React Router v6

Comece criando um novo aplicativo React. Use o seguinte comando em uma janela de terminal para gerar o diretório do projeto, navegue dentro do diretório do projeto e instale as dependências necessárias para adicionar a biblioteca React Router v6:

```
npx create-react-app react-router-v6-example
cd react-router-v6-example

npm install react-router-dom
```

Criando rotas com React Router v6

Para criar a primeira rota usando a biblioteca React Router, abra o arquivo src/App.js e adicione a seguinte instrução de importação:

App.js

```
//depois de outras instruções de importação
import { BrowserRouter as Router } from 'react-router-dom';
```

Este é o primeiro componente a ser importado da biblioteca react-router-dom. É usado para envolver diferentes rotas. Ele usa a API de histórico HTML5 para acompanhar o histórico de rotas no aplicativo React. A parte Router no trecho acima é o alias que facilita a escrita. É recomendado importá-lo e usá-lo no componente de nível superior na hierarquia de componentes de um aplicativo React:

App.js

```
function App() {
  return <Router>{/* Todas as rotas estão aninhadas dentro
dele */}</Router>;
}
```

O próximo componente a ser importado do react-router-dom são as `Routes`:

App.js

```
import { BrowserRouter as Router, Routes } from
'react-router-dom';
```

Ele inclui recursos como roteamento e vinculação relativos, classificação automática de rotas, rotas aninhadas e layouts. O último componente necessário do react-router-dom é chamado `Route` e é responsável por renderizar a UI de um componente React:

App.js

```
import { BrowserRouter as Router, Routes, Route } from
'react-router-dom';
```

Possui um prop chamado `path` que sempre corresponde à URL atual do aplicativo. A segunda prop necessária é chamada de `element` que informa ao componente `Route` quando uma URL atual é encontrada e qual componente React deve ser renderizado.

Construindo componentes funcionais

Para criar a primeira rota na demonstração a seguir, vamos criar um componente funcional básico chamado `Home` que retorna algum JSX:

função Home no arquivo App.js

```
function Home() {
  return (
    <div style={{ padding: 20 }}>
      <h2>Home View</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
  );
}
```

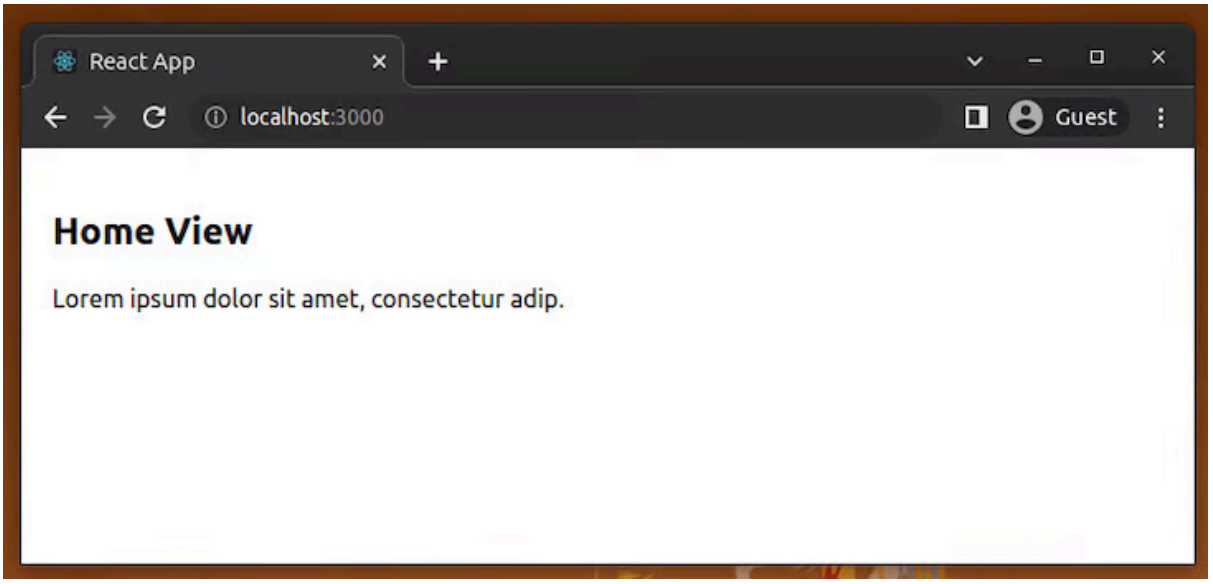
Em seguida, atualize o componente funcional `App` com a rota a seguir. O prop `element` de um componente `Route` agora permite que você passe um componente React em vez de apenas o nome desse componente React. Isso facilita a passagem de acessórios pelas rotas:

