21 DE DEZEMBRO DE 2022 / #REACT

Tutorial de aplicação com CRUD em React – como criar uma aplicação de gerenciamento de livros em React do zero

Tradutor: Elizabete Nakamura

Autor: Yogesh Chavan (em inglês)

Neste artigo, você construirá uma aplicação de gerenciamento de livros em React do zero.

Ao criar essa aplicação, você aprenderá:

1. Como realizar operações de CRUD
2. Como usar o React Router para navegação entre rotas
3. Como usar a API React Context para passar dados através de rotas
4. Como criar um hook personalizado em React
5. Como armazenar dados em armazenamento local para persistir mesmo após a atualização da página
6. Como gerenciar os dados armazenados no armazenamento local usando um hook personalizado

e muito mais.

Usaremos hooks do React para construir esta aplicação. Portanto, se você é novo no React Hooks, confira meu artigo Introdução ao React Hooks (texto em inglês) para aprender o básico.

> *Quer aprender Redux desde o início absoluto e construir uma aplicação de pedido de comida a partir do zero? Confira o curso Mastering Redux.*

Configuração inicial

Crie um projeto utilizando o `create-react-app`:

Uma vez criado o projeto, exclua todos os arquivos da pasta `src` e crie os arquivos `index.js` e `styles.scss` dentro da pasta `src`. Além disso, crie as pastas `components`, `context`, `hooks` e `router` dentro da pasta `src`.

Instale as dependências necessárias:

```
yarn add bootstrap@4.6.0 lodash@4.17.21 react-bootstrap@1.5.2 node-sass
```

Abra `styles.scss` e adicione este conteúdo a ela.

Como criar as páginas iniciais

Crie um novo arquivo `Header.js` dentro da pasta `components` com o seguinte conteúdo:

```
import React from 'react';
import { NavLink } from 'react-router-dom';

const Header = () => {
  return (
    <header>
      <h1>Book Management App</h1>
      <hr />
      <div className="links">
        <NavLink to="/" className="link" activeClassName="active" exact
          Books List
        </NavLink>
        <NavLink to="/add" className="link" activeClassName="active">
          Add Book
        </NavLink>
```

```
export default Header;
```

Aqui, adicionamos dois links de navegação usando o componente `NavLink` do `react-router-dom` : um para ver uma lista de todos os livros e o outro para adicionar um novo livro.

Estamos usando o `NavLink` em vez da tag de âncora (`<a />`). Portanto, a página não será atualizada quando um usuário clicar em qualquer um dos links.

Crie um arquivo chamado `BooksList.js` dentro da pasta `components` com o seguinte conteúdo:

```
import React from 'react';

const BooksList = () => {
  return <h2>List of books</h2>;
};

export default BooksList;
```

Crie um arquivo chamado `AddBook.js` dentro da pasta `components` com o seguinte conteúdo:

```
import React from 'react';
import BookForm from './BookForm';

const AddBook = () => {
  const handleOnSubmit = (book) => {
    console.log(book);
```

```
    </React.Fragment>
  );
};

export default AddBook;
```

Neste arquivo, exibimos um componente `BookForm` (formulário de livros, que ainda não criamos).

Para o componente `BookForm`, passamos o método `handleOnSubmit` para que possamos fazer o processamento mais tarde, quando enviarmos o formulário.

Agora, crie um arquivo `BookForm.js` dentro da pasta `components` com o seguinte conteúdo:

```
import React, { useState } from 'react';
import { Form, Button } from 'react-bootstrap';
import { v4 as uuidv4 } from 'uuid';

const BookForm = (props) => {
  const [book, setBook] = useState({
    bookname: props.book ? props.book.bookname : '',
    author: props.book ? props.book.author : '',
    quantity: props.book ? props.book.quantity : '',
    price: props.book ? props.book.price : '',
    date: props.book ? props.book.date : ''
  });

  const [errorMsg, setErrorMsg] = useState('');
  const { bookname, author, price, quantity } = book;

  const handleOnSubmit = (event) => {
    event.preventDefault();
    const values = [bookname, author, price, quantity];
    let errorMsg = '';
```

```
  ...

  if (allFieldsFilled) {
    const book = {
      id: uuidv4(),
      bookname,
      author,
      price,
      quantity,
      date: new Date()
    };
    props.handleOnSubmit(book);
  } else {
    errorMsg = 'Please fill out all the fields.';
  }
  setErrorMsg(errorMsg);
};

const handleInputChange = (event) => {
  const { name, value } = event.target;
  switch (name) {
    case 'quantity':
      if (value === '' || parseInt(value) === +value) {
        setBook((prevState) => ({
          ...prevState,
          [name]: value
        }));
      }
      break;
    case 'price':
      if (value === '' || value.match(/^\d{1,}(\.\d{0,2})?$/)) {
        setBook((prevState) => ({
          ...prevState,
          [name]: value
        }));
      }
      break;
    default:
      setBook((prevState) => ({
        ...prevState,
        [name]: value
      }));
  }
};
```

```
    <Form.Label>Book Name</Form.Label>
    <Form.Control
      className="input-control"
      type="text"
      name="bookname"
      value={bookname}
      placeholder="Enter name of book"
      onChange={handleInputChange}
    />
  </Form.Group>
  <Form.Group controlId="author">
    <Form.Label>Book Author</Form.Label>
    <Form.Control
      className="input-control"
      type="text"
      name="author"
      value={author}
      placeholder="Enter name of author"
      onChange={handleInputChange}
    />
  </Form.Group>
  <Form.Group controlId="quantity">
    <Form.Label>Quantity</Form.Label>
    <Form.Control
      className="input-control"
      type="number"
      name="quantity"
      value={quantity}
      placeholder="Enter available quantity"
      onChange={handleInputChange}
    />
  </Form.Group>
  <Form.Group controlId="price">
    <Form.Label>Book Price</Form.Label>
    <Form.Control
      className="input-control"
      type="text"
      name="price"
      value={price}
      placeholder="Enter price of book"
      onChange={handleInputChange}
    />
  </Form.Group>
  <Button variant="primary" type="submit" className="submit-btn"
```

```
};

export default BookForm;
```

Vamos entender o que estamos fazendo aqui.

