

NextJS

Diversos Recursos

1 Introdução

Neste material, desenvolveremos uma aplicação NextJS que nos permitirá averiguar o funcionamento de diferentes recursos providos pelo framework. Trata-se de uma aplicação que

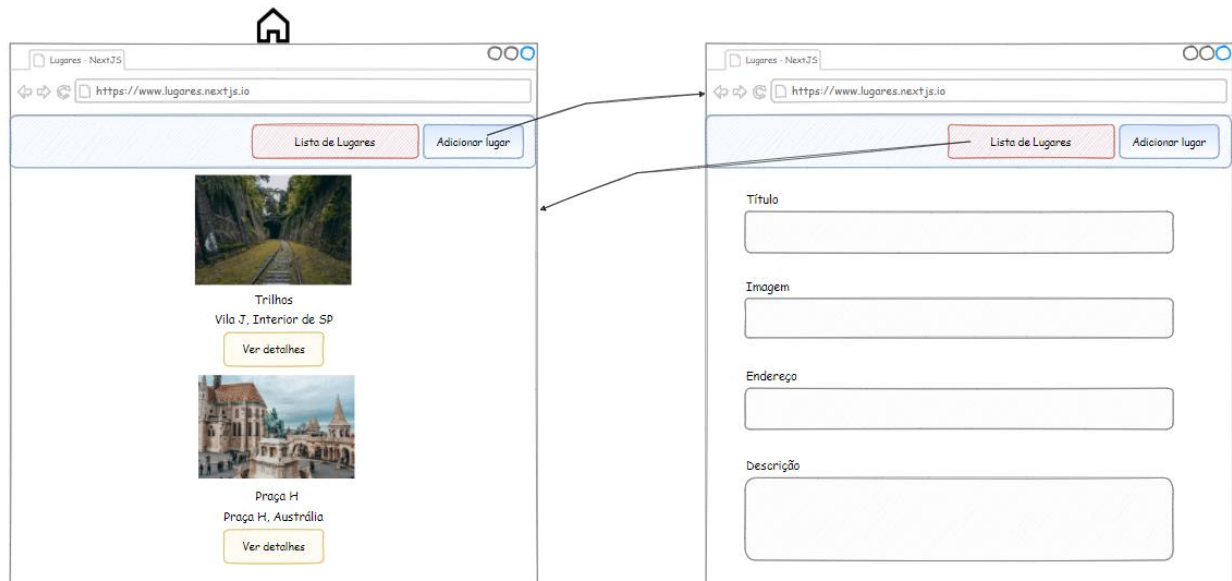
- permite realizar o cadastro de lugares visitados
- permite visualizar uma lista de lugares cadastrados
- permite visualizar os detalhes de um determinado lugar selecionado

Cada lugar tem as seguintes propriedades

- Título
- Foto
- Endereço
- Descrição.

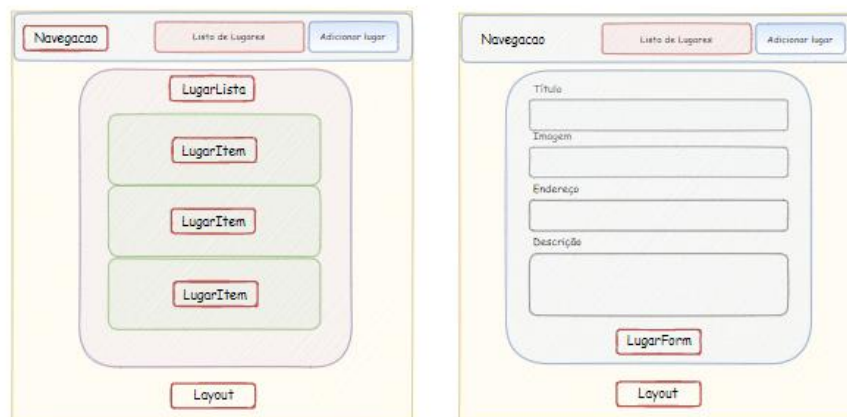
Veja a Figura 1.1.

Figura 1.1



A Figura 1.2 mostra os componentes React que utilizaremos. Estes são apenas os componentes "puramente" React. Ao longo do desenvolvimento, adicionaremos outros que representam páginas que serão gerenciadas pelo NextJS.

Figura 1.2



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 lugares-nextjs
```

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

```
cd lugares-nextjs
```

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 e ajustes) 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.**

Crie uma pasta chamada **components** na raiz da aplicação, lado a lado com a pasta **pages**.

2.5 (Instalação da PrimeReact) Utilizaremos componentes da biblioteca PrimeReact e os utilitários e grid system da PrimeFlex. As suas respectivas documentações podem ser visitadas por meio dos links 2.5.1 e 2.5.2, respectivamente.

Link 2.5.1

<https://www.primefaces.org/primereact/>

Link 2.5.2

<https://www.primefaces.org/primeflex/>

Faça a instalação da PrimeReact com

```
npm install primereact
npm install primeicons
npm install react-transition-group
```

A PrimeFlex pode ser instalada com

```
npm install primeflex
```

A seguir, importe o CSS da PrimeReact, PrimeIcons, PrimeFlex e um dos temas descritos na documentação, na página **Get Started**. Isso pode ser feito no arquivo **pages/_app.js**. Veja.

```
import 'primereact/resources/primereact.min.css'
import 'primeicons/primeicons.css'
import 'primeflex/primeflex.css'
import 'primereact/resources/themes/bootstrap4-light-purple/theme.css'
import '../styles/globals.css'

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

export default MyApp
```

Nota. Arquivos CSS importados no arquivo **pages/_app.js** têm efeito global. Visite o Link 2.5.3 para conhecer as regras sobre o funcionamento do CSS em projetos NextJS.

Link 2.5.3

<https://nextjs.org/docs/basic-features/built-in-css-support>

2.6 (Pastas para os componentes) Os componente React que não representarem páginas NextJS serão armazenados em uma pasta própria para eles. Lado a lado com a pasta **pages**, crie uma pasta chamada **components**. Crie, como subpastas de **components**, duas pastas chamadas **layout** - abrigará um componente que define a barra de navegação e um componente que manipula o layout da tela principal - e **lugar** - abrigará os componentes que definem a exibição de um lugar e de uma lista de lugares.

2.7 (Barra de navegação) Crie um arquivo chamado **Navegacao.js** na pasta **layout**. Utilizaremos um **Toolbar** da PrimeReact para definir nossa barra de navegação. Veja a sua definição no Bloco de Código 2.7.1.

Nota. Descubra mais detalhes sobre o componente Toolbar da PrimeReact no Link 2.7.1.

Link 2.7.1

<https://www.primefaces.org/primereact/showcase/#!/toolbar>

Bloco de Código 2.7.1

```
import { Toolbar } from 'primereact/toolbar'
import { Button } from 'primereact/button'
const Navegacao = () => {
  const botoes = (
    <>
      <Button
        label="Lista de lugares"
        icon="pi pi-list"
        color="warn"
        className="mr-2 p-button-secondary"
      />
      <Button
        label="Novo Lugar"
        icon="pi pi-plus"
      />
    </>
  )
  return (
    <Toolbar right={botoes} />
  )
}

export default Navegacao
```

Faça um teste no arquivo **index.js**. Adicione a ele o conteúdo do Bloco de Código 2.7.2.

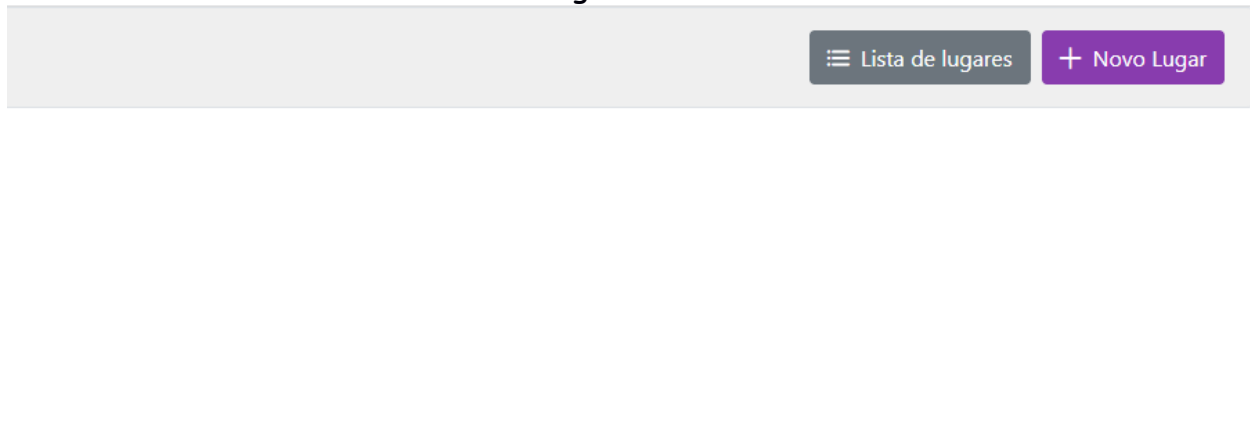
Bloco de Código 2.7.2