App.js

```
...

export default function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
      </Routes>
    </Router>
  );
}
```

Para vê-lo funcionando, volte para a janela do terminal e inicie o servidor de desenvolvimento usando o comando `npm start`. Em seguida, visite a URL `http://localhost:3000` em uma janela do navegador. Aqui está a saída após esta etapa:



Vamos criar rapidamente outro componente funcional chamado `About` que só é renderizado quando a URL em uma janela do navegador é `http://localhost:3000/about`:

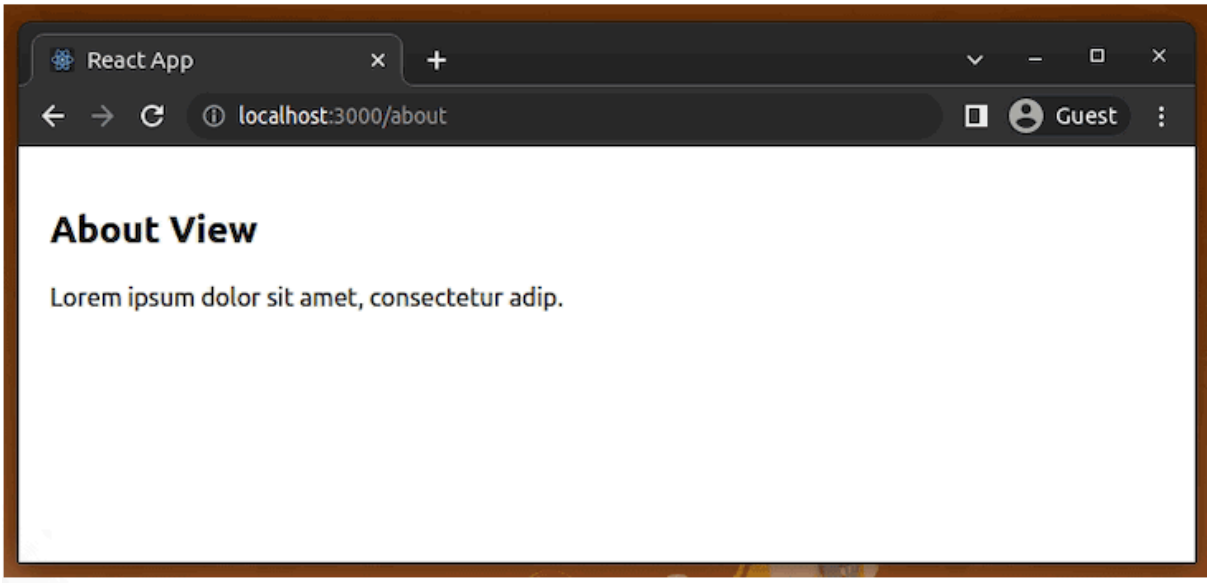
```
function About() {
  return (
```

```
    <div style={{ padding: 20 }}>
      <h2>About View</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
  );
}
```

Em seguida, adicione a `Route` para o componente `About`:

```
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </Router>
```

Agora, volte para a janela do navegador e navegue até a URL `http://localhost:3000/about`, conforme mostrado na visualização a seguir:



Conforme mostrado na visualização acima, você pode navegar até a página Sobre usando a rota `/about`. O botão avançar/voltar do navegador também funciona e altera as visualizações com base na pilha do histórico.