Inicialmente, definimos um estado como um objeto usando o hook `useState` para armazenar todos os detalhes inseridos deste modo:

```
const [book, setBook] = useState({
    bookname: props.book ? props.book.bookname : '',
    author: props.book ? props.book.author : '',
    quantity: props.book ? props.book.quantity : '',
    price: props.book ? props.book.price : '',
    date: props.book ? props.book.date : ''
  });
```

Como usaremos o mesmo componente `BookForm` para adicionar e editar o livro, verificamos primeiro se a prop `book` é passada ou não usando o operador ternário.

Se a prop tiver sido passada, definimos a prop como o valor passado, Do contrário, a definimos como uma string vazia (`''`).

> *Não se preocupe se isso parece complicado agora. Você entenderá melhor quando criarmos as funcionalidades iniciais.*

Em seguida, adicionamos um *state* para exibir uma mensagem de erro e usamos a sintaxe de desestruturação da ES6 para nos referirmos a cada uma das propriedades dentro do *state* assim:

A partir do componente `BookForm`, retornamos um formulário onde inserimos o nome do livro, autor do livro, quantidade e preço. Usamos o react-bootstrap para exibir o formulário em um formato mais bonito.

Cada campo de entrada adicionou um manipulador de eventos `onChange`, que chama o método `handleInputChange`.

Dentro do método `handleInputChange`, adicionamos uma declaração de mudança para alterar o valor do *state* com base no campo de entrada que é alterado.

Quando digitarmos qualquer coisa no campo de entrada `quantity`, `event.target.name` será a quantidade para corresponder ao primeiro caso do switch. Dentro desse caso switch, verificamos se o valor inserido é um número inteiro sem um ponto decimal.

Se for um número inteiro, simplesmente atualizamos o *state* conforme mostrado abaixo:

```
if (value === '' || parseInt(value) === +value) {
  setBook((prevState) => ({
    ...prevState,
    [name]: value
  }));
}
```

Para o caso do switch de `price`, estamos verificando um numero decimal com apenas dois algarismos após a vírgula. Depois, adicionamos uma verificação de expressão regular come esta aparência: `value.match(/^\d{1,}(\d{0,2})?$/)`.

Se o valor de `price` corresponder à expressão regular, atualizamos o *state*.

Observação: tanto para os casos de switch de `quantity` como de `price`, verificamos valores vazios como este: `value === ''`. Isso é para permitir que o usuário exclua totalmente o valor inserido, se necessário.

Sem essa verificação, o usuário não poderá excluir o valor inserido pressionando `Ctrl + A + Delete`.

Para todos os outros campos de entrada, será executado o caso switch que atualizará o *state* com base no valor inserido pelo usuário.

Em seguida, uma vez enviado o formulário, o método `handleOnSubmit` será chamado.

Dentro desse método, verificamos primeiro se o usuário digitou todos os detalhes usando o método de array `every`:

```
const allFieldsFilled = values.every((field) => {
  const value = `${field}`.trim();
  return value !== '' && value !== '0';
});
```

> *Confira o meu artigo aqui para aprender sobre os métodos mais úteis de array JavaScript junto com o seu suporte ao navegador.*

Se todos os valores forem preenchidos, criaremos um objeto com eles. Também chamamos o método `handleOnSubmit` passando livro como argumento. Caso contrário, definimos uma mensagem de erro.

O método `handleOnSubmit` é passado como uma prop do componente `AddBook`.

```
if (allFieldsFilled) {
  const book = {
    id: uuidv4(),
    bookname,
    author,
    price,
    quantity,
    date: new Date()
  };
  props.handleOnSubmit(book);
} else {
  errorMsg = 'Please fill out all the fields.';
}
```

Note que, para criar uma ID única, chamamos o método `uuidv4()` a partir do pacote uuid do npm.

Agora, crie um arquivo `AppRouter.js` dentro da pasta `router` com o seguinte conteúdo:

```
import React from 'react';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
```

```
  return (
    <BrowserRouter>
      <div>
        <Header />
        <div className="main-content">
          <Switch>
            <Route component={BooksList} path="/" exact={true} />
            <Route component={AddBook} path="/add" />
          </Switch>
        </div>
      </div>
    </BrowserRouter>
  );
};

export default AppRouter;
```

Aqui, estabelecemos rotas para vários componentes como `BooksList` e `AddBook` usando a biblioteca `react-router-dom`.

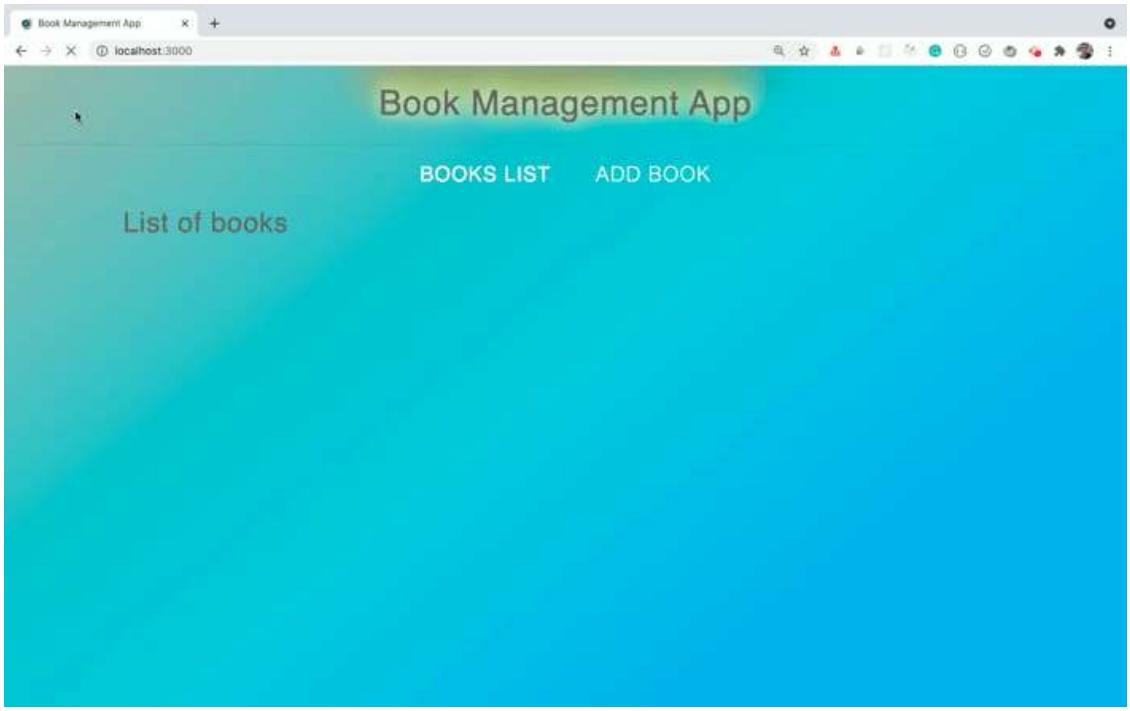
> *Se você é novo no React Router, confira meu curso gratuito de Introdução ao React Router (em inglês).*