```
import Navegacao from "../components/layout/Navegacao"

const HomePage = () => {
  return(
    <Navegacao/>
  )
}
export default HomePage
```

O resultado esperado é exibido na Figura 2.7.1.

Figura 2.7.1



Nota. As cores podem variar dependendo do tema que você tenha escolhido.

2.8 (Componente para exibição de um único lugar) O Bloco de Código 2.8.1 mostra a definição de um componente capaz de fazer a exibição de um único lugar. Para defini-lo, crie um arquivo chamado **LugarItem.js** na pasta **componentes/lugar**. Observe que ele ainda não exibe a descrição do lugar. Isso será feito por outro componente que será acionado quando o botão for clicado.

Bloco de Código 2.8.1

```
import { Button } from 'primereact/button'
const LugarItem = ({titulo, foto, endereco, id}) => {
  return (
    <div className="align-items-center flex-column flex m-3 border border-
round border-1 border-400 p-2">
      <p className="text-center sm:text-base md:text-lg lg:text-
3xl">{titulo}</p>
      <div>
        <img className="border border-round" src={foto} alt={titulo}
width={400}/>
      </div>
      <address className="text-center text-xs md:text-sm lg:text-base mb-
2">{endereco}</address>
      <Button
        label="Ver detalhes"
      />
    </div>
  )
}
export default LugarItem
```

Para testar este componente, ajuste o conteúdo do arquivo **index.js** como destaca o Bloco de Código 2.8.2.

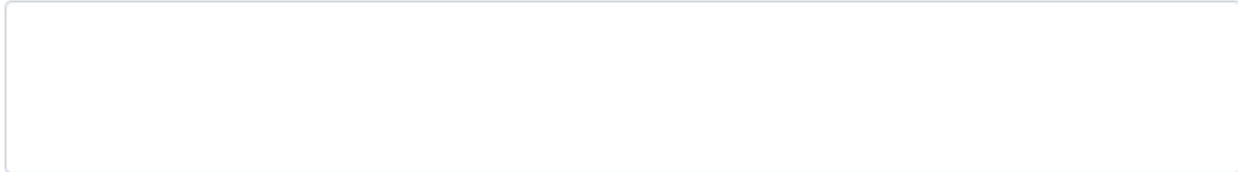
Bloco de Código 2.8.2

```
// import Navegacao from "../components/layout/Navegacao"
import LugarItem from "../components/Lugar/LugarItem"

const HomePage = () => {
  return(
    // <Navegacao/>
    <LugarItem/>
  )
}
export default HomePage
```


Como mostra a Figura 2.8.1, o resultado esperado é apenas uma caixinha com bordas, já que nada ainda lhe foi passado via props.

Figura 2.8.1



2.9 (Componente para exibição de uma lista de lugares) O componente definido no Bloco de Código 2.9.1 apenas mapeia cada lugar recebido via **props** a uma expressão JSX condizente com a forma como desejamos que um lugar seja exibido. Ou seja, a cada lugar, associamos um componente **LugarItem**. Ele deve ser definido num novo arquivo chamado **LugarLista.js**, que deve ser criado na pasta **componentes/lugar**.

Bloco de Código 2.9.1

```
import LugarItem from './LugarItem'
const LugarLista = ({lugares}) => {

  return lugares.map(lugar => (
    <LugarItem
      key={lugar.id}
      id={lugar.id}
      titulo={lugar.titulo}
      foto={lugar.foto}
      endereco={lugar.endereco}
    />
  ))
}

export default LugarLista
```

Para testar o componente, vamos definir uma lista de lugares fictícia e utilizá-lo no componente definido no arquivo **index.js**. Veja o Bloco de Código 2.9.2.

Bloco de código 2.9.2

```
// import Navegacao from "../components/layout/Navegacao"
import LugarItem from "../components/Lugar/LugarItem"
import LugarLista from "../components/Lugar/LugarLista"

const HomePage = () => {
  const lugares = [
    {
      id: '1',
      titulo: 'Trilhos',
      foto: 'https://images.unsplash.com/photo-1556905200-279565513a2d?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1470&q=80',
      endereco: 'Vila J, Interior de SP',
      descricao: 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quaerat harum odio omnis est fugit, incidunt, magni corporis vero aperiam fuga facere ab delectus natus placeat dolor nam quos sint impedit!'
    },
    {
      id: '2',
      titulo: 'Praça H',
      foto: 'https://images.unsplash.com/photo-1549877452-9c387954fbc2?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1170&q=80',
      endereco: 'Praça H, Austrália',
      descricao: 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quaerat harum odio omnis est fugit, incidunt, magni corporis vero aperiam fuga facere ab delectus natus placeat dolor nam quos sint impedit!'
    }
  ]
  return(
    // <Navegacao/>
    // <LugarItem/>
    <LugarLista lugares={lugares} />
  )
}
export default HomePage
```

Veja o resultado esperado na Figura 2.9.1.

Figura 2.9.1

Trilhos



Vila J, Interior de SP

[Ver detalhes](#)

Praça H



Praça H, Austrália

[Ver detalhes](#)

O componente chamado **Layout**, a ser criado em um arquivo chamado **Layout.js** na pasta **componentes/layout** será responsável por lidar com a responsividade da página. Ele também se encarrega de exibir uma barra de navegação. Veja seu conteúdo no Bloco de Código 2.10.1.

Bloco de Código 2.10.1

```
import Navegacao from "../Navegacao"
const Layout = ({children}) => {
  return (
    <div className="grid justify-content-center" >
      <div className="col-12">
        <Navegacao/>
      </div>
      <div className="col-12 md:col-10 lg:col-8 border border-
round border-1 border-400">
        {children}
      </div>
    </div>
  )
}

export default Layout
```

Passa a utilizá-lo no arquivo **index.js** como mostra o Bloco de Código 2.10.2.

Bloco de Código 2.10.2

```
import Layout from '../components/layout/Layout'
import LugarLista from '../components/lugar/LugarLista'