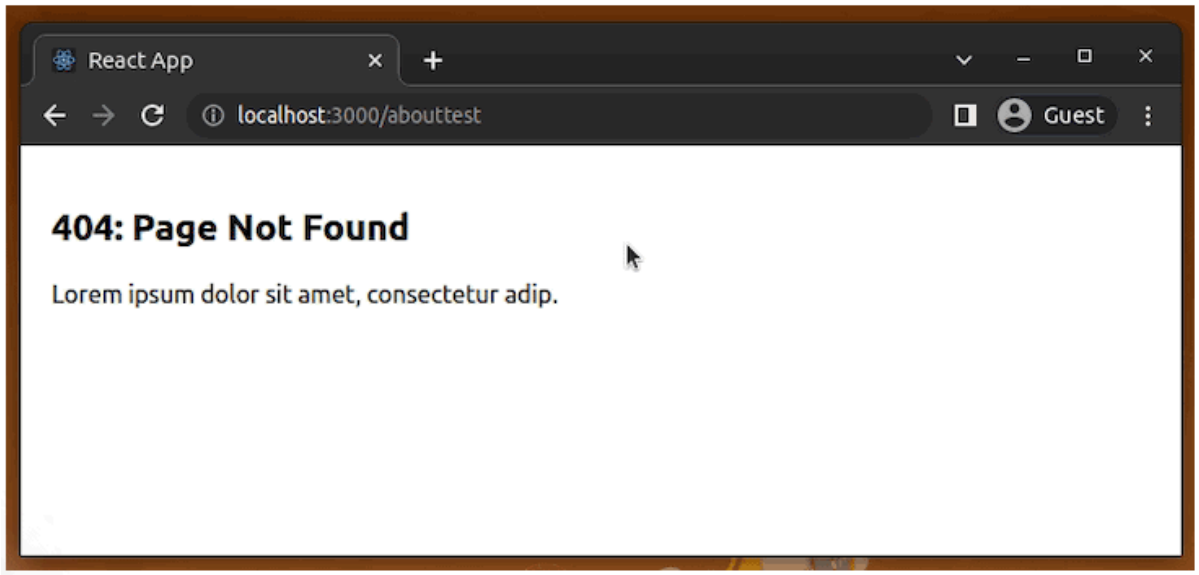
Criando visualização página não encontrada 404

Você pode implementar uma visualização 404 para entradas de rota inválidas adicionando uma `no-match` com a sintaxe `*` da seguinte forma:

```
function NoMatch() {
  return (
    <div style={{ padding: 20 }}>
      <h2>404: Page Not Found</h2>
      <p>Lorem ipsum dolor sit amet, consectetur adip.</p>
    </div>
  );
}

export default function App() {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="*" element={<NoMatch />} />
      </Routes>
    </Router>
  );
}
```

Depois de usar o segmento de código acima em sua fonte `App.js`, você verá uma página 404 ao inserir uma rota inválida no URL do navegador:



Adicionando um menu de navegação

Para navegar em uma rota específica no aplicativo React, ou nas duas rotas atualmente existentes no aplicativo de demonstração, vamos adicionar uma barra de navegação mínima com a ajuda do componente `Link` do react-router-dom. Comece importando `Link` da biblioteca:

```
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
```

O conceito de navegar entre diferentes páginas da web em HTML é usar uma tag âncora, conforme mostrado abaixo:

