanto, tal estrutura compromete demais o SEO das aplicações SPA, uma vez que esses conteúdos são

carregados dinamicamente somente com a ação externa do usuário, não estando disponíveis para a indexação dos mecanismos de busca.

Quando acessam o HTML de uma página SPA, os robôs de indexação encontram uma página com o conteúdo vazio, pois a tela somente será montada à medida que o usuário navegar pelos recursos disponíveis.

Atenção!

Outra preocupação que o desenvolvedor deve ter em relação às SPAs é a questão de disponibilidade de recursos do lado do cliente, já que as páginas serão renderizadas no navegador. Alguns fatores, como hardware e velocidade de conexão, por exemplo, precisam fazer parte do levantamento de requisitos não funcionais na hora de decidir pela adoção dessa tecnologia.

A renderização do lado do cliente é chamada de **CSR**, sendo a mais comum em aplicações SPA. Observe o fluxo na figura adiante, que usa o ReactJs como exemplo:

CSR

Client side render.



Como vemos no fluxo apresentado, o conteúdo só é montado à medida que o usuário acessa os recursos da aplicação. Isso prejudica a indexação dos mecanismos de busca, pois os conteúdos não são renderizados antes da solicitação do usuário.

Busca de dados (data fetching)


A primeira coisa que se precisa aprender no Next.js para trabalhar com os tipos de renderização é **como buscar dados**, pois é por meio dessa busca que se consegue renderizar os conteúdos de maneiras diferentes de acordo com a necessidade de uso da sua aplicação.

Descreveremos agora **duas funções**:

a) getServerSideProps (SSR)

Quando é exportada de uma página, a função `getServerSideProps` faz com que o Next.js pré-renderize a página em cada solicitação, só sendo executada no lado do servidor. Ao solicitar uma página, o `getServerSideProps` é executado no exato momento da solicitação para realizar a pré-renderização e retornar os dados.

Caso você esteja transacionando entre as páginas através de rotas e links da aplicação, o Next.js enviará uma solicitação de API ao servidor que vai executar o `getServerSideProps`.

```
JavaScript 
```

```
1 export async function getServerSideProps(context){  
2     return {  
3         props: {}, // Esse objeto será enviado ao cc  
4     }  
5 }
```

O `getServerSideProps` retorna um JSON, que é usado para renderizar a página. Todo esse fluxo é orquestrado automaticamente pelo Next.js sem nenhuma configuração extra, desde que você tenha declarado corretamente a função.

A função `getServerSideProps` só funcionará se for exportada de um componente que seja uma página, como evidencia o exemplo a seguir:

```
JavaScript 
```

```
1 function Pagina({ data }) {  
2     // Renderiza os dados da página...  
3 }  
4  
5 // A função getServerSideProps será chamada a cada  
6 export async function getServerSideProps() {  
7     // Buscando dados externos em uma API  
8     const res = await fetch(`https://api.exemplo.t  
9     const data = await res.json()  
10    // Passando os dados recuperados para a página  
11    return { props: { data } }  
12 }  
13  
14 export default Pagina
```

b) getStaticProps (SSG)

Para realizar uma renderização de sites estáticos, o Next.js permite a exportação de uma função chamada `getStaticProps` de um componente que seja uma página. Dessa maneira, em tempo de compilação (build), ele usa o atributo `props` para enviar as informações para a geração das páginas estáticas, como podemos ver no pedaço de código a seguir:

```
JavaScript  
1 export async function getStaticProps(context) {  
2     return {  
3         props: {}, // Dados que serão passados para o co  
4     }  
5 }
```

Deve-se considerar a utilização de `getStaticProps` nos casos em que estão disponíveis os dados para gerar a página antes da solicitação do usuário ou quando eles vêm de algum gerenciador de conteúdo (CMS),

por exemplo. Esses cenários descritos há pouco permitem que você pré-renderize as páginas, deixando-as amigáveis não só para práticas de SEO, mas também melhorando a performance de acesso da aplicação, uma vez que tais conteúdos gerados podem estar hospedados em uma CDN e prontos para atender às requisições dos usuários.

A execução do `getStaticProps` sempre ocorre previamente no servidor durante o tempo de build da aplicação, porém, se você estiver usando a regeneração estática incremental, essa função será executada em segundo plano para gerar a nova página estática.

Neste pedaço de código, podemos ver um exemplo mínimo de como funciona o `getStaticProps`:

```
JavaScript
1 // Os usuários serão gerados na lista em tempo de
2 function Usuarios({ usuarios }) {
3   return (
4     <ul>
5       {usuarios.map((usuario)=>(
6         <li>{usuario.nome}</li>
7       ))}
8     </ul>
9   )
10 }
11
12 // Esta função será chamada em tempo de construção
13 export async function getStaticProps() {
14   // Chamando API externa para trazer os usuários
```

Nesse exemplo do código, podemos ver que seu funcionamento é bem simples: como na maioria das funcionalidades do Next.js, ele não requer nenhuma configuração extra: apenas a declaração da função basta para que o servidor saiba o momento de ser executado.

Em tempo de build da aplicação, o Next.js, nesse exemplo, vai acessar a API para retornar todos os usuários e, com os dados recebidos, enviar via props para o componente da página `Usuários`. Isso permite a construção do HTML estático para a montagem da página com os nomes dos usuários.

Uma página gerada por meio da função `getStaticProps` produz dois arquivos: um HTML e um JSON, que contém o resultado da execução. A função do arquivo JSON é alimentar os componentes de página quando o usuário navega internamente por meio de link ou rotas do Next.js.

Regeneração estática incremental (ISR)

Conceito

Como vimos anteriormente, o Next.js tem a capacidade de gerar páginas estáticas que permitem à aplicação obter um ganho em performance de acesso, bem como ser amigável aos mecanismos de busca, para obter um melhor ranqueamento nos resultados de pesquisa.

O Next.js também oferece a oportunidade de **criar ou atualizar uma página estática após a criação da aplicação**. A regeneração estática incremental permite que, por meio da geração estática de páginas, seja possível atualizar páginas sem a necessidade de reconstruir toda a aplicação.

A abordagem incremental é bem parecida com a renderização SSG, tendo inclusive a mesma função. A `getStaticProps`, porém, tem um parâmetro a mais, como veremos no pedaço de código a seguir:



```
JavaScript

1 function Noticias({ noticias }) {
2   return (
3     <ul>
4       {noticias.map((noticia) => (
5         <li key={noticia.id}>{noticia.titulo}<
6       ))}
7     </ul>
8   )
9 }
10
11 // Esta função vai ser chamada em tempo de constru
12
13 export async function getStaticProps() {
14   const res = await fetch('https://exemplo.api.i
```

No código da imagem, podemos ver que a implementação é a mesma que usamos para gerar páginas estáticas: a única diferença é o parâmetro `revalidate`, usado pelo Next.js como gatilho para regenerar as páginas quando se faz necessário. No entanto, outra função apareceu no código (e até agora não falamos dela): a `getStaticPaths`.

Essa função é utilizada pelo Next.js para definir uma lista de caminhos dinâmicos (rotas) que precisam ser gerados estaticamente. Após você declarar e exportar essa função em uma página que usa rotas dinâmicas, o servidor Next.js vai pré-renderizar estaticamente todos os caminhos especificados em `getStaticPaths`.

Usamos a função `getStaticPaths` principalmente quando precisamos pré-renderizar estaticamente páginas que usam rotas dinâmicas. Muitas vezes, tais cenários se apresentam nas situações em que é necessário recuperar dados de algum gerenciador de conteúdo, banco de dados ou sistema de arquivos.