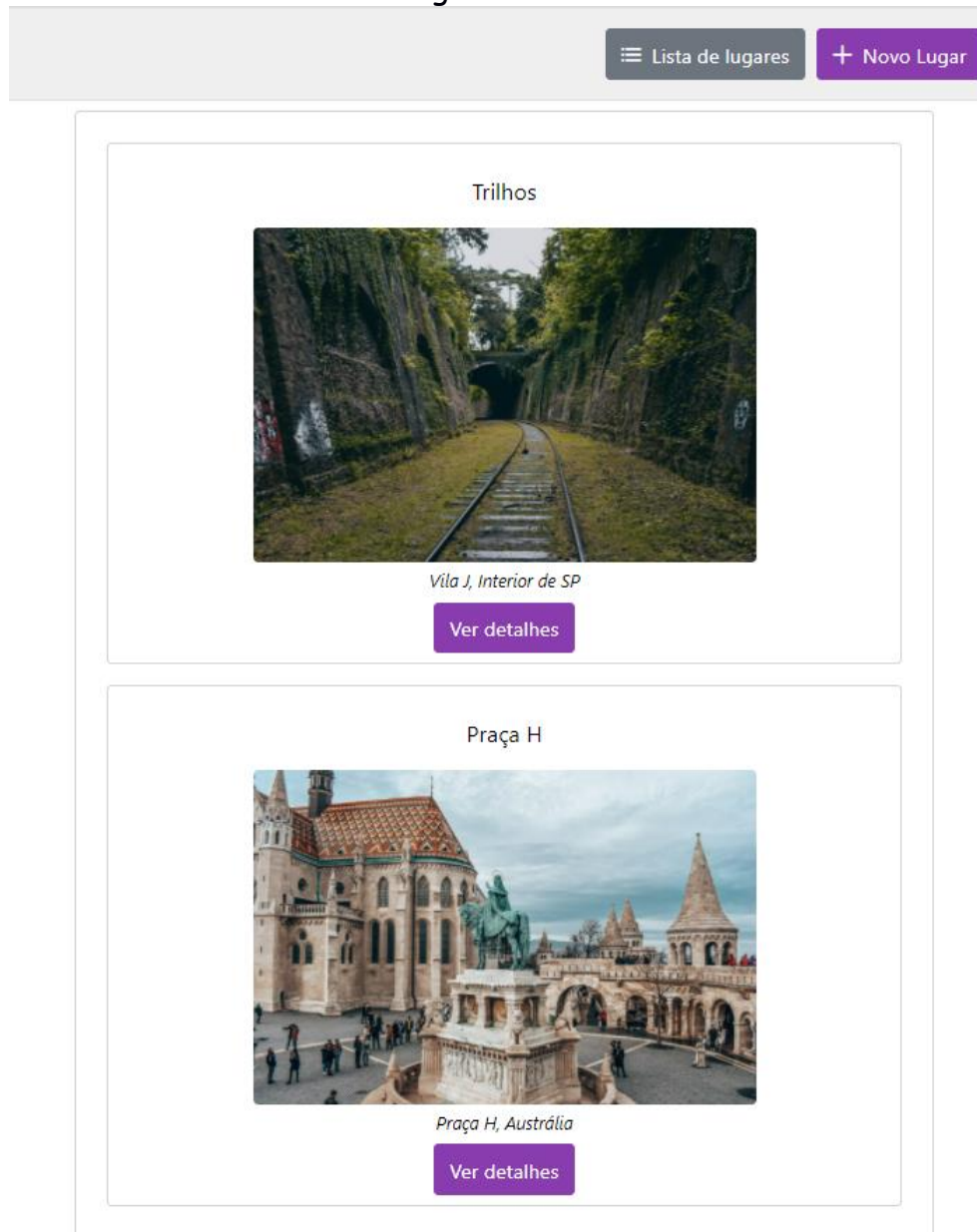
const HomePage = () => {
  const lugares = [
    {
      id: '1',
      titulo: 'Trilhos',
      foto: 'https://images.unsplash.com/photo-1556905200-279565513a2d?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1470&q=80',
      endereco: 'Vila J, Interior de SP',
      descricao: 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quaerat harum odio omnis est fugit, incidunt, magni corporis vero aperiam fuga facere ab delectus natus placeat dolor nam quos sint impedit!'
    },
    {
      id: '2',
      titulo: 'Praça H',
      foto: 'https://images.unsplash.com/photo-1549877452-9c387954fbc2?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1170&q=80',
      endereco: 'Praça H, Austrália',
      descricao: 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quaerat harum odio omnis est fugit, incidunt, magni corporis vero aperiam fuga facere ab delectus natus placeat dolor nam quos sint impedit!'
    }
  ]

  return(
    <Layout>
      <LugarLista lugares={lugares}/>
    </Layout>
  )
}

export default HomePage
```

Veja o resultado esperado na Figura 2.10.1. Repare que não mudou muita coisa. O que mudou foi a quantidade de espaço horizontal sendo utilizado. Essa quantidade passa a ser escolhida em função do tamanho da tela do dispositivo.

Figura 2.10.1



2.11 (Componente para a adição de novos lugares) A adição de lugares será feita por meio de um form que oferece campos para o usuário informar título, URL da foto, endereço e descrição. Para defini-lo, crie um arquivo chamado **LugarForm.js** na pasta **componentes/lugar**. Veja sua definição nos blocos de código 2.11.1 e 2.11.2.

Bloco de Código 2.11.1

```
import { useState } from 'react'
import { Button } from 'primereact/button'
const LugarForm = (props) => {
  const [titulo, setTitulo] = useState('')
  const [foto, setFoto] = useState('')
  const [endereco, setEndereco] = useState('')
  const [descricao, setDescricao] = useState('')
  //será chamada quando o form for submetido
  const submitForm = (e) => {
    //evita que o form seja submetido
    e.preventDefault()
  }
  //para limpar os campos
  const limparCampos = (e) => {
    //evita que o form seja submetido
    e.preventDefault()
    setTitulo('')
    setFoto('')
    setEndereco('')
    setDescricao('')
  }
  //vamos voltar para a página anterior aqui
  const cancelar = (e) => {
    //evita que o form seja submetido
    e.preventDefault()
  }
  return (//continua no próximo bloco
```

Bloco de Código 2.11.2

```
return (//continuação do bloco anterior
  <form onSubmit={handleSubmit}>
    <div className="formgrid grid">
      <div className="field col-12 lg:col-6">
        <label htmlFor="titulo">Título</label>
        <input type="text" value={titulo}
          onChange={(e) => setTitulo(e.target.value)}
          id="titulo" className="inputfield w-full p-2" />
      </div>
      <div className="field col-12 lg:col-6">
        <label htmlFor="foto">URL da foto</label>
        <input type="text" value={foto}
          onChange={(e) => setFoto(e.target.value)} id="foto"
          className="inputfield w-full p-2" />
      </div>
      <div className="field col-12">
        <label htmlFor="endereco">Endereço</label>
        <input type="text" value={endereco}
          onChange={(e) => setEndereco(e.target.value)}
          id="endereco" className="inputfield w-full p-2" />
      </div>
      <div className="field col-12">
        <label htmlFor="descricao">Descrição</label>
        <textarea id="descricao" value={descricao}
          onChange={(e) => setDescricao(e.target.value)}
          className="inputfield w-full p-3" />
      </div>
      <div className="col-12 flex justify-content-end">
        <span className="p-buttonset">
          <Button label="Salvar" icon="pi pi-check" />
          <Button label="Limpar" onClick={limparCampos} icon="pi
pi-trash" />
          <Button label="Cancelar" onClick={cancelar} icon="pi pi-
times" />
        </span>
      </div>
    </div>
  </form>
)
}

export default LugarForm
```


Ele pode ser testado como mostra o Bloco de Código 2.11.3.

Bloco de Código 2.11.3

```
import Layout from '../components/layout/Layout'
// import LugarLista from '../components/lugar/LugarLista'
import LugarForm from '../components/lugar/LugarForm'
return(
  <Layout>
    { /* <LugarLista lugares={lugares}/> */ }
    <LugarForm />
  </Layout>
)
}
export default HomePage
```

O resultado deve ser parecido com aquele que a Figura 2.11.1 mostra.

Figura 2.11.1

Depois de realizar este teste, não deixe de ajustar o conteúdo do arquivo **index.js** para que a aplicação volte a exibir a lista de lugares. Veja o Bloco de Código 2.11.4.

Bloco de Código 2.11.4