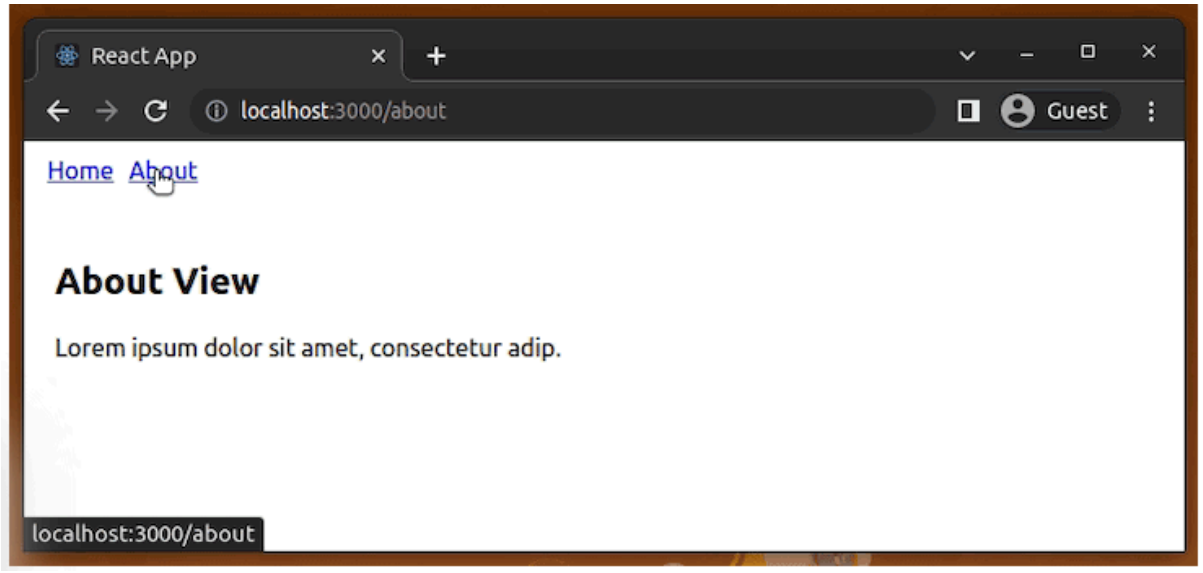
```
<a href="">Some Link Name</a>
```

Usar essa abordagem em um aplicativo React levará à atualização de uma página da web cada vez que uma nova visualização ou página for renderizada. Esta não é a vantagem que você procura ao usar uma biblioteca como React. Para evitar a atualização das páginas da web, a biblioteca react-router-dom fornece o componente `Link`. A seguir, dentro do componente de função `App`, crie uma barra de navegação conforme mostrado no trecho de código:

```
export default function App() {
  return (
```

```
    <Router>
      <nav style={{ margin: 10 }}>
        <Link to="/" style={{ padding: 5 }}>
          Home
        </Link>
        <Link to="/about" style={{ padding: 5 }}>
          About
        </Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="*" element={<NoMatch />} />
      </Routes>
    </Router>
  );
}
```

Vá para a janela do navegador para ver a barra de navegação em ação:



Como lidar com rotas aninhadas

O roteamento de aninhamento é um conceito importante a ser entendido. Quando as rotas são aninhadas, geralmente assume-se que uma determinada parte de uma página web permanece constante e apenas a parte filha da página web muda. Por exemplo, se você visitar um blog simples, o título do blog será sempre exibido, com uma lista de postagens exibida abaixo dele.

No entanto, quando você clica em uma postagem, a lista de postagens é substituída pelo conteúdo ou pela descrição dessa postagem específica. Este é um exemplo que será executado nesta seção para entender como lidar com rotas aninhadas na biblioteca React Router 6.

Para começar, importe o Outlet da biblioteca react-router-dom:

```
import { BrowserRouter as Router,
   Routes,
    Route,
    Link,
    Outlet } from "react-router-dom";
```

Para imitar um blog básico, vamos adicionar alguns dados simulados no arquivo App.js. O trecho de código consiste em um objeto chamado BlogPosts, que consiste ainda em diferentes objetos como propriedades. Cada objeto é constituído de três coisas:

- Uma postagem única
- Título dessa postagem
- Descrição dessa postagem

início do seu arquivo App.js (após todas as importações):

```
const BlogPosts = {
  "first-blog-post": {
    title: "First Blog Post",
    description: "Lorem ipsum dolor sit amet, consectetur
adip.",
  },
  "second-blog-post": {
    title: "Second Blog Post",
    description: "Hello React Router v6",
```

```
  },
};
```

Esta postagem exclusiva será usada na URL de um navegador da web para ver o conteúdo de cada postagem. A seguir, crie um componente funcional chamado `Posts`, onde uma lista de todas as postagens é exibida:

```
function Posts() {
  return (
    <div style={{ padding: 20 }}>
      <h2>Blog</h2>
      <Outlet />
    </div>
  );
}
```

A definição do componente acima de `Outlet` renderizará os componentes filhos com base nas definições de roteamento aninhadas. Defina outro componente chamado `PostLists` que exibirá uma lista de todas as postagens sempre que a URL na janela do navegador atingir `http://localhost:3000/posts`. Vamos usar o método JavaScript `Object.entries()` para retornar um array do objeto `BlogPosts`. Essa matriz é então mapeada para exibir uma lista de títulos de todas as postagens:

```
function PostLists() {
  return (
    <ul>
      {Object.entries(BlogPosts).map(([slug, { title }]) => (
        <li key={slug}>
          <h3>{title}</h3>
        </li>
      ))}
    </ul>
  );
}
```

