Professor Rodrigo Bossini
https://www.rodrigobossini.com.br

NextJS
Páginas e Roteamento

1 Introdução

Neste material, criaremos uma aplicação que nos permitirá estudar os recursos fundamentais do NextJS.

2 Desenvolvimento

2.1 (NodeJS) Certifique-se de que você possui uma versão recente do NodeJS instalada no seu computador. Procure fazer a instalação do NodeJS usando um Node Version Manager.

2.2 (Workspace) Crie uma pasta que terá como finalidade abrigar diversas subpastas. Cada uma delas será um projeto NextJS. Lembre-se de evitar nomes com espaço em branco, caracteres especiais e de não utilizar diretórios sob restrições impostas pelo sistema operacional. Para usuários do Windows, o diretório pode ser algo como **C:\Users\rodri\Documents\workspaces\pessoal\nextjs**.

2.3 (Criando um projeto) Abra um terminal e navegue até o seu workspace com

cd path/do/seu/workspace

Use

npx create-next-app fundamentos-nextjs

Uma pasta chamada **fundamentos-nextjs** será criada. Use

cd fundamentos-nextjs

Produção: outubro de 2021

no terminal e

code .

para abrir uma instância do VS Code vinculada a ela. No VS Code, clique **Terminal >> New Terminal** para obter uma instância de um terminal vinculada ao VS Code, o que simplifica o trabalho. No terminal, use

npm run dev

para executar a aplicação em modo de desenvolvimento.

2.4 (Breve limpeza) O template criado automaticamente inclui alguns arquivos e pastas que não nos serão necessários no momento. São eles:

- pasta **api** e seu conteúdo
- arquivo **Home.module.css**

Por isso, apague todos eles de seu projeto. **Apague** também todo **o conteúdo do arquivo index.js**.

2.5 (Página para exibição de artigos) Nossa aplicação exibirá uma coleção de artigos, cada qual composto por um título e um texto detalhado. Desejamos que a página esteja disponível no endereço

host:porta/artigos

Para utilizar o mecanismo de roteamento do NextJS, basta criar um arquivo chamado **artigos.js** na pasta **pages**. Vá em frente e faça isso, crie o arquivo.

OBS: O conteúdo que definirmos no arquivo **index.js** poderá ser acessado por meio do endereço

host:porta

2.6 (Código inicial para o componente principal e para o componente de exibição de artigos) Cada página de nossa aplicação será definida como um componente React comum. O componente principal deve ser criado no arquivo **index.js**. Veja seu conteúdo inicial.

Nota. Embora não seja obrigatório, utilizaremos o sufixo **Page** nos nomes dos componentes que representam páginas da aplicação.

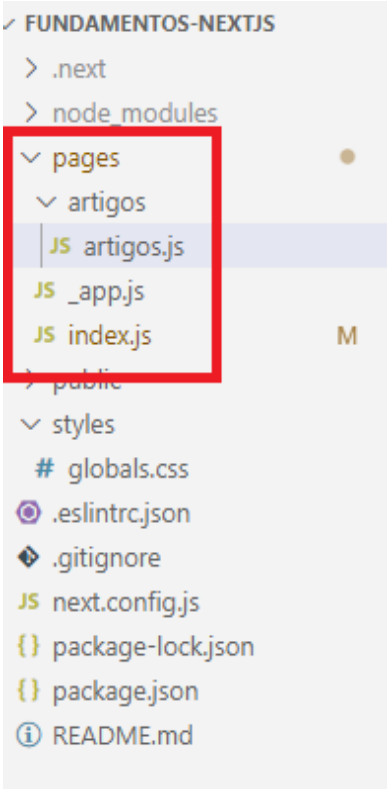
```
const PrincipalPage = () => {
  return <h1>
    Página principal
  </h1>
}
export default PrincipalPage
```

O componente de exibição de artigos tem definição inicial praticamente idêntica. Ele é definido no arquivo **artigos.js**.

```
const ArtigosPage = () => {
    return <h1>
      Artigos
    </h1>
  }
  export default ArtigosPage
```

No seu navegador, visite os endereços **localhost:3000** e **localhost:3000/artigos**. Eles devem dar acesso ao conteúdo do componente principal e do componente de artigos, respectivamente. Perceba que navegamos até a página "artigos" apenas utilizando o nome do arquivo em que o seu componente está definido. Não foi necessário realizar nenhuma configuração ou escrever código específico para isso. Esse é o **roteamento baseado em arquivos** do NextJS.