```
import Layout from '../components/layout/Layout'
import LugarLista from '../components/lugar/LugarLista'
// import LugarForm fom '../components/lugar/LugarForm'

const HomePage = () => {
  const lugares = [
    {
      id: '1',
      titulo: 'Trilhos',
      foto: 'https://images.unsplash.com/photo-1556905200-279565513a2d?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1470&q=80',
      endereco: 'Vila J, Interior de SP',
      descricao: 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quaerat harum odio omnis est fugit, incidunt, magni corporis vero aperiam fuga facere ab delectus natus placeat dolor nam quos sint impedit!'
    },
    {
      id: '2',
      titulo: 'Praça H',
      foto: 'https://images.unsplash.com/photo-1549877452-9c387954fbc2?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1170&q=80',
      endereco: 'Praça H, Austrália',
      descricao: 'Lorem ipsum dolor sit amet consectetur adipisicing elit. Quaerat harum odio omnis est fugit, incidunt, magni corporis vero aperiam fuga facere ab delectus natus placeat dolor nam quos sint impedit!'
    }
  ]

  return(
    <Layout>
      <LugarLista lugares={lugares}/>
    </Layout>
  )
}

export default HomePage
```

O componente de adição de lugares espera receber, via props, uma função a ser chamada quando o form for submetido. Ela deve adicionar o objeto é construído em

função dos valores armazenados nos controles do form, quando o botão é clicado. Veja o Bloco de Código 2.11.5.

Bloco de Código 2.11.5

```
...
const LugarForm = () => {
  ...
  //será chamada quando o form for submetido
  const submitForm = (e) => {
    //evita que o form seja submetido
    e.preventDefault()
    //constrói um lugar
    const lugar = {
      titulo, foto, endereco, descricao
    }
    props.onAdicionarLugar(lugar)
  }
  ...
}
```

2.12 (Página para a adição de novos lugares) A exibição do form será feita em uma página própria para isso. Ela será uma página gerenciada pelo NextJS. Por isso, precisa ser um componente criado na pasta **pages**. Crie uma pasta chamada **novo-lugar** como subpasta de **pages** e nela crie um arquivo chamado **index.js**. Seu conteúdo inicial aparece no Bloco de Código 2.12.1. Apenas definimos o novo componente, importamos o componente que define o form e o utilizamos na nova expressão JSX.

Bloco de Código 2.12.1

```
import { NovoLugarForm } from '../components/lugar/LugarForm'
const NovoLugarPage = () => {
  return (
    <NovoLugarForm />
  )
}

export default NovoLugarPage
```

Visite **localhost:3000/novo-lugar** e verifique se o resultado obtido se parece com aquele que a Figura 2.12.1 exibe. Como estamos exibindo o form diretamente, é de se esperar que a barra de navegação não seja exibida neste momento.

Figura 2.12.1

| |
|----------------------|
| Título |
| <input type="text"/> |
| URL da foto |
| <input type="text"/> |
| Endereço |
| <input type="text"/> |
| Descrição |
| <input type="text"/> |

✓ Salvar 🗑 Limpar ✕ Cancelar

Ainda no arquivo **pages/novo-lugar/index.js**, defina a função que aparece no Bloco de Código 2.12.2. Ela deve ser passada via props para o componente que define o form.

Bloco de Código 2.12.2

```
import LugarForm from '../..components/lugar/LugarForm'
const NovoLugarPage = () => {
  const onAdicionarLugar = (lugar) => {
    console.log(lugar)
  }
  return (
    <LugarForm onAdicionarLugar={onAdicionarLugar}/>
  )
}


export default NovoLugarPage
```

No seu navegador, preencha os campos do form usando dados quaisquer, apenas para teste. A seguir, clique no botão "Salvar" do form. O resultado esperado, que pode ser visto no Chrome Dev Tools (CTRL+SHIFT+I), aparece na Figura 2.12.2.

Figura 2.12.2

| |
|-------------|
| Título |
| Teste |
| URL da foto |
| Teste |
| Endereço |
| Teste |
| Descrição |
| Teste |

✓ Salvar 🗑 Limpar ✕ Cancelar



2.13 (Onde usar o componente Layout? O arquivo _app.js aparece novamente)

Temos duas páginas e desejamos que o conteúdo de ambas seja exibido de acordo com o que define o componente Layout.js, afinal, ele inclui a barra de navegação e o controle de responsividade. Uma alternativa é aplicá-lo duas vezes: nos arquivos **pages/novo-lugar/index.js** e **pages/index.js**. Ele já está em uso no arquivo principal (**pages/index.js**), portanto, vamos passar a utilizá-lo no arquivo **pages/novo-lugar/index.js** também. Veja o Bloco de Código 2.13.1.

Bloco de Código 2.13.1

```
import Layout from '../components/layout/Layout'
import LugarForm from '../components/lugar/LugarForm'
const NovoLugarPage = () => {

  const onAdicionarLugar = (lugar) => {
    console.log(lugar)
  }
  return (
    <Layout>
      <LugarForm onAdicionarLugar={onAdicionarLugar}/>
    </Layout>
  )
}

export default NovoLugarPage
```

Evidentemente, aplicar o componente Layout repetidas vezes conforme novas páginas forem sendo adicionadas ao projeto caracteriza uma péssima prática de programação. O que desejamos é utilizá-lo uma única vez em algum ponto específico da aplicação de modo que seu efeito se propague para todas as páginas de uma única vez. Isso pode ser feito no arquivo **_app.js**. Observe que ele define um componente React comum. Como qualquer outro, ele recebe um objeto props. A notação utilizada desestrutura o objeto extraíndo duas propriedades: um componente e um objeto props. O componente é sempre aquele a ser renderizado conforme a navegação acontece. O objeto props recebido é aquele a ser aplicado ao componente a ser renderizado. Veja o Bloco de Código 2.13.2.

Bloco de Código 2.13.2

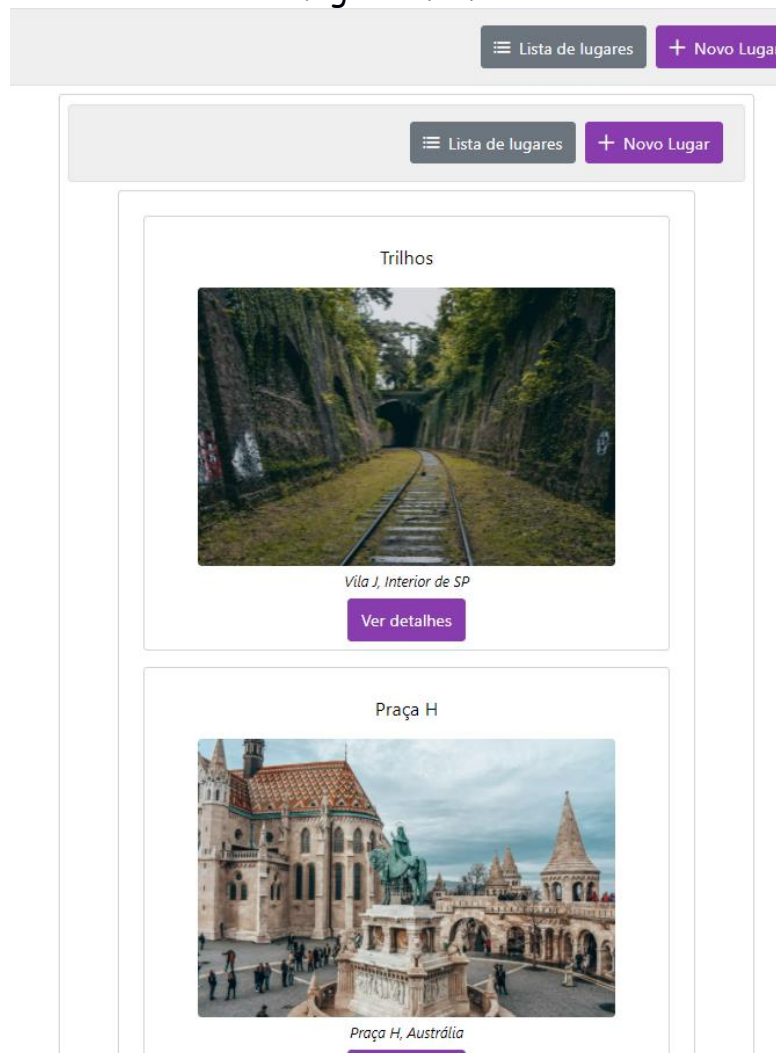
```
import 'primereact/resources/primereact.min.css'
import 'primeicons/primeicons.css'
import 'primeflex/primeflex.css'
import 'primereact/resources/themes/bootstrap4-light-purple/theme.css'
import '../styles/globals.css'
import Layout from '../components/layout/Layout'

function MyApp({ Component, pageProps }) {
  return <Layout>
    <Component {...pageProps} />
  </Layout>
}

export default MyApp
```

A partir de agora, toda página exibida sofrerá os efeitos definidos pelo componente Layout. Por isso, se você visitar qualquer uma das páginas neste momento, verá duas barras de navegação, já que o componente Layout está sendo utilizado duas vezes. Visite, por exemplo, a página principal em **localhost:3000** e obtenha um resultado parecido com aquele que a Figura 2.13.2 exibe.

Figura 2.13.2



Resolver este problema é muito simples: basta remover o componente Layout dos arquivos `pages/index.js` e `pages/novo-lugar/index.js` como mostram, respectivamente, os blocos de código 2.13.3 e 2.13.4.

Bloco de Código 2.13.3