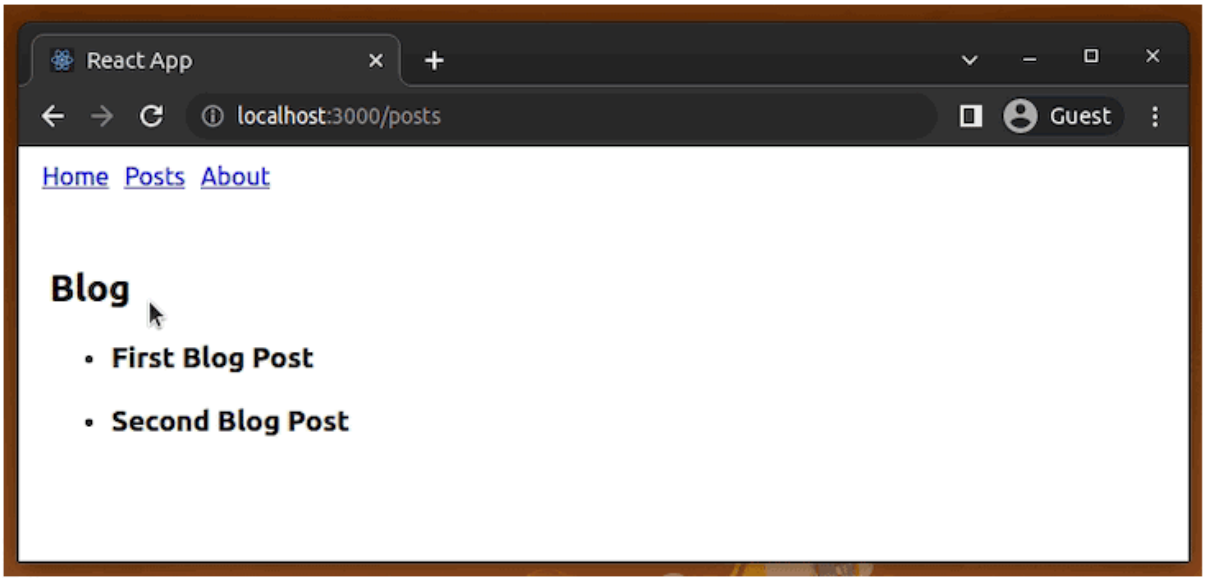
Modifique as rotas no componente de função `App` assim:

```
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/posts" element={<Posts />}>
          <Route index element={<PostLists />} />
        </Route>
        <Route path="/about" element={<About />} />
        <Route path="*" element={<NoMatch />} />
      </Routes>
```

Aqui usamos a propriedade `index` para a rota `PostLists` para especificar o índice de `/posts`. Isso indica que sempre que a URL `http://localhost:3000/posts` for acionada, uma lista de posts será renderizada, daí o componente `PostsLists`. A seguir, atualize a navegação adicionando um link para a página Postagens:

```
    <nav style={{ margin: 10 }}>
        <Link to="/" style={{ padding: 5 }}>
          Home
        </Link>
        <Link to="/posts" style={{ padding: 5 }}>
          Posts
        </Link>
        <Link to="/about" style={{ padding: 5 }}>
          About
        </Link>
      </nav>
```

Depois de fazer as atualizações acima, olhe a janela do seu navegador. Você verá a seguinte saída:



