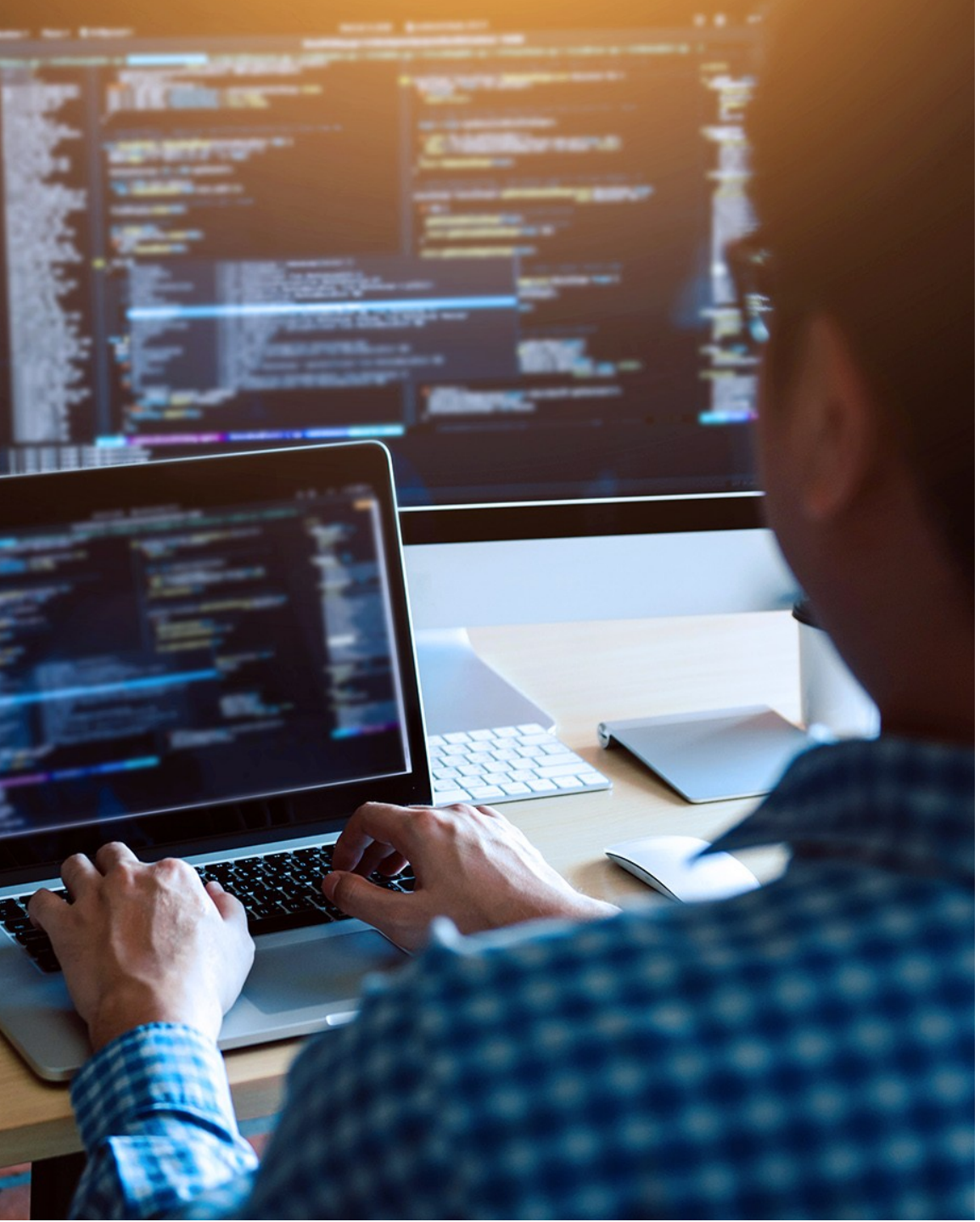

DATAPREV

React

UMA BIBLIOTECA JAVASCRIPT PARA CRIAR INTERFACES DE USUÁRIO

Ricardo Glodzinski
Analista de Tecnologia da Informação



AULA 5

PLANEJAMENTO

- Hooks
- Typescript
- Consumindo API externa
- Estado da aplicação com Redux
- Empacotamento para produção
- Encerramento

HOOKS

- Hooks são uma nova adição ao React e estão disponíveis a partir da versão 16.8.0.
- Eles permitem que você use o state e outros recursos do React sem escrever uma classe.
- Completamente opcionais.
- 100% retro compatíveis
- Não há planos de remoção do uso de classes.
- A minha opinião **pessoal** é usar, gradualmente, em projetos já existentes, e sempre usar em novos projetos.