```
...  
const HomePage = () => {  
...  
  return(  
    <LugarLista lugares={lugares}/>  
  )  
}  
...
```

Bloco de Código 2.13.4

```
import LugarForm from '../..components/lugar/LugarForm'  
const NovoLugarPage = () => {  
  
  const onAdicionarLugar = (lugar) => {  
    console.log(lugar)  
  }  
  return (  
    <LugarForm onAdicionarLugar={onAdicionarLugar}/>  
  )  
}  
  
export default NovoLugarPage
```

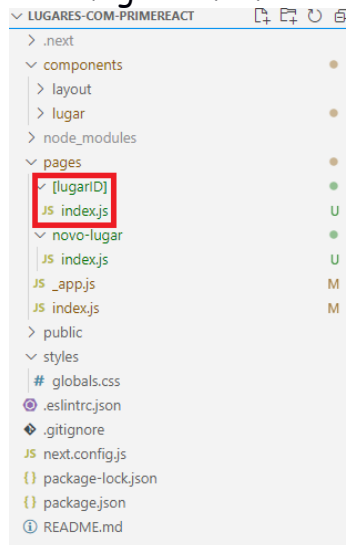
Cada página visitada deve agora exibir uma única barra de navegação e seu conteúdo deverá estar sob o controle do grid da PrimeReact definido no componente Layout.

2.14 (Uma página dinâmica para exibir os detalhes dos lugares) Temos potencialmente diversos botões "Ver detalhes", já que cada lugar define o seu. Quando um desses for clicado, evidentemente, desejamos exibir os detalhes do lugar a que se refere. É natural que a página de exibição de detalhes tenha o mesmo aspecto para todos os lugares. O que se espera mudar é apenas o conteúdo. Por isso, vamos definir uma página dinâmica que será acionada mediante um clique em um dos botões mencionados. Lembre-se que o nome de uma página dinâmica é definido assim.

[nome].js

Os símbolos [] é que a caracterizam como uma página dinâmica. Ocorre que esta notação se estende aos nomes de pastas. Neste exemplo, vamos criar uma pasta chamada **[lugarID]**, subpasta de **pages**. Dentro dela, criaremos um arquivo chamado **index.js**. Veja a Figura 2.14.1.

Figura 2.14.1



Para navegar até ela, podemos usar dois mecanismos.

- um componente Link do NextJS
- o roteador do NextJS.

Para exemplificar o seu uso, vamos adotar o roteador neste exemplo. Veja o Bloco de Código 2.14.1. Estamos no arquivo **LugarItem.js**.

Bloco de Código 2.14.1

```
import { useRouter } from 'next/router'
import { Button } from 'primereact/button'
const LugarItem = ({titulo, foto, endereco, id}) => {
  const router = useRouter()
  const exibirDetalhes = () => {
    //empilha uma nova página na pilha de navegação
    //observe que, seja lá qual for o id,
    //a página definida em [lugarID]/index.js
    router.push(`/${id}`)
  }
  return (
    <div className="align-items-center flex-column flex m-3 border border-
round border-1 border-400 p-2">
      <p className="text-center sm:text-base md:text-lg lg:text-
3xl">{titulo}</p>
      <div>
        <img className="border border-round" src={foto} alt={titulo}
width={400}/>
      </div>
      <address className="text-center text-xs md:text-sm lg:text-base mb-
2">{endereco}</address>
      <Button
        label="Ver detalhes"
        onClick={exibirDetalhes}
      />
    </div>
  )
}

export default LugarItem
```

Clicar em algum botão "Ver detalhes" neste momento causará um erro, já que o arquivo `[lugarID]/index.js` não define/exporta componente algum. Façamos a sua definição. Digamos que ele exibirá a foto e a descrição do lugar recebido via props. Adotaremos a seguinte abordagem. O componente de exibição de detalhes de um lugar será criado na pasta `components/lugar`, num arquivo chamado `LugarDetalhes.js`. Depois, criaremos uma página para exibi-lo. Crie, portanto, o arquivo `LugarDetalhes.js`. Veja seu conteúdo no Bloco de Código 2.14.2.

Bloco de Código 2.14.2

```
const LugarDetalhes = ({foto, titulo, descricao}) => {
  return(
    <div className="align-items-center flex-column flex m-3 border border-round border-1
border-400 p-2">
      <div>
        <img className="border border-round" src={foto} alt={titulo}
width={400}/>
      </div>
      <p className="text-center sm:text-base md:text-lg lg:text-
3xl">{descricao}</p>
    </div>
  )
}

export default LugarDetalhes
```

No arquivo `[lugarID]/index.js`, crie o componente que o Bloco de Código 2.14.3 mostra.

Bloco de Código 2.14.3

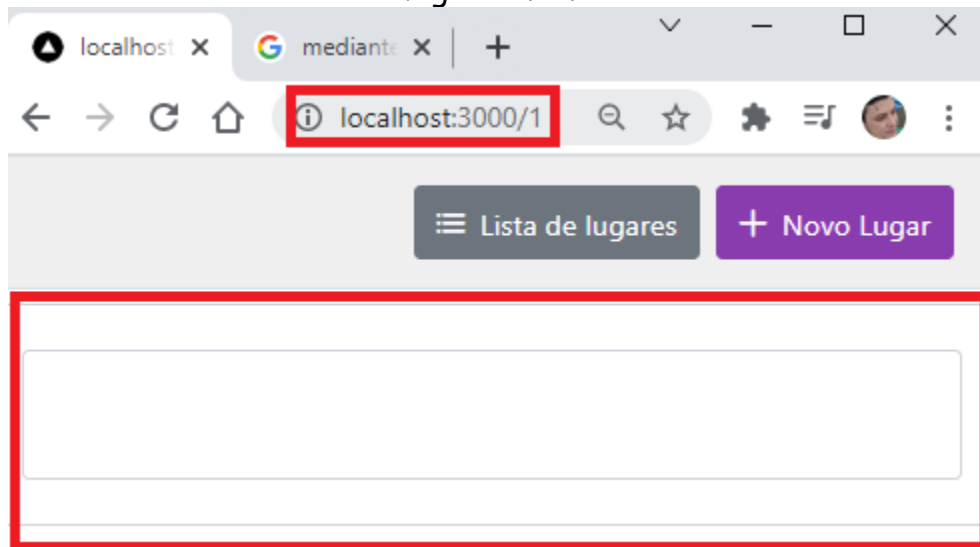
```
import LugarDetalhes from "../../components/lugar/LugarDetalhes"