Essa função só pode ser exportada de uma página que utilize rota dinâmica e que faça uso da função `getStaticProps`, como pudemos verificar no pedaço de código mostrado anteriormente.

Otimização de imagens

Gerar páginas estáticas é uma abordagem fundamental para obter uma boa performance de carregamento do site, porém outros componentes de tela também impactam nesse resultado. Fazem parte do grupo dos mais comuns (e dos mais negligenciados em termos de otimização) as **imagens**.

O Next.js oferece um componente para facilitar a otimização delas, estendendo o elemento de HTML de imagem (``). Preocupar-se com a performance de carregamento de um site é necessário tanto para oferecer uma boa experiência ao usuário quanto para obter boas classificações de indexação nos resultados de busca, por exemplo.

As otimizações oferecidas pelo Next.js incluem não só a redução de peso dos arquivos, veiculando imagens do tamanho correto para cada tipo de dispositivo, mas também controlam o carregamento das imagens somente quando elas entram no espaço de visualização, aumentando significativamente o tempo de carregamento das páginas.

Para utilizar esse componente em suas páginas, basta realizar a importação `"next/image"` como a do pedaço de código a seguir:



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

Com o componente importado, você pode definir a fonte da imagem, determinando se ela será carregada local ou remotamente.

Carregando uma imagem local

Para usar uma imagem local, você precisa importar o seu arquivo; com isso, o Next.js, em tempo de construção, será capaz de analisar a imagem e realizar os ajustes necessários para a geração da página estática. A largura e altura (width e height, respectivamente) são determinados automaticamente caso não sejam informados.

Além disso, também é possível informar um arquivo de pré-carregamento (propriedade blurDataURL) para manter a imagem borrada, enquanto a imagem original tem seu carregamento finalizado, conforme mostra o pedaço de código adiante:

JavaScript

```
1 import Image from 'next/image'
2 import fotoAvatar from '../public/assets/avatar.pr
3
4 function Home() {
5   return (
6     <>
7       <h1>Perfil de Usuário</h1>
8       <Image
9         src={fotoAvatar}
10        alt="Foto do Usuário"
11        // width=(200)
```

```
12      // height={200}  
13      // blurDataURL="data:  
14      // ..."
```

Carregando uma imagem remota

O carregamento de uma imagem remota é bem semelhante ao de uma local, porém é importante prestar atenção em alguns detalhes. Na propriedade de fonte (src), é obrigatório informar sobre uma URL, seja ela absoluta ou relativa.

O Next.js não terá acesso a arquivos remotos em tempo de construção. Por isso, deve-se determinar a largura e a altura manualmente, o que é diferente do carregamento local, como se observa neste pedaço de código:

```
JavaScript  
  
1 import Image from 'next/image'  
2  
3 function Home() {  
4   return (  
5     <>  
6     <h1>Perfil de Usuário</h1>  
7     <Image  
8       src="/imagens/avatar.png"  
9       alt="Foto do usuário"  
10      width={200}  
11      height={200}  
12    </>  
13  )  
14 }
```

Talvez você precise acessar uma imagem remota como a do código mostrado anteriormente, embora, ainda assim, deseje usar a API do Next.js para a otimização de imagens. Para que isso aconteça de maneira segura, é necessário, no arquivo de configuração do Next.js, informar em quais domínios está autorizado o acesso, conforme aponta este arquivo `next.config.js`:

```
JavaScript  
  
1 module.exports = {  
2   images: {  
3     domains: [  
4       // ...  
5     ]  
6   }  
7 }
```

```
4      exemplo-1.com.br,  
5      exemplo-2.com.br,  
6      exemplo-3.com.br,  
7    ],  
8  },  
9 }
```

Dimensionamento da imagem

Como já falamos, **as imagens figuram entre os elementos que mais impactam na performance de carregamento de um site**. Não se preocupar com a otimização pode impactar drasticamente no desempenho do carregamento de uma página, além de gerar um efeito desagradável na experiência do usuário, o que ocorre quando os elementos das páginas, como textos, por exemplo, são empurrados à medida que o carregamento das imagens ocorre.

Para evitar essa experiência visual desagradável, é importante **dimensionar sua imagem**, fazendo com que o navegador já reserve o espaço antes mesmo de completar o carregamento. Componente do Next.js, o “next/image”, que auxilia nessa otimização de imagens, foi projetado para garantir um bom desempenho não só de carregamento, mas também de experiência do usuário.

Para determinar o dimensionamento correto das imagens, é preciso empregar uma destas três abordagens:

1. Usar a importação local de imagens, pois, tendo o acesso à imagem, o Next.js é capaz de determinar a largura e a altura;
2. Determinar explicitamente a largura e a altura da imagem;
3. Utilizar a propriedade “layout=fill”; dessa maneira, a imagem vai se expandir para preencher totalmente o espaço do elemento (HTML) pai.

Aplicação passo a passo

O que vamos fazer?

Agora você vai criar um pequeno projeto que consiste em gerar um blog utilizando o Next.js. O objetivo dessa atividade é entender **como funciona a função “getStaticProps” para criar as páginas estáticas de notícias do blog.**

Criaremos um blog com ReactJs e Next.JS, cujos posts serão escritos em Markdown e convertidos para HTML no final do processo de construção. Antes de começarmos a criação do projeto, porém, precisamos entender o que é um arquivo Markdown.

Entendendo um arquivo Markdown

Markdown é um sistema de formatação simples que visa, por meio de uma codificação mínima, a estilizar um conteúdo que depois será convertido para HTML. A vantagem da utilização dele é que o autor não precisa conhecer as tags de HTML para a formatação do conteúdo.

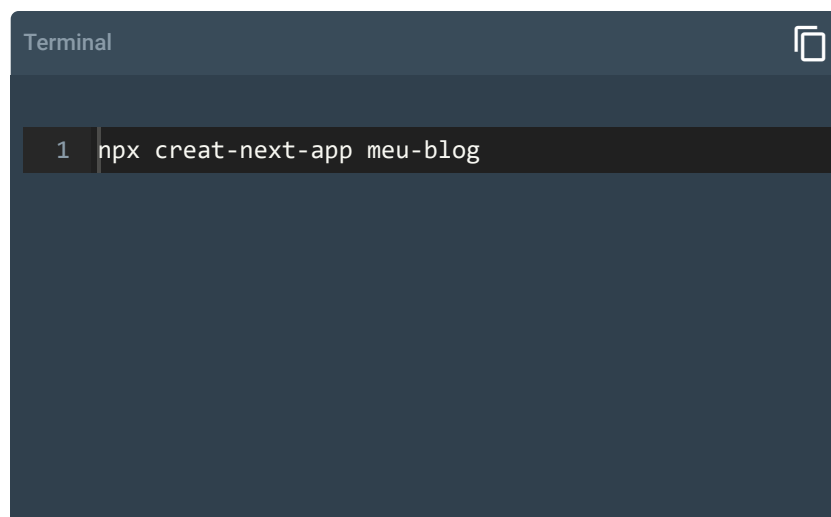
Por isso, tal linguagem é frequentemente usada para criar:

- Arquivos de documentação de código, como, por exemplo, os famosos arquivos README;
- Textos ricos em editores de textos simples.

Em nosso projeto, utilizaremos os arquivos Markdown para escrever o conteúdo das notícias a fim de que, em seguida, o Next.js leia um a um esse conteúdo e gere estaticamente nossas páginas HTML do blog.