HOOKS

useState

- Utiliza a sintaxe atribuição via desestruturação.
- Permite adicionar estado a componentes funcionais.
- Recebe um único parâmetro que é o valor inicial desse estado.
- Retorna um par de valores: o estado atual e uma função para atualizar o estado, respectivamente.
- É possível passar qualquer valor para para esse hook: objeto, string, número, booleano, etc.

HOOKS

useState

```
import React, { useState } from "react";

const Counter = () => {
    const [count, setCount] = useState(0);

    return (
        <div>
            <p>Você clicou {count} vezes no botão</p>
            <button onClick={() => setCount(count + 1)}>Incrementar</button>
        </div>
    );
};

export default Counter;
```

Hands on!

HOOKS

useEffect

- O Effect Hook (Hook de Efeito) permite executar efeitos colaterais em componentes funcionais.
- Buscar dados, configurar uma subscription, e mudar o DOM manualmente dentro dos componentes React são exemplos de efeitos colaterais.
- O Hook useEffect pode ser visto como se fossem os métodos do ciclo de vida dos componentes React, componentDidMount, componentDidUpdate, e componentWillUnmount, combinados.
- Por padrão, ele roda depois da primeira renderização e depois de toda atualização.

HOOKS

useEffect

```
import React, { useState, useEffect } from "react";

const CounterUseEffect = () => {
    const [count, setCount] = useState(0);

    useEffect(() => {
        document.title = `Você clicou ${count} vezes no botão`;
    });

    return (
        <div>
            <p>Você clicou {count} vezes no botão</p>
            <button onClick={() => setCount(count + 1)}>Incrementar</button>
        </div>
    );
};

export default CounterUseEffect;
```

Hands on!

HOOKS

useEffect – EFEITOS SEM LIMPEZA

- Executa algum código adicional depois que o React atualiza a DOM.
- Requisições, mutações manuais do DOM e log são exemplos comuns de efeitos que não precisam de limpeza.
- O contador que implementamos é um exemplo que não precisa efetuar nenhuma limpeza.

HOOKS

useEffect – EFEITOS COM LIMPEZA

- O React executa a limpeza quando o componente desmonta.
- O React também limpa os efeitos da renderização anterior antes de rodar os efeitos da próxima vez.
- Para efetuar a limpeza basta retornar uma função com o código a ser executado.

HOOKS

useEffect – EFEITOS COM LIMPEZA

```
import React, { useState, useEffect } from "react";

const CounterUseEffectLimpeza = () => {
    const [inputValue, setInputValue] = useState("");
    const [originalTitle] = useState(document.title);

    const showHourInTitle = () => {
        const now = new Date();
        document.title = `Agora é ${now.getHours()}:${now.getMinutes()}:${now.getSeconds()}`;
    }

    useEffect(() => {
        console.log("Ligando intervalo de showHourInTitle");
        const intervalId = setInterval(showHourInTitle, 1000);
        return () => {
            console.log("LIMPANDO intervalo de showHourInTitle");
            clearInterval(intervalId);
            document.title = originalTitle;
        };
    });

    return (
        <div>
            <p>Você escreveu: {inputValue}</p>
            <input type="text" onChange={e => setInputValue(e.target.value)} />
        </div>
    );
};

export default CounterUseEffectLimpeza;
```

Hands on!

HOOKS

useEffect – DICAS

- Usar **múltiplos** efeitos para separar preocupações: hooks permitem dividir o código com base no que ele está fazendo em vez de encaixá-lo em algum dos métodos do ciclo de vida.
  - Código mais conciso.
  - Melhor manutenção.
  - Menos bugs.

HOOKS

useEffect – MELHORANDO A PERFORMANCE

- É possível indicar ao React que o efeito só deve ser executado em determinadas situações.

- Para fazer isso basta passar para a função useEffect um array como segundo parâmetro, indicando quais valores o React deve verificar se tiveram mudança de uma renderização para outra. Assim, se o valor não tiver sido alterado o efeito não será executado.

```
useEffect(() => {
  document.title = `Você clicou ${count} vezes`;
}, [count]); // Apenas re-execute o efeito quando o count mudar
```

HOOKS

useEffect – MELHORANDO A PERFORMANCE

- Se for usada essa abordagem, o array deve, obrigatoriamente, incluir TODOS os valores de escopo (como props e state), que mudam com o tempo, utilizados no efeito.