Agora, abra o arquivo `src/index.js` e adicione o seguinte conteúdo a ele:

```
import React from 'react';
import ReactDOM from 'react-dom';
import AppRouter from './router/AppRouter';
import 'bootstrap/dist/css/bootstrap.min.css';
import './styles.scss';

ReactDOM.render(<AppRouter />, document.getElementById('root'));
```

```
yarn start
```

Você verá a tela a seguir ao acessar a aplicação em http://localhost:3000/.



Como você pode ver, conseguimos adicionar corretamente um livro e exibi-lo no console.

Em vez de logar no console, no entanto, vamos adicioná-lo ao armazenamento local.

Como criar um hook personalizado para armazenamento local

O armazenamento local é surpreendente. Ele nos permite armazenar facilmente os dados da aplicação no navegador e é uma alternativa aos cookies para o armazenamento de dados.

A vantagem de se usar o armazenamento local é que os dados serão salvos permanentemente no cache do navegador até que os excluamos manualmente para podermos acessá-los mesmo após a atualização da página. Como você deve saber, os dados armazenados no *state* do React serão perdidos uma vez que atualizarmos a página.

Para adicionar dados ao armazenamento local, usamos o método `setItem`, fornecendo uma chave e um valor:

```
localStorage.setItem(key, value)
```

> *Tanto a chave quanto o valor precisam ser strings. Podemos, contudo, armazenar o objeto JSON também usando o método `JSON.stringify`.*

Para aprender em detalhes sobre o armazenamento local e sobre suas diversas aplicações, confira este artigo (em inglês).

Crie um arquivo `useLocalStorage.js` dentro da pasta `hooks` com o seguinte conteúdo:

```
import { useState, useEffect } from 'react';

const useLocalStorage = (key, initialValue) => {
  const [value, setValue] = useState(() => {
    try {
      const localValue = window.localStorage.getItem(key);
      return localValue ? JSON.parse(localValue) : initialValue;
    } catch (error) {
      return initialValue;
    }
  });

  useEffect(() => {
    window.localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
```

Aqui, usamos um hook `useLocalStorage`, que aceita uma `key` e um `initialValue`.

Para declarar o *state* usando o hook `useState`, estamos usando inicialização preguiçosa (em inglês).

Assim, o código dentro da função passada para `useState` será executado apenas uma vez, mesmo que o hook `useLocalStorage` seja chamado várias vezes em cada renderização da aplicação.

Então, inicialmente, estamos verificando se existe algum valor no armazenamento local com a `key` fornecida e retornamos o valor usando o método `JSON.parse`:

```
try {
  const localValue = window.localStorage.getItem(key);
  return localValue ? JSON.parse(localValue) : initialValue;
} catch (error) {
  return initialValue;
}
```

Depois, se houver alguma mudança em `key` ou em `value`, atualizaremos o armazenamento local:

```
useEffect(() => {
    window.localStorage.setItem(key, JSON.stringify(value));
}, [key, value]);

return [value, setValue];
```

atualizar os dados do armazenamento local.

Como usar o hook de armazenamento local

Agora, vamos usar este hook chamado `useLocalStorage` para que possamos adicionar ou remover dados do armazenamento local.

Abra o arquivo `AppRouter.js` e use o hook de `useLocalStorage` dentro do componente:

```
import useLocalStorage from '../hooks/useLocalStorage';

const AppRouter = () => {
 const [books, setBooks] = useLocalStorage('books', []);

 return (
  ...
 )
}
```

Agora, precisamos passar `books` e `setBooks` como props para o componente `AddBook` para que possamos adicionar o livro ao armazenamento local.

Portanto, mude a rota a partir deste código:

```
<Route component={AddBook} path="/add" />
```

```
<Route
  render={(props) => (
    <AddBook {...props} books={books} setBooks={setBooks} />
  )}
  path="/add"
/>
```

Aqui, estamos usando o padrão de renderização de props para passar as props padrão do React Router juntamente com `books` e `setBooks`.

> *Confira o meu curso gratuito de Introdução ao React Router (em inglês) para entender melhor esse padrão de renderização de props e a importância de se usar a palavra-chave `render` em vez de `component`.*

Seu arquivo `AppRouter.js` terá agora esta aparência:

```
import React from 'react';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
import Header from '../components/Header';
import AddBook from '../components/AddBook';
import BooksList from '../components/BooksList';
import useLocalStorage from '../hooks/useLocalStorage';

const AppRouter = () => {
  const [books, setBooks] = useLocalStorage('books', []);

  return (
    <BrowserRouter>
      <div>
        <Header />
        <div className="main-content">
          <Switch>
            <Route component={BooksList} path="/" exact={true} />
            <Route
```

```
          </Switch>
        </div>
      </div>
    </BrowserRouter>
  );
};

export default AppRouter;
```

Agora, abra `AddBook.js` e substitua o seu conteúdo com o seguinte código:

```
import React from 'react';
import BookForm from './BookForm';

const AddBook = ({ history, books, setBooks }) => {
  const handleOnSubmit = (book) => {
    setBooks([book, ...books]);
    history.push('/');
  };

  return (
    <React.Fragment>
      <BookForm handleOnSubmit={handleOnSubmit} />
    </React.Fragment>
  );
};

export default AddBook;
```

Primeiro, usamos a sintaxe de desestruturação da ES6 para acessar as props `history`, `books` e `setBooks` no componente.