2.7 (Subpastas também podem participar do mecanismo de roteamento) Também é possível criar subpastas da pasta **pages** cujos nomes farão parte das URLs visitadas pelos usuários. Para testar esse mecanismo, crie uma pasta chamada **artigos**, subpasta de **pages**. A seguir, arraste o arquivo **artigos.js** para dentro dela. Ou seja, ele sai de onde se encontra no momento e passa a existir somente dentro da nova pasta. Desse jeito.



Se você visitar o endereço **localhost:3000/artigos**, a aplicação exibirá um típico erro 404, indicando que o recurso não foi encontrado. Ele pode agora ser encontrado no endereço **localhost:3000/artigos/artigos**. Essa construção pode ser de interesse para que agrupemos todos os recursos relacionados a artigos, para que eles possam todos ser acessados como localhost:3000/artigos/artigo1, localhost:3000/artigos/artigo2 e assim por diante.

Entretanto, o acesso da página inicial da seção de artigos pode ser mais intuitivo se ele puder ser feito por meio do endereço **localhost:3000/artigos** apenas. Para isso, basta renomear o arquivo **artigos.js** para **index.js**. Lembre-se de mantê-lo dentro da pasta **artigos**. Observe que, por padrão, quando acessamos a raiz de uma pasta, o

conteúdo entregue é aquele definido em um arquivo chamado **index.js** que deve ser criado dentro dessa pasta.

Depois de mudar o nome do arquivo **artigos.js** para **index.js**, visite novamente o endereço **localhost:3000/artigos**. O navegador deverá exibir o conteúdo produzido pelo componente definido neste arquivo.

Crie também um arquivo chamado **programacao-com-java.js** na pasta **artigos**. A ideia é que Programação com Java seu o tópico sobre o qual este artigo trata. Veja seu conteúdo inicial.

```
const DetalhesPage = () => {
    return <h1>Detalhes</h1>
}
export default DetalhesPage
```

Visite o endereço **localhost:3000/artigos/programacao-com-java** no seu navegador e visualize o conteúdo produzido por este novo componente.

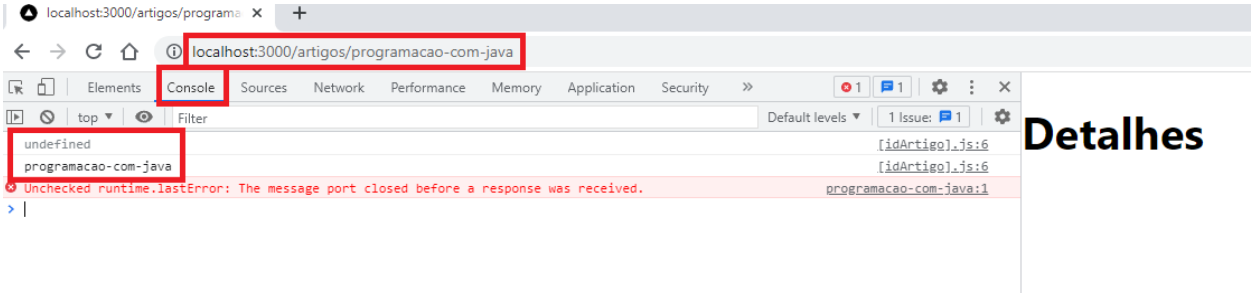
2.8 (Páginas dinâmicas) Observe que o componente criado no arquivo **programacao-com-java.js** - e muitos outros, evidentemente – tem o potencial de ser reutilizado para a exibição de conteúdos diferentes. Para cada artigo existente em uma base de dados, por exemplo, desejamos utilizar o mesmo componente definido neste arquivo para exibir os seus detalhes. Precisamos de **páginas dinâmicas**. Em particular, o nome do arquivo também deve, de alguma forma, ser definido em função do artigo que será exibido. Afinal, ele fará parte da URL a ser utilizada pelo usuário. Para definir uma página dinâmica, utilizamos o operador **[ ]** no nome do arquivo.

Façamos um teste. Renomeie o arquivo **programacao-com-java.js** para **[idArtigo].js**. Observe que o nome **idArtigo** pode ser escolhido por você. É apenas uma espécie de placeholder para ser utilizado adiante. A partir de agora, é possível visitar endereços como **localhost:3000/artigos/programacao-com-java**, **localhost:3000/artigos/programacao-com-python**, **localhost:3000/artigos/abc** e assim por diante. Todos eles levam a esta página cujo nome do arquivo inclui o operador **[ ]**.