- É possível passar um array vazio. Dessa forma o efeito será executado apenas uma vez, na montagem do componente, e a limpeza apenas uma vez, na desmontagem do componente.

```
UseEffect(() => {
    (…)
    return () => { (…) };
}, []);
```

HOOKS

REGRAS

- Hooks são funções JavaScript, mas é necessário seguir duas regras ao utilizá-los:

1) Usar Apenas no Nível Superior

  - Não usar Hooks dentro de loops, regras condicionais ou funções aninhadas. Seguindo essas regras, se garante que os Hooks serão chamados na mesma ordem a cada vez que o componente renderizar. É isso que permite que o React preserve corretamente o estado dos Hooks quando são usados várias chamadas a useState e useEffect na mesma função.

HOOKS

REGRAS

- 2) Usar Apenas Dentro de Funções do React
  - Não usar dentro de funções JavaScript comuns. Em vez disso, é possível:
    - Chamar Hooks em componentes React.
    - Chamar Hooks dentro de Hooks Customizados.
  - Seguindo essas regras, se garante que toda lógica de estado (state) no componente seja claramente visível no código fonte.

HOOKS

ESLINT PLUGIN

- Há um plugin ESLint, chamado eslint-plugin-react-hooks, que aplica essas duas regras.

- Esse plugin está incluindo por padrão no Create React App.

HOOKS

CRIANDO HOOKS CUSTOMIZADAS

- Criar Hooks customizadas permite que se extraia a lógica de um componente em funções reutilizáveis.

A mesma lógica pode ser compartilhada entre componentes.

Tradicionalmente em React, havia duas maneiras populares para compartilhar lógica com estado entre componentes: render props e componentes de alta ordem. Hooks resolvem diversos dos mesmos problemas sem nos forçar a adicionar mais componentes à árvore de renderização.

HOOKS

CRIANDO HOOKS CUSTOMIZADAS

- Deve-se, por convenção, nomear a função hook iniciando com “use”. Exemplo: useStatus.
- Dois componentes usando o mesmo hook **NÃO** compartilham estado (state).
- Cada chamada a um Hook gera um estado (state) isolado.
- Visto que Hooks são funções, podemos passar informações entre eles.

HOOKS

CRIANDO HOOKS CUSTOMIZADAS

```
import React, { useState, useContext, createContext } from "react";
import * as AuthAPI from "../../api/auth";

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
    const auth = useProvideAuth();
    return <AuthContext.Provider
value={auth}>{children}</AuthContext.Provider>;
}

export const useAuth = () => {
    return useContext(AuthContext);
};

const useProvideAuth = () => {
    const [user, setUser] = useState(null);

    const isLoggedIn = (user && user !== false);

    const signin = (email, password) => {
        return AuthAPI.signInWithEmailAndPassword(email, password)
            .then(user => {
                setUser(user);
                return user;
            });
    };

    const signout = () => AuthAPI.signOut().then(() => setUser(false));

    return {
        user,
        isLoggedIn,
        signin,
        signout
    };
}
```

```
import React from "react";

import { AuthProvider } from "./useAuth";
import UserProfile from "./UserProfile";
import Footer from "../../Layout/Footer";

const CustomHooks = () => {
    return (
        <AuthProvider>
            <UserProfile />
            <Footer />
        </AuthProvider>
    );
};

export default CustomHooks;
```

```
import React from "react";
import { useAuth } from "../Pages/Hooks/useAuth";

const Footer = () => {
    const auth = useAuth();
    return (
        <footer>
            {auth && auth.user &&
                <div>Usuário autenticado: <b>{auth.user.nome}</b></div>
            }
            {(!auth || !auth.user) &&
                <div>Nenhum usuário autenticado</div>
            }
        </footer>
    );
};

export default Footer;
```

Hands on!