Armazenamos todos os livros adicionados em um array. Dentro do método `handleOnSubmit`, chamamos a função `setBooks` passando um array ao adicionar primeiro um livro recém-adicionado e depois fazendo o spread de todos os livros já adicionados no array `books`, como mostrado abaixo:

```
setBooks([book, ...books]);
```

Aqui, adicionamos primeiro o livro recém-adicionado e, depois, fazemos o spread dos livros já adicionados, pois queremos que o último livro seja exibido primeiro quando mostrarmos a lista de livros mais tarde.

Você, no entanto, pode mudar a ordem se quiser:

```
setBooks([...books, book]);
```

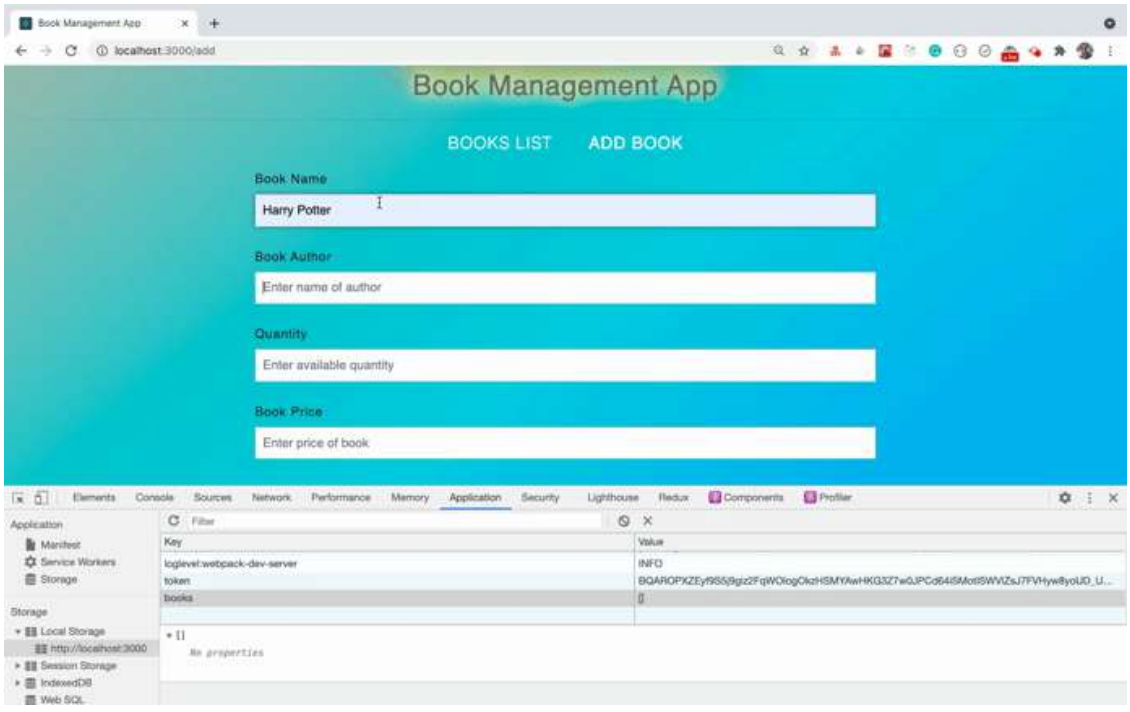
Isto adicionará o livro recém-adicionado no final de todos os livros já adicionados.

Podemos usar o operador spread, pois sabemos que `books` é um array (que iniciamos como um array vazio [] no arquivo `AppRouter.js`, como mostrado abaixo):

Então, uma vez que o livro é adicionado ao armazenamento local chamando o método `setBooks`, dentro do método `handleOnSubmit`, estamos redirecionando o usuário para a página `Books List` usando o método `history.push`:

```
history.push('/');
```

Agora, vamos verificar se conseguimos salvar ou não os livros no armazenamento local.



Como você pode ver, o livro está sendo corretamente adicionado ao armazenamento local (e você pode confirmar isso na aba *Applications* das ferramentas de desenvolvedor do Chrome).

usuario

Agora, vamos exibir os livros adicionados na UI (interface do usuário) no menu de `Books List`.

Abra o `AppRouter.js` e passe `books` e `setBooks` como props para o componente `BooksList`.

O seu arquivo `AppRouter.js` terá esta aparência agora:

```
import React from 'react';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
import Header from '../components/Header';
import AddBook from '../components/AddBook';
import BooksList from '../components/BooksList';
import useLocalStorage from '../hooks/useLocalStorage';

const AppRouter = () => {
  const [books, setBooks] = useLocalStorage('books', []);

  return (
    <BrowserRouter>
      <div>
        <Header />
        <div className="main-content">
          <Switch>
            <Route
              render={(props) => (
                <BooksList {...props} books={books} setBooks={setBooks}
              )}
              path="/"
              exact={true}
            />
            <Route
              render={(props) => (
                <AddBook {...props} books={books} setBooks={setBooks} /
              )}
              path="/add"
            />
          </Switch>
```

```
export default AppRouter;
```

Aqui, acabamos de alterar a primeira rota relacionada ao componente `BooksList`.