const LugarDetalhesPage = () => {
  return (
    <LugarDetalhes/>
  )
}

export default LugarDetalhesPage
```

No seu navegador, visite localhost:3000. Clique no primeiro botão “Ver detalhes”. Você deverá ser levado para o endereço localhost:3000/1. A página deverá exibir apenas as caixinhas vazias, já que o componente ainda não recebe coisa alguma para exibir. Isso será ajustado em seções a seguir. A Figura 2.14.2 mostra o resultado esperado no momento.

Figura 2.14.2



2.15 (E se os dados forem obtidos remotamente?) A aplicação está exibindo uma lista de lugares que está definida estaticamente no arquivo **pages/index.js**. Ele renderiza um LugarLista entregando-lhe a lista de lugares via props. Entretanto, é natural que a coleção de lugares, em algum momento, passe a seja obtida de uma base remota. Para fazê-lo, uma possibilidade é a seguinte.

- Usamos o hook `useEffect` para executar código **depois** de o componente ter sido renderizado pela primeira vez.
- Utilizamos alguma biblioteca para requisições HTTP - talvez a `axios` - e fazemos a requisição, possivelmente do tipo **GET**.
- A função nos devolve uma `promise` e registramos uma função que executa quando o resultado estiver pronto. Ou seja, de maneira assíncrona.
- Armazenamos o resultado numa variável de estado que passamos ao componente LugarLista, via props.

Para entender o tópico tratado nesta seção, façamos uma simulação, tal como descrito. A menos de um detalhe: vamos utilizar uma `promise` que termina com sucesso imediatamente e nos permite obter a lista estática (é um teste ainda). Veja o Bloco de Código 2.15.1. Estamos no arquivo **pages/index.js**.

Bloco de Código 2.15.1

```
import LugarLista from '../components/lugar/LugarLista'
import { useState, useEffect } from 'react'

const HomePage = () => {
  //uma variável de estado
  const [lugaresObtidosExternamente, setLugaresObtidosExternamente] =
    useState([])
  //essa definição estática permanece, pois estamos apenas simulando a requisição
  remota por enquanto
  const lugares = [
    //mantenha essa coleção inalterada, com todos os seus objetos JSON
  ]

  useEffect(() => {
    //A promise termina imediatamente, dando acesso à coleção obtida
    //Ela é colocada na variável de estado, que é passada, via props,
    ao componente LugarLista
    Promise.resolve(lugares)
      .then(res => setLugaresObtidosExternamente(res))
  }, [])

  return(
    //LugarLista recebe a lista obtida "remotamente"
    <LugarLista lugares={lugaresObtidosExternamente}/>
  )
}

export default HomePage
```

Acesse **localhost:3000** no seu navegador e perceba que o resultado é o mesmo. Isso acontece, pois embora estejamos obtendo a coleção de maneira assíncrona, ela está definida localmente e, portanto, a busca pelos dados não leva tempo extra nenhum. Entretanto, neste cenário, o mais importante é perceber que **a busca pelo dados acontece apenas depois de o componente ser renderizado pela primeira vez**. Com a coleção em mãos, o React se encarrega de produzir os elementos da árvore necessários para a exibição adequada. Isso quer dizer que:

- O usuário pode ter de esperar até que os dados estejam disponíveis. Talvez esse seja ou não seja o comportamento que você espera para a sua aplicação.
- Mecanismos de buscas não têm acesso ao conteúdo completo da página. Para checar isso, abra localhost:3000 no navegador e aperte CTRL+U para visualizar o código fonte. Como mostra a Figura 2.15.1, toda a parte estática do site está disponível. Os dados não estão.

Nota. Você pode copiar o código do Chrome e colar num arquivo separado do VS Code. A seguir, utilize CTRL+SHIFT+F para indentar. O atalho pode ser diferente dependendo de seu sistema operacional. Verifique a documentação.

Figura 2.15.1

```
<body>
  <div id="__next">
    <div class="grid justify-content-center">
      <div class="col-12">
        <div class="p-toolbar p-component" role="toolbar">
          <div class="p-toolbar-group-left"></div>
          <div class="p-toolbar-group-right">
            <button
              color="warn"
              class="p-button p-component mr-2 p-button-secondary"
            >
              <span
                class="p-button-icon p-c pi pi-list p-button-icon-left"
              ></span>
              <span class="p-button-label p-c"
                >Lista de lugares</span>
            </button>
            <button class="p-button p-component">
              <span
                class="p-button-icon p-c pi pi-plus p-button-icon-left"
              ></span>
              <span class="p-button-label p-c">Novo Lugar</span>
            </button>
          </div>
        </div>
      </div>
    </div>
    <div
      class="
        col-12
        md:col-10
        lg:col-8
        border border-round border-1 border-400
      "
    ></div>
  </div>
</div>
```

A lista de lugares
seria filha dessa div



Nota. Se desejar, faça o ajuste destacado no Bloco de Código 2.15.2 e visualize o código fonte - com CTRL+U - uma vez mais. Você deverá ver o código HTML de todos os lugares. Depois do teste, desfaça esta atualização.

Bloco de Código 2.15.2

```
...
const HomePage = () => {
  ...
  return(
    //LugarLista recebe a lista obtida "remotamente"
    <LugarLista lugares={lugares}/>
  )
}
...
```

2.16 (Pré-renderização: geração estática com `getStaticProps`) Uma das principais funcionalidades oferecidas pelo NextJS é a pré-renderização de páginas. Faremos uso dela para resolver a questão discutida anteriormente de duas formas diferentes. A primeira se chama **geração estática**. A ideia da geração estática é a seguinte: **o conteúdo a ser exibido para o usuário é construído quando fazemos o "build" da aplicação**. Ou seja, antes de fazer a sua implantação. Quando ela é implantada, todo o seu conteúdo já está incluído e se torna desnecessário realizar buscas por ele quando a aplicação já estiver em funcionamento. Repare como tratamos das duas questões mencionadas anteriormente

- Já que o conteúdo está todo completo antes mesmo de a aplicação ser implantada, não há um tempo de espera por parte do usuário.
- Já que o conteúdo está todo completo antes mesmo de a aplicação ser implantada, os mecanismos de busca terão acesso a todo o seu código fonte.

Observe que, **caso a aplicação tenha dados atualizados quando já estiver em produção, será necessário eventualmente fazer um novo build e uma nova implantação**. Obviamente, a geração estática é útil para diferentes casos de uso e pode não ser apropriada para muitos outros.

A fim de utilizar a geração estática, fazemos a definição de um método específico do NextJS: **`getStaticProps`**. Ele somente tem efeito quando definido em componentes que definem páginas. Observe como seu nome é um tanto intuitivo. A sua missão é obter os dados necessários e entregá-los ao componente via props. Veja algumas de suas características.

- `getStaticProps` pode acessar o sistema de arquivos, uma base de dados, fazer requisições externas etc.
- o código de `getStaticProps` não faz parte do build final da aplicação, ou seja, ele não é entregue ao cliente.
- `getStaticProps` executa durante o processo de build da aplicação.

Nota. Muitas vezes utiliza-se a sigla **SSG**, de *Static Site Generation*, para se referir à funcionalidade em questão.

O Bloco de código 2.16.1 mostra a sua definição. Estamos no arquivo **pages/index.js**.

Bloco de Código 2.16.1

```
...
const HomePage = () => {
  ..
}

//defina essa função fora da função que define o componente
//o nome getStaticProps é obrigatório
export async function getStaticProps(){
  return {

  }
}
export default HomePage
```

O objeto devolvido por `getStaticProps` deve incluir uma chave chamada `props`. A ela associamos um objeto JSON que será entregue como `props` ao componente `HomePage`. Veja o Bloco de Código 2.16.2. Observe que, no momento, estamos utilizando a mesma coleção, já definida localmente. Mas lembre-se que essa poderia ser uma requisição externa que seria executada somente em tempo de “build”.

Bloco de Código 2.16.2

```
//defina essa função fora da função que define o componente
//o nome getStaticProps é obrigatório
export async function getStaticProps(){
  return {
    //esse objeto será entregue como props do componente HomePage
    props: {
      //usamos a coleção já definida localmente
      //nada muito útil por enquanto
      lugares: lugares
    }
  }
}
```

A seguir, podemos acessar a coleção lugares por meio do objeto props. Assim, não precisamos da variável de estado e da busca feita com o hook useEffect. Veja o Bloco de Código 2.16.3.

Bloco de Código 2.16.3