Criando o projeto

Abra o terminal e execute o seguinte comando para criar o projeto com o nome “meu-blog”:

A screenshot of a terminal window with a dark blue background. The title bar at the top says "Terminal" and has a close button icon on the right. The terminal shows a single line of command: "1 npx creat-next-app meu-blog". The command is highlighted with a dark background and white text. The number "1" is in a light blue font, indicating it's the first line of input.

```
Terminal
```

```
1 npx creat-next-app meu-blog
```



Agora vamos acessar o diretório “meu-blog”, que foi criado para instalar a biblioteca que vai nos ajudar a converter o Markdown para HTML:

```
Terminal
```

```
1 npm install react-markdown
```

Em seguida, instalaremos uma biblioteca que vai permitir a criação de metadados para nossas notícias dentro do arquivo Markdown:

Criando o diretório de notícias

Criaremos um diretório chamado “noticias” na raiz do projeto. Esse diretório tem como objetivo guardar os arquivos Markdown de notícias.

Dentro desse diretório “noticias”, vamos criar um arquivo da primeira notícia chamado “ola-mundo.md”:

```
1 ---  
2 title: "Olá Mundo"  
3 date:: "05/05/2022"  
4 ---  
5 Olá mundo!
```

```
6 Podemos demonstrar um **texto e negrito** e um _tex
7 Além disso vamos ver como fica uma lista:
8 - Item 1
9 - Item 2
10 - Item 3
11 - Item 4
```

Como podemos observar no início do arquivo, ali foram declarados o título e a data da notícia. Dentro dos três traços, nós declararemos os metadados da notícia.

Preparação da página inicial

Agora vamos limpar o arquivo “/pages/index.js” a fim de que possamos começar a customizar nosso blog. Por enquanto, vamos deixar o arquivo o mais simples possível, como mostra a imagem adiante:

LINGUAGEM DE PROGRAMAÇÃO UTILIZADA

```
1 import styles from '../styles/Home.module.css'
2
3 const Home = () => {
4   return <h1>Notícias</h1>;
5 };
6
7 export default Home;
```

pages/index.js

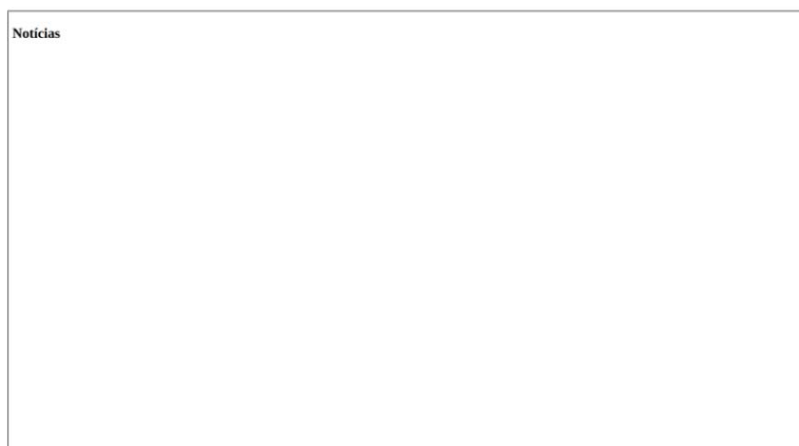
No arquivo “pages/_app.js”, vamos incluir tanto o componente Head do Next.js quanto o nome do blog na tag de HTML “<title>”:

JavaScript

```
1 import Head from 'next/head';
2
3 function MyApp({ Component, pageProps }) {
4   return (
5     <>
6       <Head>
7         <title>Meu Blog</title>
8         <link rel="icon" href="/favicon.ico" />
9       </Head>
10
11       <Component {...pageProps} />
12     </>
13   );
14 }
```

pages/_app.js

Em seguida, no terminal, execute o comando “yarn dev”. Se tudo correr bem, você, ao acessar o link “localhost:3000”, verá a seguinte tela em seu navegador:



Obtendo as notícias

Precisamos criar uma função para ler os arquivos Markdown e retornar uma lista das notícias. Para isso, criaremos um diretório chamado “lib” e, dentro dele, um arquivo JavaScript denominado “noticias.js”.

```
JavaScript
1 import { promises as fs } from "fs";
2 import path from "path";
3 import matter from "gray-matter";
4
5 const buscaNoticias = async () => {
6   const diretorioNoticias = path.join(process.cwd(),
```

```
7   const filenames = await fs.readdir (diretorioNoticias)
8
9   return await Promise.all(
10     filenames.map(async (filename) => {
11       const filePath = path.join(diretorioNoticias, filename)
12       const fileContents = await fs.readFile(filePath)
13       const document = matter(fileContents);
```

lib/noticias.js

O código da imagem é responsável por ler o diretório “noticias”: ele devolve uma lista com um objeto para cada notícia, contendo título, data, URL de acesso (slug) e o conteúdo em Markdown. Usamos a biblioteca “gray-matter” para obter os metadados entre os três traços nos arquivos Markdown.

Incluindo as notícias na página inicial

Para montar a listagem das notícias na página inicial, devemos importar a função “buscaNoticias” que acabamos de criar, como mostra o pedaço de código a seguir:

JavaScript

```
1 import Link from "next/link"
2 import buscaNoticias from '../lib/noticias'
3
4 const Home = () => ({ noticias H) => {
5   return (
6     <>
7       <h1>Notícias</h1>
8       <ul>
9         {noticias.map(({ slug, title }) => (
10           <li key={slug}>
11             <Link href={`/${slug}`}>
12               <a>{title}</a>
13             </Link>
14           </li>
```

pages/_app.js

Criando as páginas dinâmicas

Desejamos o mesmo template para as notícias. No entanto, usando rotas dinâmicas (baseadas no nome dos seus arquivos), vamos criar e nomear o arquivo dos posts com colchetes: `pages/[slug].js`.

```
JavaScript

1 import ReactMarkdown from "react-markdown";
2 import buscaNoticias from "../lib/noticias";
3
4 const Noticia = ({ title, date, markdown }) => (
5   <article>
6     <h1>{title}</h1>
7     <time>{date}</time>
8     <ReactMarkdown>{markdown}</ReactMarkdown>
9   </article>
10 );
11
12 export const getStaticPaths = async () => {
13   const noticias = await buscaNoticias();
14 }
```

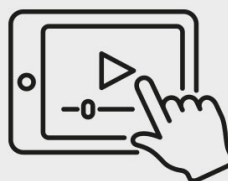
`pages/[slug].js`

Agora, toda vez que tivermos de gerar a página estática de notícias, o “`getStaticPaths`” buscará todos os arquivos “Markdown” dentro de notícias. A partir desse conjunto, criaremos a página no componente “Noticia” com o “slug” (link) baseado no nome do arquivo “Markdown”.



Criação de uma página passo a passo

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Uma das principais vantagens da geração de páginas estáticas é a performance de carregamento; no entanto, quando o conteúdo a ser mostrado nas páginas precisa ser atualizado, é necessário recriar todas as páginas. O que o Next.js oferece de solução para resolver esse problema?

A

A regeneração estática incremental (ISR).

B

Infelizmente, a cada atualização de conteúdo, é necessário realizar a construção do site novamente.

C

A geração de estática de páginas (SSG).

D

A geração do lado do servidor (SSR).

E

Infelizmente, o framework não oferece nenhum suporte à pré-renderização.

Parabéns! A alternativa A está correta.