Agora, crie um arquivo `Book.js` dentro da pasta `components` com o seguinte conteúdo:

```
import React from 'react';
import { Button, Card } from 'react-bootstrap';

const Book = ({
  id,
  bookname,
  author,
  price,
  quantity,
  date,
  handleRemoveBook
}) => {
  return (
    <Card style={{ width: '18rem' }} className="book">
      <Card.Body>
        <Card.Title className="book-title">{bookname}</Card.Title>
        <div className="book-details">
          <div>Author: {author}</div>
          <div>Quantity: {quantity} </div>
          <div>Price: {price} </div>
          <div>Date: {new Date(date).toDateString()}</div>
        </div>
        <Button variant="primary">Edit</Button>{' '}
        <Button variant="danger" onClick={() => handleRemoveBook(id)}>
          Delete
        </Button>
      </Card.Body>
    </Card>
```

Agora, abra o arquivo `BooksList.js` e substitua o seu conteúdo pelo seguinte código:

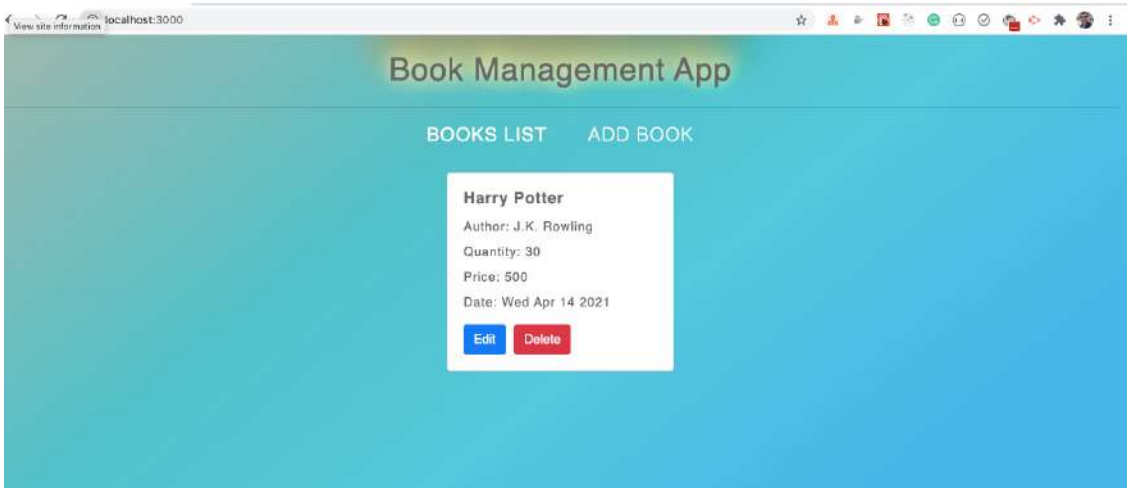
```
import React from 'react';
import _ from 'lodash';
import Book from './Book';

const BooksList = ({ books, setBooks }) => {

  const handleRemoveBook = (id) => {
    setBooks(books.filter((book) => book.id !== id));
  };

  return (
    <React.Fragment>
      <div className="book-list">
        {!_.isEmpty(books) ? (
          books.map((book) => (
            <Book key={book.id} {...book} handleRemoveBook={handleRemov
          ))
        ) : (
          <p className="message">No books available. Please add some bo
        )}
      </div>
    </React.Fragment>
  );
};

export default BooksList;
```

Neste arquivo, estamos percorrendo `books` em um laço usando o método de array `map` e passando os livros como uma prop para o componente `Book`.

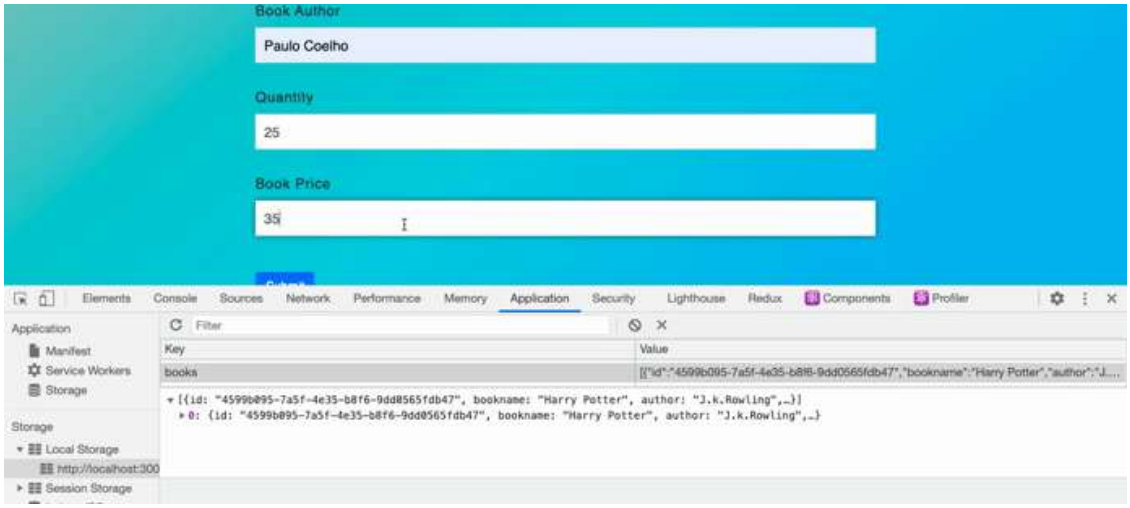
Dentro da função `handleRemoveBook`, estamos chamando a função `setBooks` usando o método de array `filter` para manter somente os livros que não correspondem ao ID do livro fornecido.

```
const handleRemoveBook = (id) => {
    setBooks(books.filter((book) => book.id !== id));
};
```

Agora, se você verificar a aplicação acessando http://localhost:3000/, poderá ver o livro adicionado na interface do usuário.



Vamos adicionar outro livro para verificar todo o fluxo.



Como você pode ver, quando adicionamos um novo livro, somos redirecionados para a página da lista onde podemos excluir o livro. Você pode ver que ele é instantaneamente excluído da UI, bem como do armazenamento local.

Quando atualizamos a página, além disso, os dados não se perdem. Esse é o poder do armazenamento local.

Como editar um livro

Já temos a funcionalidade de adicionar e excluir os livros. Agora, vamos adicionar um modo de editar os livros que temos.