```
import LugarLista from '../components/lugar/LugarLista'
//apague os imports desses dois hooks abaixo
import { useState, useEffect } from 'react'

//defina a coleção de lugares fora da função que define o componente
const lugares = [
  //mantenha todos os itens da coleção
  //eles foram omitidos apenas para economizar espaço
]
//receba props
const HomePage = (props) => {
  //essa variável já não é necessária, apague
  const [lugaresObtidosExternamente, setLugaresObtidosExternamente] = useState([])
  //essa definição estática permanece, pois estamos apenas simulando a requisição remota
  por enquanto

  //apague esse useEffect inteiro
  useEffect(() => {
    Promise.resolve(lugares).then(res => setLugaresObtidosExternamente(res))
  }, [])

  return(
    //use a lista de lugares de props
    <LugarLista lugares={props.lugares}/>
  )
}

//defina essa função fora da função que define o componente
//o nome getStaticProps é obrigatório
export async function getStaticProps(){
  return {
    //esse objeto será entregue como props do componente HomePage
    props: {
      //usamos a coleção já definida localmente
      //nada muito útil por enquanto
      lugares: lugares
    }
  }
}
export default HomePage
```

No seu navegador, visite `localhost:3000` e use `CTRL+U` para visualizar o código fonte. Copie e cole o código fonte numa aba do VS Code e indente o código. Como a Figura 2.16.1 mostra, o código fonte inclui os elementos referentes a todos os lugares da lista.

Figura 2.16.1

```
</div>
<div
  class="
    col-12
    md:col-10
    lg:col-8
    border border-round border-1 border-400
  "
>
  <div
    class="
      align-items-center
      flex-column flex
      m-3
      border border-round border-1 border-400
      p-2
    "
  >
    <p class="text-center sm:text-base md:text-lg lg:text-xl">
      Trilhos
    </p>
    <div>
      
    </div>
    <address class="text-center text-xs md:text-sm lg:text-base mb-2">
      Vila J, Interior de SP
    </address>
    <button class="p-button p-component">
      <span class="p-button-label p-c">Ver detalhes</span>
    </button>
  </div>
</div>
```

esse é o primeiro, o outro aparece logo abaixo

Apenas por curiosidade, execute o seguinte comando num terminal interno do VS Code.

npm run build

A Figura 2.16.2 mostra o resultado esperado. Observe que ele mostra as páginas que foram geradas e cita as técnicas de renderização utilizadas. Uma delas é a que acabamos de usar.

Figura 2.16.2

```
PS C:\Users\rodri\Documents\workspaces\pessoal\nextjs\lugares-com-primereact
> npm run build

> lugares-com-primereact@0.1.0 build C:\Users\rodri\Documents\workspaces\pes
soal\nextjs\lugares-com-primereact
> next build

info - Using webpack 5. Reason: Enabled by default https://nextjs.org/docs/messages/webpack!
info - Checking validity of types
error - ESLint must be installed in order to run during builds: yarn add --dev eslint
info - Creating an optimized production build
info - Compiled successfully
info - Collecting page data
info - Generating static pages (5/5)
info - Finalizing page optimization
```

| Page | Size | First Load JS |
|---------------|-------|---------------|
| • / | 604 B | 83.5 kB |
| - /_app | 0 B | 82.9 kB |
| o /[lugarID] | 431 B | 83.3 kB |
| o /404 | 194 B | 83.1 kB |
| o /novo-lugar | 791 B | 83.7 kB |

```
+ First Load JS shared by all 82.9 kB
  | chunks/framework.2191d1.js 42.4 kB
  | chunks/main.62b8ca.js 23.3 kB
  | chunks/pages/_app.a4ad8e.js 16.5 kB
  | chunks/webpack.1a8a25.js 729 B
  | css/fafe99dd1f4f61c5c9ee.css 71.3 kB

λ (Server) server-side renders at runtime (uses getInitialProps or getServerSideProps)
o (Static) automatically rendered as static HTML (uses no initial props)
• (SSG) automatically generated as static HTML + JSON (uses getStaticProps)
  (ISR) incremental static regeneration (uses revalidate in getStaticProps)
```

```
PS C:\Users\rodri\Documents\workspaces\pessoal\nextjs\lugares-com-primereact>
```

2.17 (Geração estática e alguns “problemas”. Usando a Regeneração Estática Incremental) Uma implicação imediata do uso da geração estática é a possibilidade de nossa aplicação estar utilizando **conteúdo desatualizado**. Para alguns sites, esse pode, na verdade, não ser um problema, pois talvez seu conteúdo não seja alterado com muita frequência. Para aqueles cujos conteúdos são alterados com frequência, o NextJS oferece um mecanismo conhecido como **ISR** - de **Incremental Static Generation**. Em português seu nome fica parecido com **Regeneração Estática Incremental**. Seu uso é bastante simples: adicionamos uma propriedade chamada **revalidate** ao objeto produzido pelo método `getStaticProps`. Ela fica associada a um número. Ele é a quantidade de segundos que o NextJS espera para gerar a página **acessada** novamente.

Repare bem na palavra "acessada". Se o valor de `revalidate` for `n`, isso quer dizer que, após uma primeira requisição, todas as requisições feitas antes de se passarem `n` segundos serão atendidas com a página atual, já gerada. Uma próxima requisição faz com que ela seja gerada novamente. No Bloco de Código 2.17.1 ilustramos o uso da propriedade `revalidate`.

Bloco de Código 2.17.1

```
...
const HomePage = (props) => {
...
}
export async function getStaticProps(){
  return {
    props: {
      lugares: lugares
    },
    revalidate: 10
  }
}
...
```

Nota. Com o ISR, as páginas permanecem sendo geradas em tempo de "build". Além disso, são geradas novamente como descrito, quando a aplicação já está em funcionamento.

Nota. Vale visitar o Link 2.17.1 para entender mais sobre ISR.

Link 2.17.1

<https://vercel.com/docs/concepts/next.js/incremental-static-regeneration>

2.18 (Pré-renderização: o método `getServerSideProps`) Como mencionamos, o funcionamento do geração estática pode não ser apropriado para todos os cenários. Em alguns casos, pode ser necessário renderizar a página novamente a cada requisição. Para estes casos, podemos utilizar o método **`getServerSideProps`**. A ideia é semelhante. O método executa do lado do servidor e entrega os dados obtidos ao componentes que representa a página, via props. A renderização do componente se dá ali, ainda do lado do servidor. O conteúdo entregue ao navegador do usuário é a página inteira pronta, como todo o seu código-fonte.

O Bloco de Código 2.18.1 mostra como o método `getServerSideProps` pode ser definido. Ainda estamos no arquivo 2.18.1.

Nota. Observe que não podemos utilizar os métodos `getStaticProps` e `getServerSideProps` simultaneamente.

Bloco de Código 2.18.1

```
...
const HomePage = (props) => {
  ...
}
export async function getServerSideProps(){
  return {

  }
}
//comentamos esse
// export async function getStaticProps(){
//   return {
//     props: {
//       lugares: lugares
//     },
//     revalidate: 10
//   }
// }
```

O método `getServerSideProps` também devolve um JSON que possui uma chave chamada `props`. Além disso, ele recebe um objeto "context" que, entre outras coisas, dá acesso aos objetos que representam a requisição e a resposta. Veja o Bloco de Código 2.18.2.

Bloco de Código 2.18.2

```
export async function getServerSideProps(context){  
  //se precisar, você pode acessar a requisição e a resposta assim  
  const {req, res} = context  
  return {  
    props: {  
      lugares: lugares  
    }  
  }  
}
```

Para os próximos passos, continuaremos utilizando a função **getStaticProps**. Por isso, você pode **remover a função getServerSideProps** e tirar os comentários da **getStaticProps**. Neste momento, o conteúdo do seu arquivo **pages/index.js** deve ser aquele exibido pelo Bloco de Código 2.18.3.

Bloco de Código 2.18.3