2.9 (Obtendo trechos da URL) Quando o componente de detalhes é utilizado, a URL visitada contém um texto arbitrário, conhecido somente em tempo de requisição. Intuitivamente, precisaremos de acesso a esse texto que pode, por exemplo, ser o id de um artigo, para exibir o conteúdo adequadamente. Ele pode ser extraído utilizando-se um **hook** provido pelo próprio **next**. Ele se chama **useRouter**. Observe como ele pode ser utilizado. Estamos no arquivo **[idArtigo].js**.

```
import { useRouter } from "next/router"
const DetalhesPage = () => {
    const router = useRouter()
    //utilizamos o placeholder que faz
    //parte do arquivo aqui, veja
    console.log (router.query.idArtigo)
    return <h1>Detalhes</h1>
}
export default DetalhesPage
```

Visite o endereço **localhost:3000/artigos/programacao-com-java** e, no Chrome Dev Tools (CTRL + SHIFT + I no seu navegador) vá até a aba console. Visualize algo assim.



Observe que há a exibição de um "undefined". Isso ocorre pois o hook executa uma primeira vez assim que o componente renderiza e, neste momento, a informação ainda não está disponível. Assim que ela se torna disponível o router tem acesso a ela e o componente renderiza de novo.

2.10 (Navegando entre componentes) O próximo problema a ser resolvido é um tanto natural: como navegar entre componentes? Para verificar como o mecanismo de navegação funciona, digamos que a página inicial de artigos – arquivo **index.js da pasta artigos** – exibe uma lista de artigos. Veja.

Nota. Fragment é um componente React que nos permite agrupar elementos sem que um elemento "pai" para eles seja criado. Imagine que a expressão JSX possui um **div** na raiz, mas que esse div não será criado na árvore DOM. Essa é a ideia. Veja o que a documentação diz sobre ele.

"A common pattern in React is for a component to return multiple elements. Fragments let you group a list of children without adding extra nodes to the DOM."

Leia mais aqui.

https://reactjs.org/docs/fragments.html

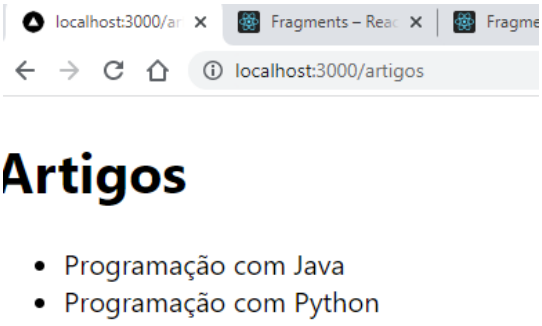
```
import { Fragment } from "react"
const ArtigosPage = () => {
    return (
      <Fragment>
        <h1>
          Artigos
        </h1>
        <ul>
          <li>Programação com Java</li>
          <li>Programação com Python</li>
        </ul>
      </Fragment>
    )

  }
  export default ArtigosPage
```

A documentação de Fragment também nos ensina que há uma nova forma para utilizá-lo. Uma espécie de "syntax sugar". Substituímos as tags **<Fragment> </Fragment>** pelas mais enxutas **<> </>**. Veja.

```
const ArtigosPage = () => {
    return (
      <>
        <h1>
          Artigos
        </h1>
        <ul>
          <li>Programação com Java</li>
          <li>Programação com Python</li>
        </ul>
      </>
    )

  }
  export default ArtigosPage
```

Nota. Observe que somente Fragments que utilizam a tag Fragment explicitamente podem definir um atributo **key**. Além disso, segundo a documentação, este é o único atributo que um Fragment pode ter. **Não é possível, por exemplo, tratar eventos de clique sobre Fragments**.

Caso visite o endereço localhost:3000/artigos agora, o resultado deve ser esse aqui.



Para fazer a navegação acontecer, podemos utilizar um elemento **a** - de **anchor**, lembra? - comum do HTML. Veja.