Abra `Book.js` e mude o código abaixo:

```
<Button variant="primary">Edit</Button>{' '}
```

para este código:

Aqui, adicionamos um manipulador de eventos `onClick` para redirecionar o usuário para a rota `/edit/id_do_livro` quando clicamos no botão de edição.

Não temos, entretanto, acesso ao objeto `history` no componente `Book`, pois a prop `history` é passada somente aos componentes que são mencionados em `<Route />`.

Estamos renderizando o componente `Book` dentro do componente `BookList` para que possamos ter acesso a `history` somente dentro do componente `BookList`. Então, podemos passá-la como uma prop ao componente `Book`.

Em vez disso, no entanto, o React Router oferece uma maneira fácil de usar o hook `useHistory`.

Importe o hook `useHistory` no topo do arquivo `Book.js`:

```
import { useHistory } from 'react-router-dom';
```

e, dentro do componente `Book`, chame o hook `useHistory`.

```
const Book = ({
  id,
  bookname,
  author,
  price,
```

```
  ...
}
```

Agora, temos acesso ao objeto `history` dentro do componente `Book`.

O arquivo `Book.js` terá agora este aparência:

```
import React from 'react';
import { Button, Card } from 'react-bootstrap';
import { useHistory } from 'react-router-dom';

const Book = ({
  id,
  bookname,
  author,
  price,
  quantity,
  date,
  handleRemoveBook
}) => {
  const history = useHistory();

  return (
    <Card style={{ width: '18rem' }} className="book">
      <Card.Body>
        <Card.Title className="book-title">{bookname}</Card.Title>
        <div className="book-details">
          <div>Author: {author}</div>
          <div>Quantity: {quantity} </div>
          <div>Price: {price} </div>
          <div>Date: {new Date(date).toDateString()}</div>
        </div>
        <Button variant="primary" onClick={() => history.push(`/edit/$
          Edit
        </Button>{' '}
        <Button variant="danger" onClick={() => handleRemoveBook(id)}>
          Delete
```

```
export default Book;
```

Crie um arquivo chamado `EditBook.js` dentro da pasta `components` com o seguinte conteúdo:

```
import React from 'react';
import BookForm from './BookForm';
import { useParams } from 'react-router-dom';

const EditBook = ({ history, books, setBooks }) => {
  const { id } = useParams();
  const bookToEdit = books.find((book) => book.id === id);

  const handleOnSubmit = (book) => {
    const filteredBooks = books.filter((book) => book.id !== id);
    setBooks([book, ...filteredBooks]);
    history.push('/');
  };

  return (
    <div>
      <BookForm book={bookToEdit} handleOnSubmit={handleOnSubmit} />
    </div>
  );
};

export default EditBook;
```

Aqui, para o manipulador de eventos `onClick` do botão Edit, redirecionamos o usuário para a rota `/edit/id_do_livro` – essa rota, no entanto, ainda não existe. Portanto, vamos criá-la primeiro.

```
<Switch>
...
<Route
  render={(props) => (
    <EditBook {...props} books={books} setBooks={setBooks} />
  )}
  path="/edit/:id"
/>
<Route component={() => <Redirect to="/" />} />
</Switch>
```

A primeira rota é para o componente `EditBook`. Aqui, o caminho é definido como `/edit/:id` onde `:id` representa qualquer id aleatória.

A segunda rota serve para lidar com todas as outras rotas que não correspondem a nenhuma das rotas mencionadas.

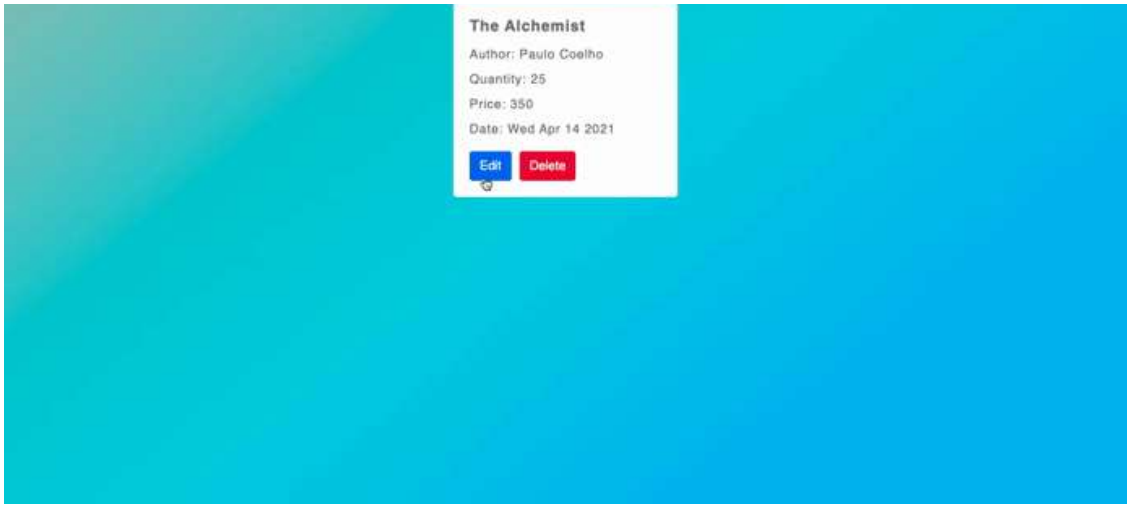
Assim, se acessarmos qualquer rota aleatória, como `/help` ou `/contact`, redirecionaremos o usuário para a rota `/`, que é a do componente `BooksList`.