HOOKS

CRIANDO HOOKS CUSTOMIZADAS

```
import React, { useState } from "react";
import { Grid, TextField, Button } from "@material-ui/core";
import { useAuth } from "./useAuth";

const UserProfile = () => {
  const [signinEmail, setSigninEmail] = useState("ricardo.glodzinski@dataprev.gov.br");
  const [signinPassword, setSigninPassword] = useState("1234");
  const auth = useAuth();
  const { user, isLoggedIn, signin, signout } = auth;
  return (
    <div>
      {isLoggedIn &&
        <fieldset>
          <label>Dados do Usuário Autenticado</label>
          <p>Nome: {user.nome}</p>
          <p>E-mail: {user.email}</p>
          <br />
          <Button secondary onClick={signout}>Sair</Button>
        </fieldset>
      }
      {!isLoggedIn &&
        <Grid direction="column" container>
          <Grid item>
            <TextField label="E-mail" value={signinEmail} name="signinEmail" onChange={e => setSigninEmail(e.target.value)} />
          </Grid>
          <Grid item>
            <TextField type="password" label="Senha" value={signinPassword} name="signinPassword" onChange={e => setSigninPassword(e.target.value)} />
          </Grid>
          <Grid item sm={6}>
            <Button primary onClick={() => signin(signinEmail, signinPassword)}>Entrar</Button>
          </Grid>
        </Grid>
      }
    </div>
  );
};

export default UserProfile;
```

Hands on!

HOOKS

API

- Hooks Básicos
  - useState
  - useEffect
  - useContext

- Hooks Adicionais
  - useReducer
  - useCallback
  - useMemo
  - useRef
  - useImperativeHandle
  - useLayoutEffect
  - useDebugValue

https://pt-br.reactjs.org/docs/hooks-reference.html

TYPESCRIPT

- O TypeScript é uma linguagem de código aberto que se baseia no JavaScript, uma das ferramentas mais usadas no mundo, adicionando definições de tipo estáticas.

- Os tipos fornecem uma maneira de descrever a forma de um objeto, fornecendo melhor documentação e permitindo que o TypeScript valide se o seu código está funcionando corretamente.

- Os tipos de escrita podem ser opcionais no TypeScript, porque a inferência de tipo permite obter muito poder sem escrever código adicional.

- O código TypeScript é transformado em código JavaScript por meio do compilador TypeScript ou Babel.

TYPESCRIPT

- Para adicionar o TypeScript em um projeto iniciado com a toolchain Create-React-App:

```
cd nome-da-app
npm install --save typescript @types/node @types/react
@types/react-dom @types/jest
```

Hands on!

```
import * as React from 'react';

export interface ITestComponent {
    name: string;
}

const TestComponent = (props: ITestComponent) => {
    const { name } = props;
    return (
        <div>Olá {name}, seja bem vindo ao uso de TypeScript!</div>
    );
};

export default TestComponent;
```

TYPESCRIPT

HOOKS - useState

- A inferência de tipo funciona muito bem na maioria das vezes:

```
const [val, toggle] = React.useState(false); // ‘val’ é inferido como um booleano,’toggle’ aceitará apenas booleanos
```

- No entanto, muitos vezes, poderão ser providos tipos inicializar os valores com null.

```
const [user, setUser] = React.useState<IUser | null>(null);
```

TYPESCRIPT

HOOKS - useRef

- Existem duas possibilidades de declaração:

```
const ref1 = useRef<HTMLElement>(null!);
const ref2 = useRef<HTMLElement | null>(null);
```

- A primeira opção tornará ref1.current somente leitura e deve ser transmitida para atributos de ref internos que o React gerenciará (porque o React lida com a configuração do valor atual para você).

- A segunda opção tornará ref2.current mutável e se destina a "variáveis de instância" que você gerencia.

TYPESCRIPT

HOOKS - useEffect

- Ao usar useEffect, não retornar nada além de uma função ou undefined.

```
function DelayedEffect(props: { timerMs: number }) {

  const { timerMs } = props;

  // ruim! setTimeout retorna implicitamente um número porque o corpo da função de seta não está envolto em chaves
  useEffect(
    () =>
      setTimeout(() => {
        /* (...) */
      }, timerMs),
    [timerMs]
  );
  return null;
}
```

TYPESCRIPT

COMPONENTES DE CLASSE

- No TypeScript, React.Component é um tipo genérico (React.Component<PropType, StateType>).

```
type MyProps = {
  message: string;
};
type MyState = {
  count: number;
};
class App extends React.Component<MyProps, MyState> {
  state: MyState = {
    count: 0,
  };
  render() {
    return (
      <div>
        {this.props.message} {this.state.count}
      </div>
    );
  }
}
```

TYPESCRIPT

TYPE OU INTERFACE

- As interfaces são diferentes dos tipos no TypeScript, mas podem ser usadas para coisas muito semelhantes:
  - Sempre usar **interface** para a definição da API pública ao criar uma biblioteca ou definições de tipo de ambiente de terceiros.
  - Considere usar o **type** para as props do componente e state, porque é mais restrito.
  - Tipos são úteis para união (por exemplo, type MyType = TypeA | TypeB), enquanto as interfaces são melhores para declarar formas de dicionário e implementá-las ou estendê-las.