O Next.js permite que você crie ou atualize páginas estáticas depois de criar o site. Essa abordagem é chamada de regeneração estática incremental (ISR), pois oferece a oportunidade de gerar a página estática de forma pontual, sem que seja necessário reconstruir todo o site a cada atualização. Na mesma função de criação de uma página do tipo (`getStaticProps()`), é possível adicionar um parâmetro denominado “revalidate” no qual se consegue acrescentar um valor numérico, que, por sua vez, representa os segundos em que a página deve ser recriada. O primeiro usuário que acessá-la após esse limite ativará a recriação dela; assim, todos os outros usuários subsequentes já terão acesso à página estática gerada, o que aumenta significativamente a performance e permite que o site escale para milhões de acessos.

Questão 2

Em que situação a geração estática de caminhos (`getStaticPaths()`) para a criação de páginas dinâmicas precisa ser usada?

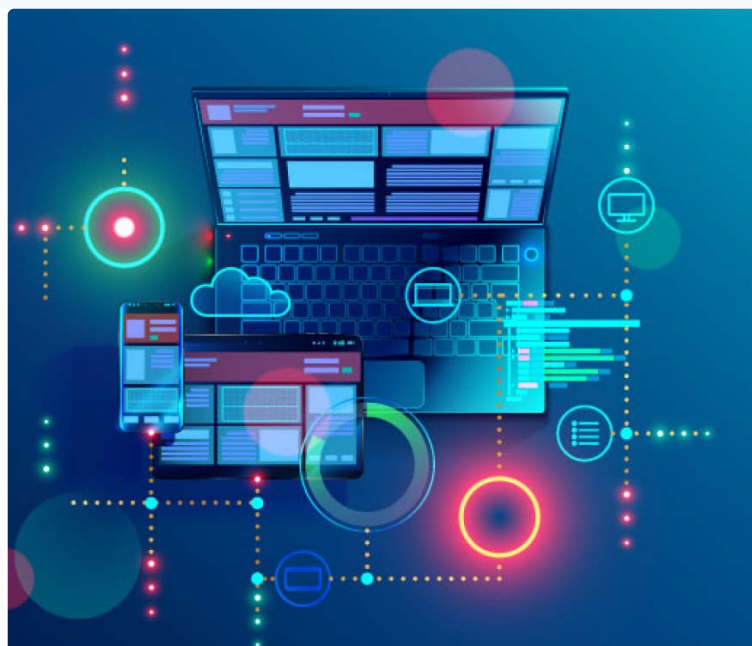
A

Quando o conteúdo vier especificamente de um gerenciador de conteúdos.

- B** Somente quando o conteúdo vier de um banco de dados externo.
- C** Somente quando o conteúdo vier de uma API.
- D** O `getStaticPaths()` deverá ser usado mesmo quando for exportado de uma página que não é acessada por meio de uma rota dinâmica.
- E** O `getStaticPaths()` terá de ser usado quando for necessário pré-renderizar estaticamente as páginas que usam rotas dinâmicas.

Parabéns! A alternativa E está correta.

A função `getStaticPaths()` deve ser usada em uma página acessada por meio de uma rota dinâmica. Ela é comumente usada para pré-renderizar páginas cujos dados sejam provenientes de um gerenciador de conteúdo (WordPress, Joomla, Contentfull e outros exemplos), banco de dados ou sistema de arquivos (Markdown, por exemplo). O Next.js vai acessar todos os caminhos gerados e fará a pré-renderização estática dessas páginas.



3 - Deploy na nuvem

Ao final deste módulo, você será capaz de desenvolver uma aplicação desenvolvida em Next.js em uma infraestrutura de nuvem.

Implantando uma aplicação Next.js

Procedimentos básicos

Agora que você é capaz de criar projetos no Next.js, gerar transições de páginas com rotas dinâmicas, otimizar imagens próprias para cada dispositivo de acesso e gerar páginas estáticas, aproveitando todas as vantagens de pré-renderização, chegou a hora de preparar a aplicação para a produção. A primeira coisa de que se precisa para implantar uma aplicação Next.js **é construir uma versão otimizada, o que vulgarmente chamamos de "build"**.

Por meio do comando "next build", o Next.js construirá uma aplicação que contém os seguintes arquivos de saída dentro de um diretório chamado ".next":

- Arquivos HTML das páginas geradas estaticamente;
- Arquivos de estilo (CSS), tanto os globais quanto os modulares de cada componente;
- JavaScript para as funcionalidades de pré-renderização do conteúdo dinâmico;
- JavaScript da aplicação gerada majoritariamente pelo ReactJs.

Dentro do diretório ".next", serão criados vários diretórios e arquivos, conforme podemos ver adiante:



.next/static/chunks/pages

Os arquivos gerados dentro desse diretório correspondem a uma rota de mesmo nome. Exemplo: ".next/static/chunks/pages/noticias.js" é o arquivo que será carregado quando o usuário acessar a URL "/noticias".



.next/static/media

As imagens importadas pelo “next/image” são criptografadas e copiadas para esse diretório.



.next/static/css

Os arquivos gerados dentro desse diretório são as folhas de estilo (CSS) usadas global ou modularmente para todas as páginas geradas para a aplicação.



.next/server/pages

Páginas HTML e JavaScript pré-renderizadas do servidor.



.next/server/chunks

Arquivos de JavaScript que se repetem em vários lugares da aplicação.



.next/cache

É usado para armazenar imagens e páginas no cache (em memória) do servidor Next.js a fim de ajudar a reduzir o tempo de compilação e melhorar o carregamento dos conteúdos.

Recomendações para construção da aplicação para produção

Antes de construir sua aplicação para produção, é importante se preocupar com alguns pontos a fim de melhorar a experiência final do usuário. Via de regra, uma das principais atitudes a serem tomadas para melhorar o tempo de carregamento da aplicação é empregar uma estratégia de cache sempre que possível.

Dica

Armazenar conteúdo em cache melhora o tempo de resposta. De forma automática, o Next.js adiciona cabeçalhos de cache a arquivos de JavaScript, CSS, imagens e HTML estáticos.

Além de empregar uma estratégia de cache, é importante que você possua um tratamento de erros robusto e tenha configurado páginas de erro, como, por exemplo, o clássico erro 404 de página não encontrada.

Customizando páginas de erro

HTTP é um protocolo Web para visualização de “hipertexto”, ou seja, de sites. Trocado entre o servidor e cliente (navegador), esse protocolo é dividido em códigos de três dígitos.

O primeiro código significa uma classificação dentro das **cinco categorias** existentes:

Faixa de código	Significado	Explicação
1xx	Informativo	A solicitação foi aceita, e o processo está em andamento.
2xx	Confirmação	A ação foi concluída com sucesso.
3xx	Redirecionamento	A solicitação precisa de alguma ação para ser concluída.
4xx	Erro no cliente	A solicitação não pode ser concluída ou contém erros.

Faixa de código	Significado	Explicação
5xx	Erro no servidor	O servidor não é capaz de concluir a solicitação.

Tabela: Faixa de código de erro.
Vicente Calfo.

Nós não vamos nos aprofundar em todos os códigos de erro HTTP, que, como podemos ver nessa tabela, estão na faixa do 4xx e 5xx. Em vez disso, vamos aprender a customizar os dois principais erros que mais ocorrem em uma aplicação Web:

404

Quando uma URL acessada recebe um erro 404, isso quer dizer que tal endereço não levou a nenhuma resposta de sucesso, isto é, a página acessada não existe mais, a URL foi alterada ou o endereço digitado está errado. Por isso, esse erro é conhecido como “página não encontrada”.