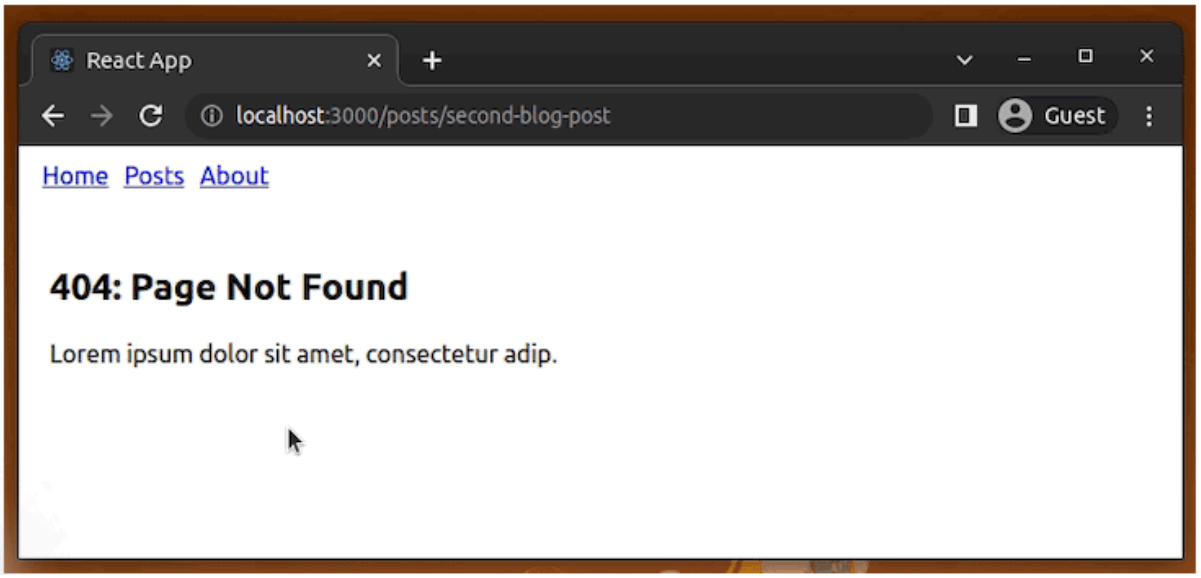
Observação: aqui renderizamos o componente filho BlogLists dentro do componente pai Blog por meio do componente Outlet integrado da biblioteca.

Acessando parâmetros de URL e parâmetros dinâmicos de uma rota

Para visitar a postagem individual clicando no título da postagem na lista renderizada de postagens, tudo o que você precisa fazer é agrupar o título de cada postagem em um componente `Link` no componente `PostsLists`. Em seguida, defina o caminho para cada post usando o `slug` de cada post. O prefixo `/posts/` permite que o caminho no navegador da web seja consistente:

```
function PostLists() {
  return (
    <ul>
      {Object.entries(BlogPosts).map(([slug, { title }]) => (
        <li key={slug}>
          <Link to={`/posts/${slug}`}>
            <h3>{title}</h3>
          </Link>
        </li>
      ))}
    </ul>
  );
}
```

Nesta fase, você também pode testar sua página 404, já que não adicionamos uma página para uma única postagem. Você obterá sua página 404 sempre que clicar no título de uma postagem:



Vamos continuar o processo de desenvolvimento e exibir uma única postagem. Importe um Hook chamado `useParams` da biblioteca react-router-dom. Este hook permite acessar quaisquer parâmetros dinâmicos que uma determinada rota (ou slug, neste caso) possa ter. Os parâmetros dinâmicos para cada `slug` serão o `title` e a `description` de cada postagem do blog.

A necessidade de acessá-los é exibir o conteúdo de cada postagem do blog quando um determinado slug de uma postagem do blog é acionado como URL na janela do navegador:

```
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Link,
  Outlet,
  useParams
} from "react-router-dom";
```

Crie um novo componente funcional chamado `Post`. Este componente irá obter o slug atual da postagem do `useParams` Hook. Usando a sintaxe de notação quadrada de colchetes em JavaScript, é criada uma nova variável `post` que possui o valor das propriedades ou o conteúdo atual de uma postagem. Desestruturando o conteúdo desta variável `post`, você pode renderizá-los, assim:

```
function Post() {
  const { slug } = useParams();
  const post = BlogPosts[slug];
  if (!post) {
    return <span>The blog post you've requested doesn't
exist.</span>;
  }
  const { title, description } = post;
  return (
    <div style={{ padding: 20 }}>
      <h3>{title}</h3>
      <p>{description}</p>
    </div>
  );
}
```

Por último, adicione uma rota dinâmica chamada `:slug` no componente de função `App` para renderizar o conteúdo de cada postagem:

```
        <Route path="/posts" element={<Posts />}>
          <Route index element={<PostLists />} />
          <Route path=":slug" element={<Post />} />
        </Route>
```

Aqui está a saída completa após esta etapa:

Apêndice

Código no Javascript - PostLists

```
const BlogPosts = {
  "first-blog-post": {
    title: "First Blog Post",
    description: "Lorem ipsum dolor sit amet, consectetur
adip.",
  },
  "second-blog-post": {
    title: "Second Blog Post",
    description: "Hello React Router v6",
  },
};

//console.log(Object.entries(BlogPosts).map(([slug, {title}])
=> (slug)));

console.log(Object.entries(BlogPosts));
```