TYPESCRIPT

EXEMPLOS DOS TIPOS MAIS COMUNS

```
type AppProps = {
    message: string;
    count: number;
    disabled: boolean;
    names: string[]; //  array tipado
    status: "waiting" | "success"; // string literais com união de duas valores aceitos
    obj: object; // um objeto, desde que não sejam usados as propriedades dele (uso incomun)
    obj2: {}; // quase o mesmo que 'object'; exatamente o mesmo que 'Object'
    /** um objeto com propriedades definidas (preferencial) */
    obj3: {
        id: string;
        title: string;
    };
    /** array de objetos (MUITO UTILIZADO) */
    objArr: {
        id: string;
        title: string;
    }[];
    /** qualquer função, desde que você não a invoque (não recomendado) */
    onSomething: Function;
    /** função que não aceita ou retorna nada (MUITO UTILIZADO) */
    onClick: () => void;
    /** função com prop nomeado (MUITO UTILIZADO) */
    onChange: (id: number) => void;
    /** sintaxe de tipo de função alternativa que leva um evento (MUITO UTILIZADO) */
    onBlue(event: React.MouseEvent<HTMLButtonElement>): void;
    /** '?' indica que a prop é opcional */
    optional?: string;
};
```

CONSUMINDO API EXTERNA

- O processo para consumir uma API externa é relativamente simples.
- Vamos ver apenas como consumir uma API Rest, trafegando dados em formato JSON – JavaScript Object Notation, sem entrar em maiores detalhes sobre Rest ou como a API foi implementada.
- O código fonte da API que iremos consumir está disponível em https://www-scm.prevnet/ricardo.glodzinski/tarefas-app
- A API está executando e respondendo a solicitações em http://10.107.5.150/api/tasks/

CONSUMINDO API EXTERNA

fetch()

- A API Fetch fornece uma interface JavaScript para acessar e manipular partes do pipeline HTTP, tais como os pedidos e respostas. Fornece o método global fetch() que fornece uma maneira fácil e lógica para buscar recursos de forma **assíncrona** através da rede.

```
fetch(URL, {
  method: 'POST', // GET, POST, PUT, DELETE, OPTIONS
  headers: new Headers({
    'Content-Type': 'application/json'
  }),
  body: {}
}).then(function(response) {
  // tratar a resposta
});
```

Referência: https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/fetch

CONSUMINDO API EXTERNA

```jsx
import React, { useState, useEffect } from "react";
import ListaTarefas from "./tarefas.list";
import actions from "./actions-fetch";

const Tarefas = () => {
  const [tarefas, setTarefas] = useState([]);

  const handleCarregar = () => {
    actions.carregar().then(tarefas => setTarefas(tarefas));
  };

  useEffect(() => handleCarregar(), []);

  const handleIncluir = tarefa =>
    actions.incluir(tarefa).then(() => handleCarregar());

  const handleEditar = tarefa =>
    actions.editar(tarefa).then(() => handleCarregar());

  const handleExcluir = tarefa =>
    actions.excluir(tarefa).then(() => handleCarregar());

  const acoes = {
    incluir: handleIncluir,
    editar: handleEditar,
    excluir: handleExcluir
  };

  return <ListaTarefas acoes={acoes} tarefas={tarefas} />;
};

export default Tarefas;
```

```jsx
const TAREFAS_API_BASE_URL = "http://10.107.5.150/api/tasks";
const jsonHeaders = new Headers({ "Content-Type": "application/json" });

const carregar = async () => {
  const response = await fetch(`${TAREFAS_API_BASE_URL}`, {
    method: "GET",
    headers: jsonHeaders
  });
  const tarefas = await response.json();
  return tarefas;
};

const incluir = async tarefa => {
  const response = await fetch(`${TAREFAS_API_BASE_URL}`, {
    method: "POST",
    headers: jsonHeaders,
    body: JSON.stringify(tarefa)
  });
  const tarefaIncluida = await response.json();
  return tarefaIncluida;
};

const editar = async tarefa => {
  const response = await fetch(`${TAREFAS_API_BASE_URL}/${tarefa._id}`, {
    method: "PUT",
    headers: jsonHeaders,
    body: JSON.stringify(tarefa)
  });
  const tarefaEditada = await response.json();
  return tarefaEditada;
};

const excluir = async tarefa => {
  const response = await fetch(`${TAREFAS_API_BASE_URL}/${tarefa._id}`, {
    method: "DELETE",
    headers: jsonHeaders,
    body: JSON.stringify(tarefa)
  });
};

export default { carregar, incluir, editar, excluir };
```