Independentemente do uso do Next.js para criar um website ou uma aplicação, o erro 404 é uma das primeiras exceções com a qual devemos nos preocupar, já que ele, como mencionamos anteriormente, se faz presente em todo projeto: ainda que a aplicação dele esteja totalmente atualizada, basta o usuário digitar uma URL errada para que tal erro seja disparado.

Por causa da alta frequência com que o erro 404 é disparado, o Next.js oferece um suporte para a criação de página estática a fim de tratá-lo. A página de erro pré-renderizada evita a carga no servidor Next.js, restando custos computacionais e experiências lentas para o usuário.

Para criar e customizar a página de erro 404, basta criar um arquivo em “pages/404.js”. Assim como os outros arquivos do diretório “pages”, O Next.js vai gerar a página estática do erro 404 em tempo de construção.

Javascript



500

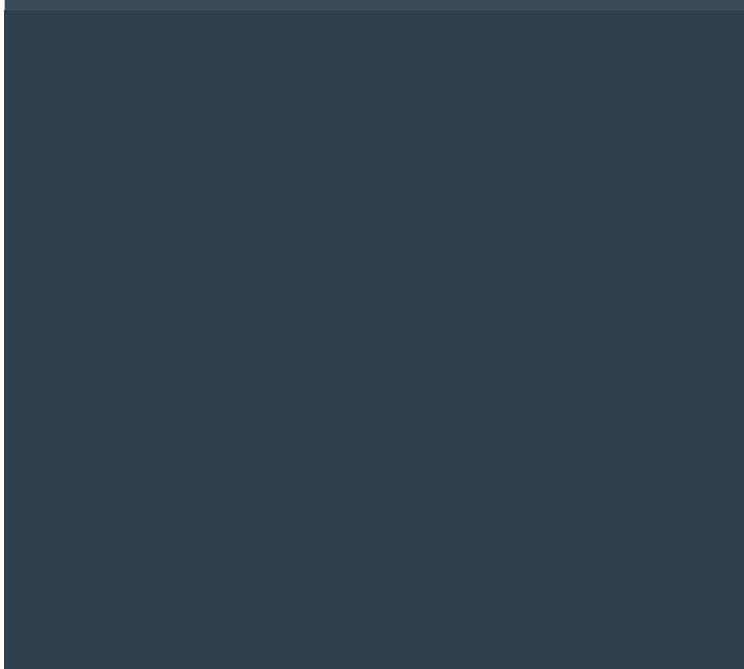


O erro 500 não é tão comum de ser disparado como o 404; no entanto, é importante existir um tratamento correto para que o usuário tenha um retorno amigável quando ele ocorrer. Normalmente, o 500 é disparado quando um script ou a solicitação não pode ser concluída com sucesso.

Por se tratar de um erro amplo dentro das solicitações no lado do servidor, o Next.js também oferece suporte para a geração de uma página estática para tal exceção, evitando que a renderização da página de erro seja feita a cada resposta de erro.

Assim como vimos no erro 404, o Next.js adota a mesma abordagem para a geração da página de erro, bastando, para isso, criar um arquivo em "pages/500.js":

Javascript



Página avançada de erro

Os erros de HTTP no Next.js são tratados por um componente de erro próprio. Entretanto, você pode querer **customizar as páginas para criar tratamentos mais detalhados**; para fazer isso, basta criar um arquivo em “pages/_error.js”.

O código adiante exporta uma função que recebe uma propriedade denominada “statusCode” com o código HTTP do erro. Acessando o código, você pode criar uma página dinâmica de erro.



```
JavaScript
1 function Error({ statusCode }) {
2   return (
3     <p>
4       {statusCode
5       ? `O erro ${statusCode} ocorreu no servidor.`
6       : 'Um erro ocorreu no cliente.'}
7     </p>
8   )
9 }
10
11 Error.getInitialProps = ({ res, err }) => {
12   const statusCode = res ? res.statusCode : err
13   return { statusCode }
14 }
```

pages/_error.js

Para fins didáticos, interceptamos no código da imagem a requisição para testar os códigos de erro com o intuito de condicionar a resposta a ser exibida na página de erro. Esse arquivo é gerado apenas em produção; assim, durante o desenvolvimento, quando o erro ocorrer, a tela exibida será a pilha de camadas para mostrar em que momento o erro foi originado.

Atenção!

Até o momento, esse componente de erro do Next.js não pode ser usado em conjunto com a busca de dados, isto é, quando se utiliza a pré-renderização com “getStaticProps” ou “getServerSideProps”.

Auto-hospedagem

Onde hospedar

Um projeto em Next.js pode ser implantado em **qualquer provedor de hospedagem que tenha o Node.js instalado**, como, por exemplo, o Google Cloud ou a AWS. Além de instalar o servidor Next.js, é possível, utilizando imagens do Docker ou diretamente em uma máquina, realizar a exportação HTML estática, embora existam algumas limitações.

Exportação de HTML

Por meio do comando “next export”, o Next.js exporta toda aplicação para arquivos estáticos HTML. Utilizando esse comando, os arquivos gerados podem ser hospedados em qualquer servidor Web sem a necessidade de um servidor Node.js em funcionamento.

No entanto, essa opção de implantação só será indicada se os recursos do seu projeto não utilizarem funcionalidades dinâmicas de servidor.

Listaremos a seguir os **recursos compatíveis com a exportação de HTML estática**:

- Pré-carregamento do JavaScript;
- Módulos de CSS (folhas de estilo);
- Busca de dados no lado do navegador (cliente);
- A função “getStaticProps”;
- A função “getStaticPaths” (rotas dinâmicas).

Como observamos, apenas recursos que não dependem de um servidor Node.js ou de lógica de programação durante o processo de construção (build) da aplicação podem ser usados na exportação estática de HTML. Logo, funções como “getStaticServerSideProps”, rotas de API, cabeçalhos dinâmicos e otimização de imagens não podem ser usados no projeto.

Plataformas cloud

As plataformas **cloud** são empresas que oferecem serviços de manutenção e disponibilização de recursos de computação sob

demanda. Entre o mix de serviços que essas plataformas oferecem, estão as máquinas para computação de alta performance, a hospedagem de sites e o armazenamento de dados e arquivos.

cloud

Nuvem

Atualmente, essas plataformas são amplamente adotadas no mercado, oferecendo um preço altamente competitivo e uma facilidade de escalar recursos conforme a necessidade. Os preços são um grande atrativo para as empresas contratantes, pois a estrutura de computação em nuvem permite que a cobrança seja feita de acordo apenas com os recursos utilizados, o que traz flexibilidade para as aplicações desenvolvidas.

No mercado, existem **diversas opções de provedores** que disponibilizam uma variedade de serviços. Apontaremos seis deles:

- Amazon Web Services (AWS).
- Microsoft Azure.
- Digital Ocean.
- Google Cloud.
- Heroku.
- Vercel

Usaremos neste conteúdo a Vercel para entender a implantação de uma aplicação Next.js na nuvem. Trata-se de uma plataforma voltada principalmente para sites estáticos e frameworks de front-end; além disso, ela é a responsável pelo desenvolvimento e pela manutenção do Next.js.

Implantando uma aplicação na Vercel

Para fins didáticos, **vamos utilizar a Vercel para implantar uma aplicação Next.js**. Usaremos o plano gratuito, que, no momento da criação deste conteúdo, tem este nome: Hobby.

Para implantarmos nossa aplicação, a primeira coisa que precisamos ter nessa etapa é uma conta gratuita no GitHub. Após a criação da sua conta, crie um repositório, faça o clone para sua máquina e monte um novo projeto Next.js.