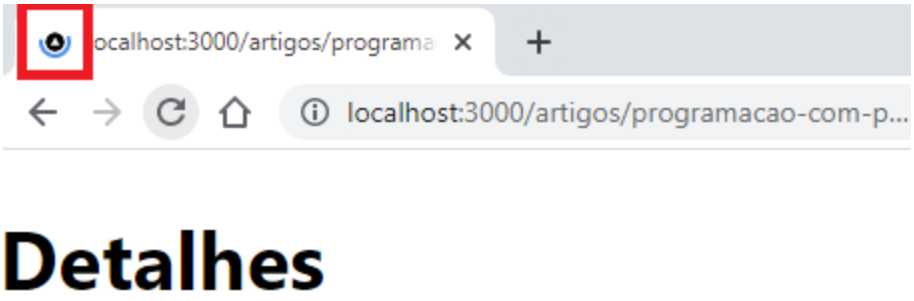
```
const ArtigosPage = () => {
    return (
      <>
        <h1>
          Artigos
        </h1>
        <ul>
            <li>
              <a href="/artigos/programacao-com-java">
                Programação com Java
              </a>
            </li>
            <li>
              <a href="/artigos/programacao-com-python">
                Programação com Python
              </a>
            </li>
        </ul>
      </>
    )

  }
  export default ArtigosPage
```

Visite o endereço **localhost:3000/artigos** e clique sobre os links. Repare que cada texto, na verdade, não tem aparência de link. Isso está acontecendo pois há uma regra CSS definida no arquivo **globals.css** assim.

```
a {
  color: inherit;
  text-decoration: none;
}
```

Não é necessário, mas, se desejar, você pode remover esta regra.

2.11 (Single Page Application e o elemento a: o que acontece quando clicamos atualizar no navegador?) Muito embora a navegação funcione, ela tem uma característica que pode ser indesejável. Para verificar, vá até o endereço **localhost:3000/artigos** novamente. Clique sobre um dos artigos. **Quando estiver sendo levado da página de artigos para a página específica**, fique de olho na parte destacada a seguir, no seu navegador. Repare que um ícone "loading" aparece no canto superior esquerdo da aba.

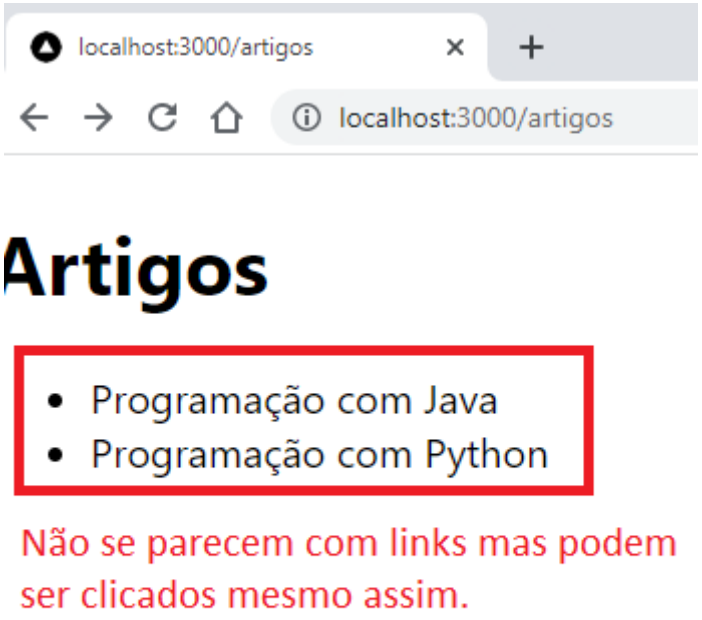


O que está acontecendo é o seguinte: a cada vez navegamos até uma página de detalhes, o navegador está fazendo uma nova requisição para obter o mesmo HTML. Ou seja, a nossa aplicação **não é exatamente uma SPA**. Além disso, um **novo componente React** está sendo renderizado, o que quer dizer que um eventual **estado** existente anteriormente é completamente perdido.

(O componente Link do NextJS) Para resolver este problema, o NextJS oferece um componente chamado **Link**. Veja como ele pode ser usado. Basta substituir a tag "**a**" por um componente do tipo **Link**.

```
import Link from 'next/link'

const ArtigosPage = () => {
    return (
      <>
        <h1>
          Artigos
        </h1>
        <ul>
            <li>
              <Link href="/artigos/programacao-com-java">
                Programação com Java
              </Link>
            </li>
            <li>
              <Link href="/artigos/programacao-com-python">
                Programação com Python
              </Link>
            </li>
        </ul>
      </>
    )

  }
  export default ArtigosPage
```

Observe que o elemento gerado pelo componente Link também não terá a aparência padrão de um link comum HTML. Eles podem ser clicados normalmente, apesar disso.



Faça um novo teste clicando em um dos links e fique de olho no que acontece com o navegador, no canto superior esquerdo da aba, como fizemos antes. Repare que o ícone "loading" não aparece dessa vez.

Referências

Next.js by Vercel - The React Framework. 2021. Disponível em <https://nextjs.org>. Acesso em outubro de 2021.