5 - Páginas Estáticas (SSG) no Next.js

E aí, programador! o/

Nessa aula vamos conhecer o recurso de geração de sites estáticos do Next.js, um dos seus recursos mais populares.

1. Para começarmos a ver como as páginas estáticas funcionam vamos criar uma nova página chamada static.tsx na pasta “pages”. Dentro dela adicione o seguinte conteúdo:

   Obs.: Repare que nossa página está quase igual ao arquivo dynamic.tsx, isso porque aqui também usaremos o fetch do lado do cliente para comparar os resultados.

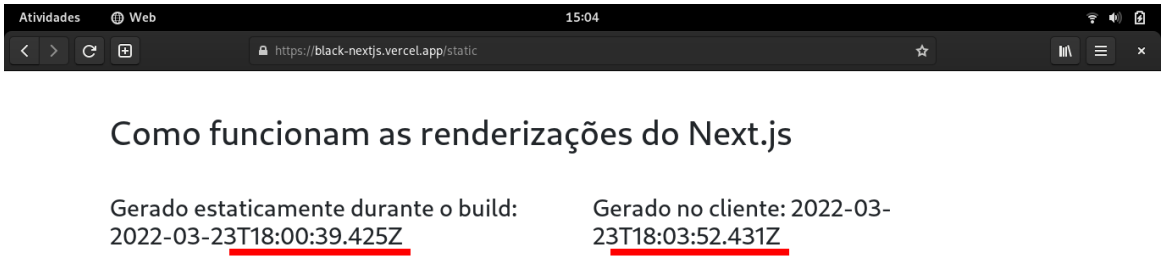
```
// pages/static.tsx

import { NextPage } from "next"
import { useEffect, useState } from "react"
import { Col, Container, Row } from "reactstrap"

type ApiResponse = {
  name: string
  timestamp: Date
}

const Static: NextPage = () => {
  const [clientSideData, setClientSideData] = useState<ApiResponse>()

  useEffect(() => {
    fetchData()
  }, [])

  const fetchData = async () => {
    const data = await fetch(`/api/hello`).then(res => res.json())
    setClientSideData(data)
  }

  return (
    <Container tag="main">
      <h1 className="my-5">
        Como funcionam as renderizações do Next.js
      </h1>

      <Row>
        <Col>
          <h3>
            Gerado estaticamente durante o build:
          </h3>
        </Col>

        <Col>
          <h3>
            Gerado no cliente: {clientSideData?.timestamp}
          </h3>
        </Col>
      </Row>
    </Container>
  )
}
```

```
export default Static
```

2. Se testarmos a página veremos que ela funciona como antes. Agora vamos adicionar a função getStaticProps, que indica para o Next.js que essa página vai ter dados dinâmicos obtidos durante o build e então disponibilizados no HTML estático que será construído:

```
// pages/static.tsx

import { GetStaticProps, NextPage } from "next"
import { ReactNode, useEffect, useState } from "react"

// ...

export const getStaticProps: GetStaticProps = async () => {
  const staticData = await fetch(`${process.env.NEXT_PUBLIC_APIURL}/api/hello`).then(res => res.json())

  return {
    props: {
      staticData
    }
  }
}

const Static: NextPage = (props: {
  children?: ReactNode
  staticData?: ApiResponse
}) => {
  const [clientSideData, setClientSideData] = useState<ApiResponse>()

// ...

  return (
    <Container tag="main">
      <h1 className="my-5">
        Como funcionam as renderizações do Next.js
      </h1>

      <Row>
        <Col>
          <h3>
            Gerado estaticamente durante o build: {props.staticData?.timestamp}
          </h3>
        </Col>

        <Col>
          <h3>
            Gerado no cliente: {clientSideData?.timestamp}
          </h3>
        </Col>
      </Row>
    </Container>
  )
}

export default Static
```

3. Se testarmos nossa aplicação no ambiente de desenvolvimento não notaremos tanta diferença, porque o Next.js reconstrói o nosso site sempre que fazemos uma nova requisição, então a propriedade estática vai estar sempre se atualizando. Mas, se fizermos o commit e o push para atualizar nossa aplicação em produção vamos conseguir notar claramente a diferença, pois a propriedade estática ficou “congelada” na versão que foi executada durante o build, e não é mais atualizada:



4. A construção de páginas estáticas tem muitos benefícios e é capaz de atender muitos casos de uso. Até mesmo em situações onde o conteúdo dinâmico do site é atualizado com uma frequência não tão alta é possível realizar um novo build quando houver atualizações. Mas se por acaso o seu cenário requer uma taxa de atualização alta, necessita de SEO e não quer perder desempenho renderizando páginas no servidor existe um outro recurso do Next.js que pode ajudar.

   Esse recurso se chama ISR, ou *incremental static regeneration*, que é uma forma de atualizar páginas estáticas do site **depois** de realizar o build. E tudo que você precisa fazer é adicionar uma propriedade “revalidate” na função getStaticProps:

```
// pages/static.tsx

// ...

export const getStaticProps: GetStaticProps = async () => {
  const staticData = await fetch(`${process.env.NEXT_PUBLIC_APIURL}/api/hello`).then(res => res.json())

  return {
    props: {
      staticData
    },
    revalidate: 10
  }
}

// ...
```

5. Agora, ao fazermos commit e push e então testarmos nossa aplicação na Vercel veremos algo muito interessante. Ao abrirmos a página vamos receber um valor gerado estaticamente e se continuarmos atualizando repetidamente a página esse valor se mantém inalterado enquanto os dados obtidos no lado do cliente são atualizados a cada atualização. No entanto, depois de 10 segundos, que foi o tempo que definimos na propriedade revalidate, o valor gerado estaticamente é atualizado, trazendo uma hora mais recente.