A tela adiante exibe o formulário de criação do repositório do Github no qual criaremos uma aplicação simples do Next.js. Nesse exemplo, crie um repositório denominado "app-teste-vercel".

Após o repositório ser criado, faça o clone dele para sua máquina e crie um projeto Next.js, conforme demonstra o comando a seguir:

```
Terminal

1 npx create-next-app app-teste-vercel
```



Assim que o projeto for criado, será necessário fazer o primeiro “commit” com o objetivo de enviar o projeto para o GitHub. Em seguida, acesse o site da Vercel e faça seu login.

Na primeira página, crie um novo projeto. Para ser redirecionado para uma tela, será necessário informar o método de criação do projeto. Em nosso exemplo, nós o criaremos com base no repositório já criado no Github.

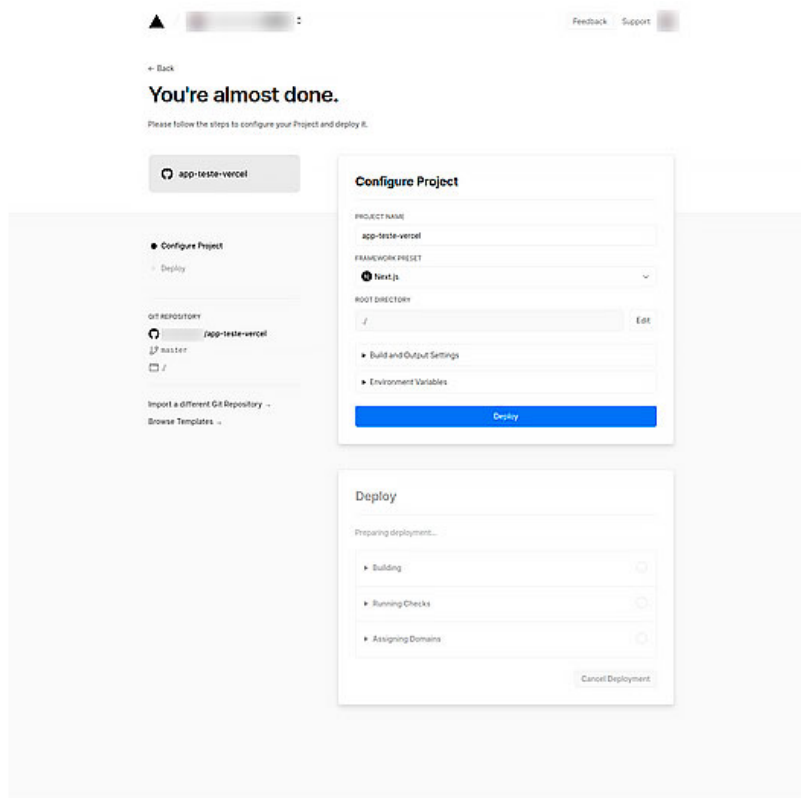
Para isso, clique no botão “Import Third-Party Git Repository →” para escolher o repositório. Após ter acesso à tela a seguir, basta copiar e colar a URL do projeto no seu Github.

The screenshot shows a web form titled "Import a Third-Party Git Repository". At the top right are icons for GitHub, GitLab, and Bitbucket. The main instruction is "Enter the URL of a Git repository to deploy it:". Below this is a text input field containing the URL "https://some-provider.com/some-organization/some-project". At the bottom of the form are two buttons: "Back" on the left and "Continue" on the right.

No campo “GIT SCOPE”, clique em adicionar repositório, escolhendo a opção “Add GitHub Account”. Você será redirecionado para uma tela no GitHub na qual vai autorizar o acesso da Vercel aos seus repositórios. Escolha o repositório que criamos: “app-teste-vercel”.

The screenshot shows a web form titled "Create Git Repository". It includes a sub-header: "To ensure you can easily update your project after deploying it, a Git repository must be created. Every push to that Git repository will be deployed automatically." Below this, there are two input fields: "GIT SCOPE" with a dropdown menu showing a GitHub icon, and "REPOSITORY NAME" with a text input field containing "my-repository". At the bottom left is a checkbox labeled "Create private Git Repository", which is currently unchecked. At the bottom right is a blue button labeled "Create".

Após a escolha do repositório, basta importar na tela que vai aparecer em sequência, clicando em “Deploy”.

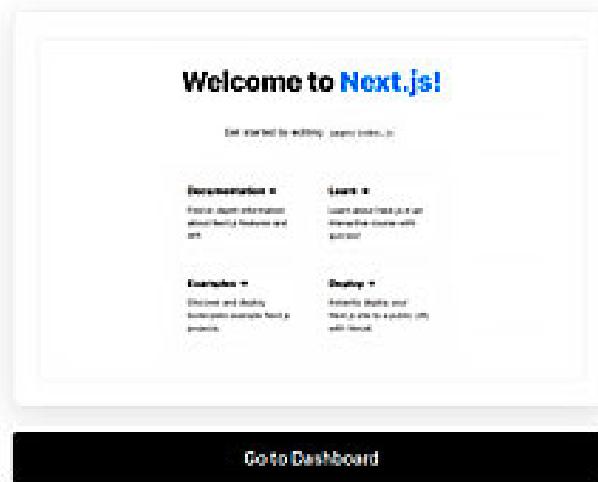


Agora a Vercel executará a compilação e a implantação da aplicação. Assim que o processo terminar, você será redirecionado para uma tela semelhante a esta:



Congratulations!

You just deployed a new Project to Vercel.



GETTING STARTED

Run `next dev` to run your project locally

PREVIEW

Push to any Git branch other than `main` to preview changes

DEPLOY

Push to `main` to ship changes to production

Clicando em “Go to Dashboard”, você poderá acessar uma página com todas as informações da implantação, como, por exemplo, status do servidor, tempo de criação, qual ramo do GIT foi usado para o “deploy” e o link de acesso da aplicação.

Pronto: agora você já pode criar seus projetos Next.js e testá-los em produção para ver o funcionamento dos recursos que precisam de uma estrutura de servidor.



Instalando uma aplicação na nuvem

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Caso deseje exportar sua aplicação Next.js para páginas HTML a fim de que não haja a necessidade de usar um servidor Node.js, você precisa executar o comando

A "next export"

B "next build"

C "next start"

D "next lint"

E "next dev"

Parabéns! A alternativa A está correta.

O comando usado para que o Next.js faça a exportação das páginas em HTML é o "next export". Esse comando fará com que o Next.js gere a aplicação por meio de páginas estáticas, que podem ser executadas de forma independente e sem a necessidade de um servidor Node.js.

Questão 2

De qual tecnologia o Next.js precisa para fazer seu servidor funcionar?