CONSUMINDO API EXTERNA

AXIOS

- É um cliente HTTP baseado em *Promise*, fácil de usar, no navegador e no Node.js.
- Como o Axios é baseado em *Promise*, é possível aproveitar as vantagens do async/await para construir um código mais legível.
- Fornece a possibilidade de interceptar e cancelar a solicitação.
- Possui um recurso integrado que fornece proteção do lado do cliente contra falsificação de solicitação entre sites.

```
cd root-da-sua-app
npm install --save axios
npm install --save-dev @types/axios
```

Referência: https://github.com/axios/axios

CONSUMINDO API EXTERNA

AXIOS

```
import axios from “axios”;

axios.get('/user?ID=12345')
  .then(function (response) {
    // sucesso
    console.log(response);
  })
  .catch(function (error) {
    // erro
    console.log(error);
  })
  .then(function () {
    // sempre executado
  });
```

```
import axios from “axios”;

// Passagem de parâmetros
axios.get('/user', {
    params: {
      ID: 12345
    }
  })
  .then(function (response) {
    console.log(response);
  })
  .catch(function (error) {
    console.log(error);
  })
  .then(function () {
    // sempre executado
  });
```

```
import axios from “axios”;

// usando async/await
async function getUser() {
  try {
    const response = await
axios.get('/user?ID=12345');
    console.log(response);
  } catch (error) {
    console.error(error);
  }
}
```

CONSUMINDO API EXTERNA

```jsx
import React, { useState, useEffect } from "react";
import ListaTarefas from "./tarefas.list";
import actions from "./actions-axios";

const Tarefas = () => {
  const [tarefas, setTarefas] = useState([]);

  const handleCarregar = () => {
    actions.carregar().then(tarefas => setTarefas(tarefas));
  };

  useEffect(() => handleCarregar(), []);

  const handleIncluir = tarefa =>
    actions.incluir(tarefa).then(() => handleCarregar());

  const handleEditar = tarefa =>
    actions.editar(tarefa).then(() => handleCarregar());

  const handleExcluir = tarefa =>
    actions.excluir(tarefa).then(() => handleCarregar());

  const acoes = {
    incluir: handleIncluir,
    editar: handleEditar,
    excluir: handleExcluir
  };

  return <ListaTarefas acoes={acoes} tarefas={tarefas} />;
};

export default Tarefas;
```

```jsx
import axios from "axios";

const TAREFAS_API_BASE_URL = "http://10.107.5.150/api/tasks";

const carregar = async () => {
  const response = await axios.get(TAREFAS_API_BASE_URL);
  return response.data;
};

const incluir = async tarefa => {
  const response = await axios.post(TAREFAS_API_BASE_URL, tarefa);
  return response.data;
};

const editar = async tarefa => {
  const response = await axios.put(
    `${TAREFAS_API_BASE_URL}/${tarefa._id}`,
    tarefa
  );
  return response.data;
};

const excluir = async tarefa => {
  const response = await axios.delete(
    `${TAREFAS_API_BASE_URL}/${tarefa._id}`,
    tarefa
  );
  return response.data;
};

export default { carregar, incluir, editar, excluir };
```

ESTADO DA APLICAÇÃO COM REDUX

REACT REDUX

- Redux é uma biblioteca JavaScript criada por Dan Abramov e Andrew Clark (inspirada na arquitetura Flux do Facebook) que tem como objetivo gerenciar estados globais da nossa aplicação front-end, seguindo a arquitetura Flux.