O arquivo `AppRouter.js` terá agora esta aparência:

```
import React from 'react';
import { BrowserRouter, Switch, Route } from 'react-router-dom';
import Header from '../components/Header';
import AddBook from '../components/AddBook';
import BooksList from '../components/BooksList';
import useLocalStorage from '../hooks/useLocalStorage';

const AppRouter = () => {
  const [books, setBooks] = useLocalStorage('books', []);
```

```
        <Switch>
          <Route
            render={(props) => (
              <BooksList {...props} books={books} setBooks={setBooks}
            )}
            path="/"
            exact={true}
          />
          <Route
            render={(props) => (
              <AddBook {...props} books={books} setBooks={setBooks} />
            )}
            path="/add"
          />
          <Route
            render={(props) => (
              <EditBook {...props} books={books} setBooks={setBooks}
            )}
            path="/edit/:id"
          />
          <Route component={() => <Redirect to="/" />} />
        </Switch>
      </div>
    </div>
  </BrowserRouter>
);
};

export default AppRouter;
```

Vamos verificar a funcionalidade de edição do aplicativo.



Como você pode ver, somos capazes de editar o livro com sucesso. Vamos entender como isso funciona.

Primeiro, dentro do arquivo `AppRouter.js`, temos uma rota como esta:

```
<Route
  render={(props) => (
    <EditBook {...props} books={books} setBooks={setBooks} />
  )}
  path="/edit/:id"
/>
```

e dentro do arquivo `Book.js`, temos um botão de edição, como este:

```
<Button variant="primary" onClick={() => history.push(`/edit/${id}`)}>
  Edit
</Button>
```

ser editado.

Depois, dentro do componente `EditBook`, usamos o hook `useParams` fornecido pelo `react-router-dom` para acessar `props.params.id`.

Portanto, as duas linhas abaixo são idênticas.

```
const { id } = useParams();

// A linha de código acima equivale à linha abaixo

const { id } = props.match.params;
```

Uma vez que tenhamos obtido essa identificação, usaremos o método de array `find` para encontrar o livro específico a partir da lista de livros com a id correspondente fornecida.

```
const bookToEdit = books.find((book) => book.id === id);
```

e esse livro específico é passado para o componente `BookForm` como uma prop `book`:

```
<BookForm book={bookToEdit} handleOnSubmit={handleOnSubmit} />
```

```
const [book, setBook] = useState({
  bookname: props.book ? props.book.bookname : '',
  author: props.book ? props.book.author : '',
  quantity: props.book ? props.book.quantity : '',
  price: props.book ? props.book.price : '',
  date: props.book ? props.book.date : ''
});
```

Aqui, verificamos se a prop `book` existe. Se existir, usamos os detalhes do livro passado como uma prop. Caso contrário, inicializamos o *state* com um valor vazio (`''`) para cada propriedade.

Cada um dos elementos de entrada forneceu uma prop `value` , que estamos definindo a partir do *state*, deste modo:

```
<Form.Control
  ...
  value={bookname}
  ...
/>
```

Podemos, porém, melhorar um pouco a sintaxe de `useState` dentro do componente `BookForm` .

Ao invés de definir diretamente um objeto para o hook `useState` , podemos usar a inicialização preguiçosa como fizemos no arquivo `useLocalStorage.js` .

Assim, mude o código abaixo:

```
    author: props.book ? props.book.author : '',
    quantity: props.book ? props.book.quantity : '',
    price: props.book ? props.book.price : '',
    date: props.book ? props.book.date : ''
  });
```

para este código:

```
const [book, setBook] = useState(() => {
  return {
    bookname: props.book ? props.book.bookname : '',
    author: props.book ? props.book.author : '',
    quantity: props.book ? props.book.quantity : '',
    price: props.book ? props.book.price : '',
    date: props.book ? props.book.date : ''
  };
});
```

Em função dessa mudança, o código para definir o *state* não será executado em cada renderização da aplicação. Ele será executado apenas uma vez quando o componente for montado.

> *Note que a renderização do componente acontece em cada state ou mudança de prop.*

Se você verificar a aplicação, verá que a aplicação funciona exatamente como antes, sem qualquer problema. Nós apenas melhoramos um pouco o desempenho da aplicação.

Como usar a API React Context

passando as mesmos props `books` e `setBooks` para cada um dos componentes, usando o padrão de renderização de props.

Assim, podemos usar a API React Context para simplificar esse código.

> *Note que este é um passo opcional. Você não precisa usar a API React Context, já que estamos passando as props com apenas um nível de profundidade, o código atual está funcionando perfeitamente bem e não usamos nenhuma abordagem errada para passar as props*

Apenas para tornar o código do Router mais simples e para dar uma ideia de como aproveitar o poder da API React Context, usaremos a API em nossa aplicação.

Crie um arquivo `BooksContext.js` dentro da pasta de contexto com o seguinte conteúdo:

```
import React from 'react';

const BooksContext = React.createContext();

export default BooksContext;
```

Agora, dentro do arquivo `AppRouter.js`, importe o contexto exportado acima.

```
import BooksContext from '../context/BooksContext';
```

```
const AppRouter = () => {
  const [books, setBooks] = useLocalStorage('books', []);

  return (
    <BrowserRouter>
      <div>
        <Header />
        <div className="main-content">
          <BooksContext.Provider value={{ books, setBooks }}>
            <Switch>
              <Route component={BooksList} path="/" exact={true} />
              <Route component={AddBook} path="/add" />
              <Route component={EditBook} path="/edit/:id" />
              <Route component={() => <Redirect to="/" />} />
            </Switch>
          </BooksContext.Provider>
        </div>
      </div>
    </BrowserRouter>
  );
};
```

Aqui, convertemos o padrão de renderização das props para as rotas normais e adicionamos todo o bloco de `Switch` dentro do componente `BooksContext.Provider` assim:

```
<BooksContext.Provider value={{ books, setBooks }}>
 <Switch>
 ...
 </Switch>
</BooksContext.Provider>
```

Aqui, para o componente `BooksContext.Provider` , fornecemos uma prop `value` ao passar os dados que queremos acessar dentro dos

podera acessar `books` e `setBooks` atraves da API React Context.

Agora, abra o arquivo `BooksList.js` e remova as props `books` e `setBooks` que estão desestruturadas, pois não estamos mais passando diretamente as props.

Importe o `BooksContext` e `useContext` na parte superior do arquivo:

```
import React, { useContext } from 'react';
import BooksContext from '../context/BooksContext';
```

Acima da função `handleRemoveBook` , adicione o seguinte código:

```
const { books, setBooks } = useContext(BooksContext);
```

Aqui, estamos retirando os livros e os adereços de `BooksContext` usando o hook `useContext` .