A Java

B PHP

C

RUST

D

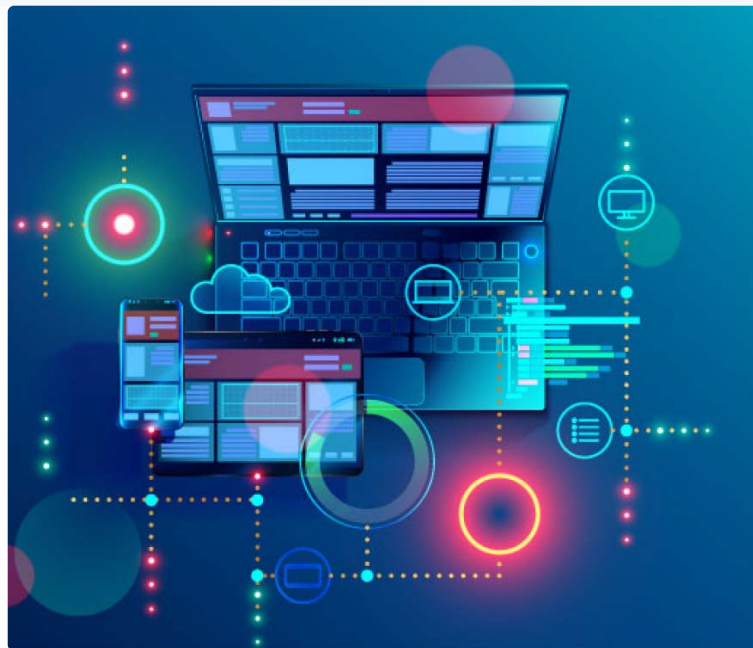
Ruby

E

Node.js

Parabéns! A alternativa E está correta.

O Next.js pode ser implementado em qualquer provedor de hospedagem compatível com o Node.js. No mercado, existem diversas opções de provedores que disponibilizam uma grande variedade de serviços, como Amazon Web Services (AWS), Microsoft Azure, Digital Ocean, Google Cloud, Heroku e Vercel.



4 - Migrando o Next.js para TypeScript

Ao final deste módulo, você será capaz de transferir a aplicação Next.js para TypeScript com as funcionalidades do JavaScript.

Motivação do uso do TypeScript

O que significa TypeScript?

Todo desenvolvedor sabe que um projeto para Web envolve tecnologias, como, por exemplo, JavaScript e CSS. Porém, no momento em que as aplicações começam a ficar maiores, os problemas de produtividade em tempo de desenvolvimento começam a aparecer.

Neste módulo, nosso foco é o JavaScript, já que vamos implementar em nosso projeto Next.js a utilização do **TypeScript**. A principal motivação de se transformar o JavaScript em uma linguagem tipada é justamente aumentar a produtividade e a robustez da aplicação.

Graças à utilização em larga escala do JavaScript, ele deixou de constituir uma linguagem para a criação de pequenos efeitos visuais, passando a ser uma linguagem de suporte à interface por se tratar diretamente da experiência do usuário da aplicação.

Nesse sentido, o **TypeScript é um superconjunto de funcionalidades aplicado ao JavaScript para melhorar o suporte à programação orientada a objetos**, oferecendo várias possibilidades que a linguagem pura não tem. Além disso, durante o desenvolvimento, os editores de código podem usar a inteligência da própria linguagem para autocompletar sintaxes e indicar erros em tempo de desenvolvimento, evitando, com isso, problemas que só seriam vistos depois da compilação.

As principais vantagens de se utilizar o TypeScript

Podemos elencar várias vantagens na utilização do TypeScript em nossas aplicações, mas focaremos apenas as principais, já que o objetivo deste módulo, conforme já frisamos, é implantá-lo em uma aplicação Next.js.

Contar com o superconjunto oferecido pelo TypeScript como ferramenta de desenvolvimento é uma de suas principais vantagens, uma vez que trabalhar com o JavaScript puro pode trazer problemas.

Exemplos

1. Resultados inesperados e pequenos erros difíceis de encontrar, os quais só apareceriam durante a implementação da aplicação. Ao se empregar o TypeScript conforme demonstramos anteriormente, é possível aproveitar o Intellisense do editor, já que ele fornece várias informações que podem ser úteis.

2. Pontos de melhoria e problemas que ocorrem durante a compilação enquanto se está programando, o que economiza um tempo valioso e aumenta significativamente a produtividade. Outro ponto positivo da adoção do TypeScript nos projetos JavaScript é o foco na tipagem estática e nas funcionalidades para a implementação de conceito de programação orientada a objetos. Tipando as variáveis, é possível construir aplicações mais seguras e manuteníveis. Isso reforça o aumento de produtividade em tempo de desenvolvimento, uma vez que os desenvolvedores sabem exatamente o que fazer e como fazê-lo a cada função e troca de variáveis.

No final, é possível entender o **TypeScript como um potencializador da linguagem JavaScript**, trazendo uma maturidade técnica de grandes linguagens de back-end, como PHP e Java, para o universo do desenvolvimento front-end, o que ainda permite que cada vez mais sistemas complexos sejam construídos com agilidade e robustez.

Autocomplete



Com suas variáveis e retornos tipados, os componentes desenvolvidos fazem com que, por meio do próprio editor, seja possível achar as propriedades desejadas, permitindo conferir e garantir que não se vai acessar recursos que não existem.

Tipagem nas bibliotecas



Durante o desenvolvimento da aplicação, é comum usar a importação de bibliotecas, as quais, por muitas vezes, não funcionam como deveriam no autocomplete mencionado no item anterior. No trabalho com o Next.js, é preciso, em vários momentos, utilizar funções, métodos e propriedades predefinidas.

Para usar todas as vantagens do autocomplete, basta importar os conjuntos de tipagens dentro da biblioteca para ter total acesso às propriedades que a função ou o objeto tem. Isso facilita bastante o desenvolvimento e evita, por exemplo, erros de digitação ou a tentativa de acesso a propriedades inexistentes.

Componentes consistentes

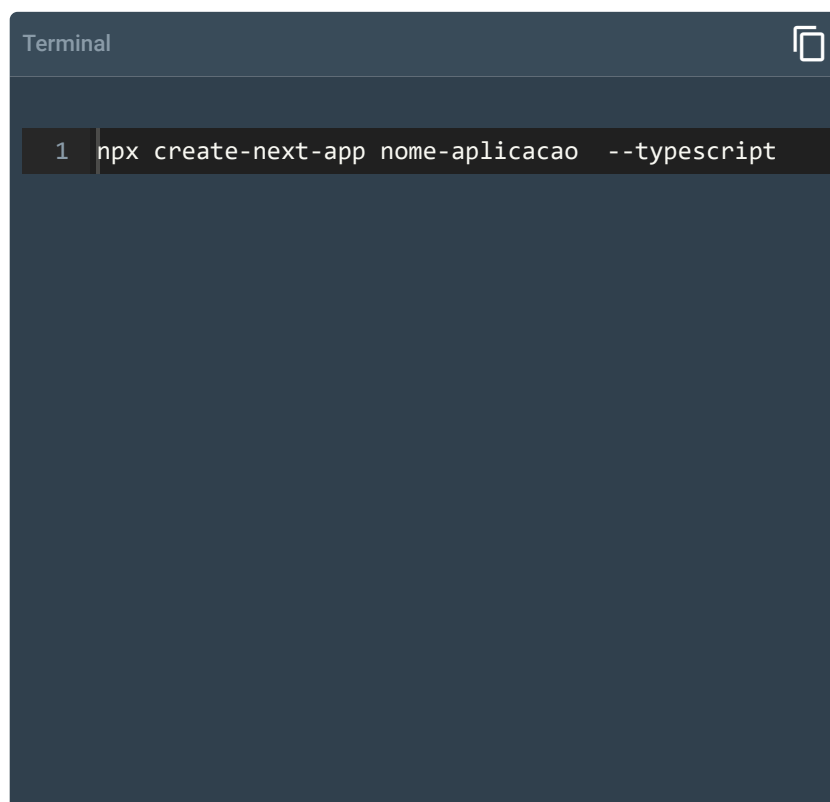


Como já mencionamos, é possível tipar propriedades, explicitando, dessa maneira, o que um componente vai receber e determinando se elas são obrigatórias, públicas ou privadas. Esse fator permite que, durante o desenvolvimento, não sejam passadas propriedades erradas, pois somos informados sobre elas em tempo real.