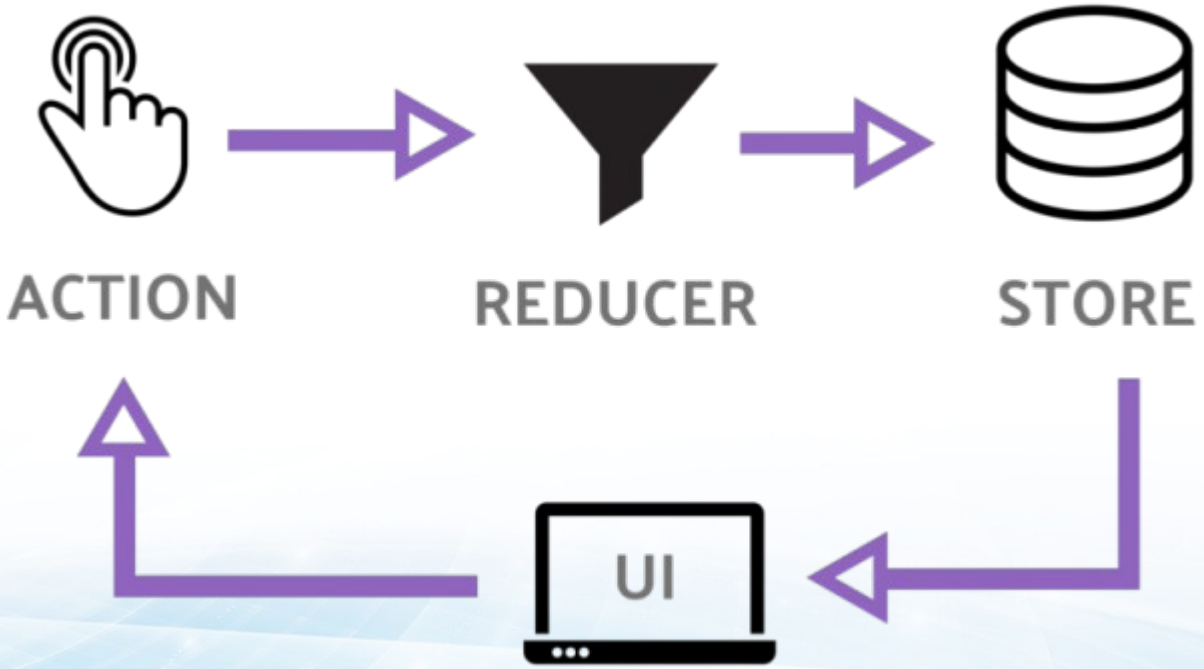
- REDUX: contêiner de estado previsível para aplicativos JS. https://redux.js.org/

- REACT REDUX: ligações oficiais do React para Redux. https://react-redux.js.org/

```
cd root-da-sua-app
npm install --save redux react-redux redux-thunk
```

ESTADO DA APLICAÇÃO COM REDUX

- O Redux é, basicamente, dividido em três partes: **store**, **reducers**, **actions**.



ESTADO DA APLICAÇÃO COM REDUX

- **STORE**: é o conjunto de estados da aplicação.
- É como se fosse um grande centro de informações, que possui disponibilidade para receber e entregar exatamente o que os componentes requisitam.
- Tecnicamente, a store é um objeto JavaScript que possui todos os estados dos seus componentes.

ESTADO DA APLICAÇÃO COM REDUX

- **REDUCERS**: responsáveis por lidar com todas as ações referente a um determinado dado do estado dentro da store.

- Cada dado na store deve possuir um reducer que será responsável por buscar, atualizar, incluir e apagar valores da store.

ESTADO DA APLICAÇÃO COM REDUX

- **ACTIONS**: são responsáveis por solicitar e enviar informações para os reducers.
- Devem ser funções puras.

ESTADO DA APLICAÇÃO COM REDUX

REACT REDUX

- Criar a store.

```
// src/aula5/redux/store.js
import { createStore, applyMiddleware } from "redux";
import thunk from "redux-thunk";

import rootReducer from "./reducers";

const store = createStore(rootReducer);

export default store;
```

- Criar um reducer principal que vai concentrar todos os reducers.

```
// src/aula5/redux/reducers/index.js
import { combineReducers } from "redux";

import tarefasReducer from "./tarefas.reducer";

const rootReducer = combineReducers({
  tarefas: tarefasReducer
});

export default rootReducer;
```

ESTADO DA APLICAÇÃO COM REDUX

REACT REDUX

- No arquivo principal da nossa aplicação, importar a store e conectar a app utilizando o <Provider /> do react-redux.

```
// src/aula5/index.jsx
import React from "react";
import { BrowserRouter as Router } from "react-router-dom";
import { Provider } from "react-redux";

import Menu from "./Layout/Menu";
import Routes from "./routes";

import store from "./redux/store";

const App = () => {
  return (
    <Provider store={store}>
      <Router>
        <Menu />
        <div className="container">
          <Routes />
        </div>
      </Router>
    </Provider>
  );
};

export default App;
```