O arquivo `BooksList.js` terá agora esta aparência:

```
import React, { useContext } from 'react';
import _ from 'lodash';
import Book from './Book';
import BooksContext from '../context/BooksContext';

const BooksList = () => {
  const { books, setBooks } = useContext(BooksContext);
```

```
    <React.Fragment>
      <div className="book-list">
        {!_.isEmpty(books) ? (
          books.map((book) => (
            <Book key={book.id} {...book} handleRemoveBook={handleRemov
          ))
        ) : (
          <p className="message">No books available. Please add some bo
        )}
      </div>
    </React.Fragment>
  );
};

export default BooksList;
```

Agora, faça mudanças similares no arquivo `AddBook.js`.

O arquivo `AddBooks.js` terá agora esta aparência:

```
import React, { useContext } from 'react';
import BookForm from './BookForm';
import BooksContext from '../context/BooksContext';

const AddBook = ({ history }) => {
  const { books, setBooks } = useContext(BooksContext);

  const handleOnSubmit = (book) => {
    setBooks([book, ...books]);
    history.push('/');
  };

  return (
    <React.Fragment>
      <BookForm handleOnSubmit={handleOnSubmit} />
    </React.Fragment>
  );
};
```

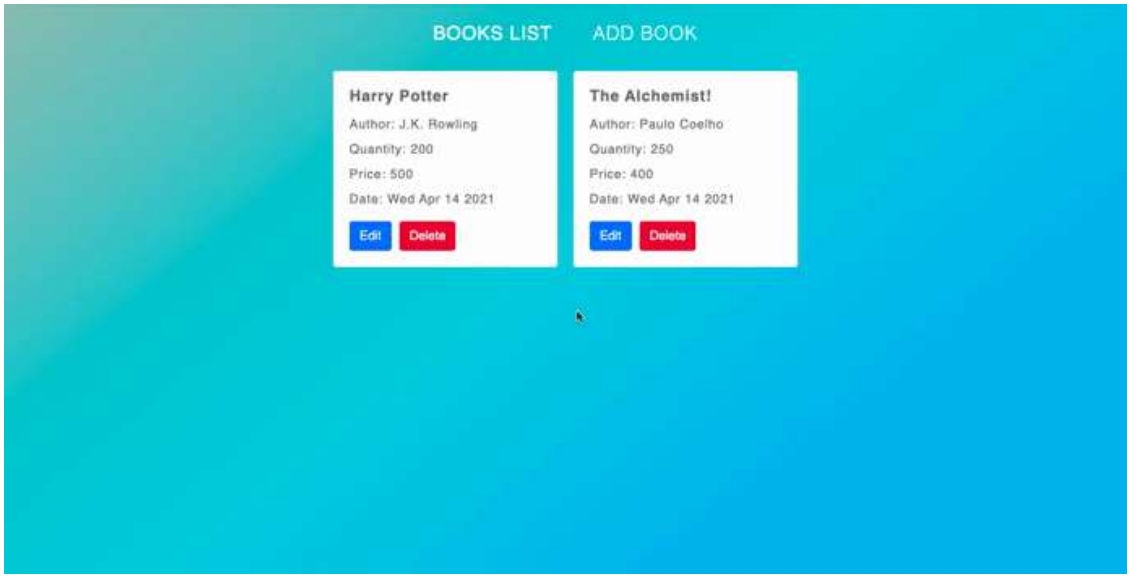
Note que, aqui, ainda estamos usando a desestruturação para a prop `history`. Apenas removemos `books` e `setBooks` da sintaxe de desestruturação.

Agora, faça mudanças similares no arquivo `EditBook.js`.

O arquivo `EditBooks.js` terá agora esta aparência:

```
import React, { useContext } from 'react';
import BookForm from './BookForm';
import { useParams } from 'react-router-dom';
import BooksContext from '../context/BooksContext';

const EditBook = ({ history }) => {
  const { books, setBooks } = useContext(BooksContext);
  const { id } = useParams();
  const bookToEdit = books.find((book) => book.id === id);

  const handleOnSubmit = (book) => {
    const filteredBooks = books.filter((book) => book.id !== id);
    setBooks([book, ...filteredBooks]);
    history.push('/');
  };

  return (
    <div>
      <BookForm book={bookToEdit} handleOnSubmit={handleOnSubmit} />
    </div>
  );
};

export default EditBook;
```

Se você verificar a aplicação, verá que ela funciona exatamente como antes, mas agora estamos usando a API React Context.



Se você quiser entender a API React Context em detalhes, confira este artigo (texto em inglês).

Obrigado pela leitura!

Você pode encontrar o código-fonte completo para esta aplicação neste repositório.

Quer aprender todas as características do ES6+ em detalhes, incluindo let e const, promises, vários métodos de promises, desestruturação de arrays e objetos, arrow functions, async/await, importação e exportação e muito mais?

Confira o livro do autor, Mastering Modern JavaScript (em inglês). Esse livro cobre todos os pré-requisitos para aprender React e ajuda você a se tornar melhor em JavaScript e React.

> Confira uma prévia do conteúdo do livro gratuitamente aqui (em inglês).

Deseja manter-se atualizado com o conteúdo regular sobre JavaScript, React, Node.js? Siga o autor no LinkedIn.

Tradutor: Elizabete Nakamura

I translate and transcribe articles, texts, videos and proofreading of correct grammar, punctuation and writing of English and Portuguese. I do work volunteer for Free Code Camp translating articles.

Autor: Yogesh Chavan (em inglês)

Technical Writer | Freelancer and Full Stack Developer | JavaScript | React | Node.js. https://dev.to/myogeshchavan97

Se você leu até aqui, agradeça ao autor para mostrar que você se importa com o trabalho. Agradeça

Aprenda a programar gratuitamente. O plano de estudos em código aberto do freeCodeCamp já ajudou mais de 40.000 pessoas a obter empregos como desenvolvedores. Comece agora