Projetos Next.js com TypeScript

Criando um projeto

O Next.js tem um comando para criar um projeto já com o suporte do TypeScript. Para tal, basta digitar isto no terminal:

A terminal window with a dark blue background. The title bar says "Terminal" and there is a copy icon in the top right corner. A single line of code is entered: `1 npx create-next-app nome-aplicacao --typescript`.

```
Terminal
```

```
1 npx create-next-app nome-aplicacao --typescript
```

Tal comando se trata exatamente do que vínhamos usando para criar um projeto. Contudo, ele criará dois tipos de arquivo (".tsx" para os componentes de páginas e "ts") em vez de arquivos JavaScript.

Usando o TypeScript em um projeto existente

Podemos migrar um projeto Next.js já existente para que ele passe a ter suporte ao TypeScript. Para começar, é necessário criar um arquivo vazio na raiz do diretório do projeto denominado “tsconfig.json”.

O Next.js configurará automaticamente o arquivo “tsconfig.json” com os valores-padrão necessários assim que for executado o comando “npm run dev”. Na tela do terminal, o Next.js guiará as instalações dos pacotes necessários para finalizar a configuração do TypeScript. Ao término dessa etapa, basta fazer a conversão das extensões do arquivo “.jsx” para “.tsx” e do “.js” para “.ts”.


Atenção!

Fique atento: um arquivo chamado “next-env.d.ts” será criado na raiz do diretório do projeto. Ele é usado pelo compilador do TypeScript; assim, você não deve alterá-lo nem removê-lo.

Ignorar erros do TypeScript

Uma aplicação Next.js falhará na compilação (“next build”) quando o código tiver algum erro de TypeScript. Embora seja algo totalmente desaconselhável, é possível, por sua conta e risco, determinar que o Next.js ignore a verificação integrada de TypeScript e execute a aplicação de maneira forçada.

Para desabilitar a verificação de compilação TypeScript do Next.js, abra o arquivo “next.config.js” na raiz do diretório e inclua a propriedade “ignoreBuildErrors” dentro da propriedade “TypeScript”, como mostra o código a seguir:

```
JavaScript   
  
1 module.exports = {  
2   typescript: {  
3     ignoreBuildErrors: true,  
4   }  
5 },
```

next.config.js

Atalho de caminho para "baseUrl"

O Next.js suporta de maneira nativa as opções "paths" e "baseUrl" nas configurações de TypeScript declaradas no arquivo "tsconfig.json".

Essas opções permitem que você configure atalhos para módulos, fazendo com que o código fique mais claro, como vamos observar nos códigos adiante.

Imagine que eu deseje organizar meus diretórios de componentes, querendo evitar que, ao importá-los, isso seja feito através dos caminhos relativos. Para fugir dessa poluição nas URLs de importação, eu posso, dentro do arquivo "tsconfig.json", determinar diretórios por meio da propriedade "paths".

```
// tsconfig.json
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@componentes/*": ["componentes/*"]
    }
  }
}
```

```
// tsconfig.json
{
  "compilerOptions": {
    "baseUrl": ".",
    "paths": {
      "@componentes/*": ["componentes/*"]
    }
  }
}
```

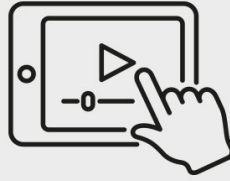
```
// pages/index.js
import Botao from '@componentes/botao'

export default function HomePage() {
  return (
    <>
    <h1>Bem vindos!</h1>
    <Botao />
    </>
  )
}
```



Usando TypeScript

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Qual é a diferença entre TypeScript e JavaScript?

A

Não há diferenças: TypeScript e JavaScript são a mesma linguagem de programação.

B

TypeScript é um superconjunto de JavaScript, ou seja, um conjunto de ferramentas e formas mais eficientes de escrever um código JavaScript, adicionando recursos que não estão presentes de maneira nativa na linguagem.

C

O TypeScript é responsável por permitir a programação orientada a objetos em JavaScript.

D

O JavaScript é uma linguagem de programação para front-end; o TypeScript, um superconjunto que permite ao JavaScript ser usado para o back-end.

E

TypeScript é o nome comercial para o ECMAScript 6, enquanto o JavaScript é o mesmo que ECMAScript 5.

Parabéns! A alternativa B está correta.

O TypeScript começou a ser desenvolvido pela Microsoft em 2012 com o objetivo de adicionar recursos e ferramentas que não estavam presentes nativamente na linguagem, como tipagem estática e orientação a objetos. Por esse motivo, ele não é considerado uma nova linguagem de programação, e sim um superconjunto de JavaScript. Todo o código é “transformado” (transcompilado) em JavaScript antes de ser executado.

Questão 2

Qual arquivo o Next.js usa para fornecer opções ao compilador de TypeScript?

- A next.config.js
- B next-env.d.ts
- C package.json
- D tsconfig.json
- E splint-config-next.json

Parabéns! A alternativa D está correta.

Em um diretório, o arquivo tsconfig.json indica que o projeto utiliza o TypeScript. Esse arquivo especifica as configurações necessárias usadas na compilação de TypeScript para JavaScript do projeto. Embora esse arquivo seja usado pelo Next.js, ele sempre existirá em projetos que fazem uso do TypeScript.

Considerações finais

Vimos neste conteúdo que as aplicações front-end escaláveis requerem uma série de configurações e cuidados com a estrutura de diretórios, a nomenclatura de componentes e uma infinidade de detalhes, itens que deixarão seu projeto muito mais preparado para crescer durante o ciclo de desenvolvimento.

Ao falarmos que ReactJs é uma biblioteca que possui um ecossistema enorme e nada opinativo no quesito padrões de desenvolvimento, ficamos expostos a decisões ruins, impactando o futuro do projeto. Salientamos, desse modo, que o framework Next.js oferece justamente as funcionalidades para o desenvolvedor resolver problemas recorrentes, ajudando-o a construir aplicações muito mais robustas.

Verificamos ainda que a utilização do Next.js oferece várias funcionalidades que ampliam as possibilidades de acelerar o processo de desenvolvimento de uma aplicação ReactJs. Sua capacidade de resolver dificuldades comuns em aplicações SPA, como renderização híbrida e estática, tratamento de rotas e suporte a TypeScript, por exemplo, faz essa tecnologia ser uma das mais adotadas hoje em dia não só para atender à criação de sistemas de informação, mas também para lidar com e-commerces, landing pages e sites estáticos.

Demonstramos, com isso, que o foco em agilidade de desenvolvimento e performance de carregamento da aplicação do Next.js garante uma escalabilidade para os projetos e uma confiança na entrega do produto tanto na qualidade de código quanto na robustez da aplicação.

Podcast

Para encerrar ouça mais sobre os conceitos de Next.js.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.



Explore +

Pesquise como o Google indica a preparação de uma **SPA** para deixá-la apta a ser indexada pelo seu mecanismo de busca.

Leia, na Central de Pesquisa Google, o **Guia detalhado de como a Pesquisa Google funciona**.

Acesse o **Guia de otimização de mecanismos de pesquisa (SEO) para iniciantes**, a fim de entender todos os aspectos que o Google leva em consideração para indexar os resultados de suas consultas.

Referências

NEXTJS. **Getting started**. Docs. Consultado na Internet em: 22 jun. 2022.

REACT. **Getting started**. Docs. Consultado na Internet em: 22 jun. 2022.



Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.



Download material

O que você achou do conteúdo?



Relatar problema