ESTADO DA APLICAÇÃO COM REDUX

REACT REDUX – REDUCERS E ACTIONS

- Deve ser definido em cada reducer um estado inicial e as actions que serão tratadas por este reducer.

```
import axios from "../config/axios";

const TAREFAS_PATH = "/tarefas";

const initialState = {
  listaTarefas: []
};

export const ACTION_TYPES = {
  CARREGAR: "tarefa/CARREGAR",
  INCLUIR: "tarefa/CRIAR",
  ATUALIZAR: "tarefa/ATUALIZAR",
  EXCLUIR: "tarefa/APAGAR"
};
```

```
export default (state = initialState, action) => {
  const { listaTarefas } = state;
  const { type, payload } = action;

  switch (type) {
    case ACTION_TYPES.CARREGAR:
      return { ...state, listaTarefas: payload };
    case ACTION_TYPES.INCLUIR:
      return { ...state, listaTarefas: [...listaTarefas, payload] };
    case ACTION_TYPES.ATUALIZAR:
      return {
        ...state,
        listaTarefas: listaTarefas.map(tarefa =>
          tarefa._id === payload._id ? payload : tarefa
        )
      };
    case ACTION_TYPES.EXCLUIR:
      return {
        ...state,
        listaTarefas: listaTarefas.filter(tarefa => tarefa._id !==
payload._id)
      };
    default:
      return state;
  }
};
```

ESTADO DA APLICAÇÃO COM REDUX

REACT REDUX – REDUCERS E ACTIONS

- As actions devem retornar um objeto JavaScript, contendo um **type** e um **payload**.
- O type é responsável por indicar ao reducer qual informação deve ser manipulada.
- O payload contém o valor da informação que será recebida pelo reducer.

```
export const carregar = () => dispatch =>
  axios.get(TAREFAS_PATH).then(response =>
    dispatch({
      type: ACTION_TYPES.CARREGAR,
      payload: response.data
    })
  );

export const incluir = tarefa => dispatch =>
  axios.post(TAREFAS_PATH, tarefa).then(response =>
    dispatch({
      type: ACTION_TYPES.INCLUIR,
      payload: response.data
    })
  );

export const editar = tarefa => dispatch =>
  axios.put(`${TAREFAS_PATH}/${tarefa._id}`, tarefa).then(response =>
    dispatch({
      type: ACTION_TYPES.ATUALIZAR,
      payload: response.data
    })
  );

export const excluir = tarefa => dispatch =>
  axios.delete(`${TAREFAS_PATH}/${tarefa._id}`, tarefa).then(response =>
    dispatch({
      type: ACTION_TYPES.EXCLUIR,
      payload: tarefa
    })
  );
```

ESTADO DA APLICAÇÃO COM REDUX

REACT REDUX – CONNECT

- Por fim, utilizar a função ***connect*** para conectar nossos componentes ao estado da aplicação.

- A função ***connect*** recebe dois parâmetros, que, por convenção, são chamados de ***mapStateToProps*** e ***mapDispatchToProps***, respectivamente.

```
import React, { useEffect } from "react";
import { connect } from "react-redux";

import ListaTarefas from "../Tarefas/tarefas.list";

Import { carregar, incluir, editar, excluir} from "../../redux/reducers/tarefas.reducer";

const ReduxPage = props => {
  const { tarefas } = props;

  useEffect(() => {
    props.carregar();
  }, []);

  const acoes = { incluir: props.incluir, editar: props.editar, excluir: props.excluir };

  return <ListaTarefas acoes={acoes} tarefas={tarefas} />;
};

const mapStateToProps = state => ({
  tarefas: state.tarefas.listaTarefas
});

const mapDispatchToProps = { carregar, incluir, editar, excluir };

export default connect(mapStateToProps, mapDispatchToProps)(ReduxPage);
```

EMPACOTAMENTO PARA PRODUÇÃO

- A toolchain create-react-app já provê um script que faz todo o trabalho de empacotamento.

```
cd root-da-sua-app
npm run build
```

- Será gerada uma versão para produção na pasta build.

- O script está configurado para usar caminhos relativos. O padrão empacota para disponibilização na pasta ROOT de um servidor HTTP, como Apache ou Nginx.

- Caso seja necessário utilizar algum contexto no servidor (Ex.: http://dataprev.gov.br/sistema, é possível mudar, no package.json, o atributo “homepage”.

Obrigado!

Ricardo Glodzinski
Analista de Tecnologia da Informação
ricardo.glodzinski@dataprev.gov.br

Julho 2020

www.facebook.com/dataprevtecnologia

@dataprev

DATAPREV