```
import LugarLista from '../components/lugar/LugarLista'  
  
const lugares = [  
  //mantenha os itens  
]  
  
const HomePage = (props) => {  
  return(  
    <LugarLista lugares={props.lugares}/>  
  )  
}  
  
export async function getStaticProps(){  
  return {  
    props: {  
      lugares: lugares  
    },  
    revalidate: 10  
  }  
}  
  
export default HomePage
```

2.19 (Todas as páginas usam `getStaticProps` ou `getServerSideProps`?) A renderização como descrita não necessariamente se aplica a todas as páginas. De fato, aquelas que não possuem dado algum não precisam desse mecanismo. Por isso, vamos avaliar cada página de nosso projeto para decidir quais precisam. Lembre-se que esse mecanismo se aplica somente a componentes que definem páginas, ou seja, aqueles cujas definições aparecem na pasta `pages` ou alguma subpasta dela.

- **`pages/novo-lugar/index.js`** - A página definida neste arquivo apenas mostra um form para o usuário cadastrar novos lugares. Ela não tem dados que serão buscados em momento algum. Portanto, dispensa o mecanismo.

- **`pages/[lugarID]/index.js`** - A razão de ser da página definida neste arquivo, é exibir os dados de um lugar. Como há dados envolvidos, ela fará uso do mecanismo. É razoável que essa página não seja atualizada com muita frequência. Por isso, vamos usar o método **`getStaticProps`**. Veja o Bloco de Código 2.19.1. Estamos no arquivo **`pages/[lugarID]/index.js`**.

Nota. Observe que a definição pode ser feita utilizando-se arrow functions também.

Bloco de Código 2.19.1

```
import LugarDetalhes from "../../components/lugar/LugarDetalhes"

const LugarDetalhesPage = (props) => {
  return (
    <LugarDetalhes
      foto={props.dadosLugar.foto}
      titulo={props.dadosLugar.titulo}
      descricao={props.dadosLugar.descricao}
    />
  )
}

export const getStaticProps = async () => {
  //buscar os dados do lugar a ser exibido de alguma forma
  //uma requisição http talvez
  return {
    props: {
      dadosLugar :{
        //vamos deixar fixo por enquanto, para dar um exemplo
        foto: 'https://images.unsplash.com/photo-1549877452-9c387954fbc2?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1170&q=80',
        titulo: 'Praça H',
        descricao: 'Lorem ipsum, dolor sit amet consectetur adipiscing elit. Asperiores, officiis.'
      }
    }
  }
}

export default LugarDetalhesPage
```

Faça o seguinte teste. No seu navegador, abra localhost:3000 e clique em um dos botões "Ver detalhes". Pode ser qualquer um. Você verá um erro - no terminal interno do VS Code - parecido com aquele ilustrado na Figura 2.19.1.

Figura 2.19.1

```
error - Error: getStaticPaths is required for dynamic SSG pages and is missing for '/[lugarID]'.
Read more: https://nextjs.org/docs/messages/invalid-getstaticpaths-value
    at Object.renderToHTML (C:\Users\rodri\Documents\workspaces\pessoal\nextjs\lugares-com-primereact\node_modules\next\dist\server\render.js:214:15)
    at doRender (C:\Users\rodri\Documents\workspaces\pessoal\nextjs\lugares-com-primereact\node_modules\next\dist\server\next-server.js:1144:57)
    at C:\Users\rodri\Documents\workspaces\pessoal\nextjs\lugares-com-primereact\node_modules\next\dist\server\next-server.js:1236:34
    at C:\Users\rodri\Documents\workspaces\pessoal\nextjs\lugares-com-primereact\node_modules\next\dist\server\response-cache.js:64:42 {
  page: '/[lugarID]'
}
```

O problema é que estamos acessando uma página dinâmica. Como estamos pedindo a sua renderização prévia, o NextJS não tem como saber quais páginas gerar, já que uma página dinâmica é apenas um modelo que pode ser utilizado para a exibição de um lugar. Qualquer um que estiver na lista. Utilizamos a função **getStaticPaths** para resolver esse problema. Ela devolve um objeto JSON que tem uma lista associada a uma chave chamada **paths**. Cada item na lista é um objeto JSON que descreve as características desejadas para cada página. Veja o Bloco de Código 2.19.2.

Bloco de Código 2.19.2

```
export const getStaticPaths = async () => {
  return {
    paths: [
      {
        //este JSON define as propriedades de interesse
        //para uma página
      }
    ]
  }
}
```

Cada página tem uma propriedade chamada **params**. Ela fica associada a um objeto JSON. No momento, vamos "chumbar" alguns valores de ids para duas páginas. No futuro, esses valores serão recuperados de uma base de dados. Ou seja, em tempo de "build", o NextJS acessará uma base de dados para encontrar os ids naquele momento existentes e gerar uma página para cada item associado. Veja o Bloco de Código 2.19.3.

Bloco de Código 2.19.3

```
export const getStaticPaths = async () => {
  return {
    paths: [
      {
        params: {
          //tem que ser lugarID pois a pasta
          //se chama [lugarID], lembra?
          lugarID: '1'
        }
      },
      {
        params: {
          lugarID: '2'
        }
      }
    ]
  }
}
```

Observe que o erro obtido agora é outro. Veja a Figura 2.19.2.

Figura 2.19.2

```
error - Error: The `fallback` key must be returned from getStaticPaths in /[
lugarID].
Expected: { paths: [], fallback: boolean }
See here for more info: https://nextjs.org/docs/messages/invalid-getstaticpa
ths-value
    at Object.buildStaticPaths (C:\Users\rodri\Documents\workspaces\pessoal\
nextjs\lugares-com-primereact\node_modules\next\dist\build\utils.js:486:15)
    at processTicksAndRejections (internal/process/task_queues.js:95:5) {
  type: 'Error',
  page: '/[lugarID]'
}
error - Error: The `fallback` key must be returned from getStaticPaths in /[
lugarID].
Expected: { paths: [], fallback: boolean }
```

O erro diz que a chave "fallback" deve fazer parte do objeto devolvido pelo método `getStaticPaths`. Esta chave desempenha o seguinte papel: ela informa ao NextJS se todos os possíveis valores já estão inclusos no vetor `paths` ou se apenas alguns deles, da seguinte forma.

- Se `fallback` estiver valendo **false**, estamos dizendo ao NextJS que todos os possíveis valores já estão inclusos no vetor `paths`. Se o usuário tentar acessar alguma página não prevista, ele verá uma página com o erro 404.
- Se `fallback` estiver valendo **true**, estamos dizendo ao NextJS que nem todos os possíveis valores já estão inclusos no vetor `paths`. O NextJS tentará gerar uma página para o id envolvido na requisição. Se ele existir, a página será entregue para o cliente. Caso contrário, o cliente verá um erro 404.

Veja o Bloco de Código 2.19.4. Permanecemos no arquivo `pages/[lugarID]/index.js`.

Bloco de Código 2.19.4

```
...
const LugarDetalhesPage = (props) => {
  ...
}

export const getStaticPaths = async () => {
  return {
    fallback: false,
    paths: [
      {
        params: {
          //tem que ser lugarID pois a pasta
          //se chama [lugarID], lembra?
          lugarID: '1'
        }
      },
      {
        params: {
          lugarID: '2'
        }
      }
    ]
  }
}

export const getStaticProps = async () => {
  ...
}

export default LugarDetalhesPage
```


Faça os seguintes testes.

I. No seu navegador, visite localhost:3000. Clique em cada um dos botões "Ver detalhes". Você deverá ser levado a uma página que mostra o mesmo lugar, independente do botão, afinal, os dados ainda estão "chumbados".

II. Digite localhost:3000/3 na barra do navegador. Você deverá ver um erro 404.

Referências

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

PrimeReact | React UI Component Library. 2021. Disponível em <<https://www.primefaces.org/primereact/>>. Acesso em outubro de 2021.