

Desenvolvimento de Software para WEB

Redux

O Redux

Redux

- O Redux é a implementação de um modelo arquitetural chamado “Flux”. Essa implementação não existe apenas para o React, podendo ser usada em outras tecnologias. A grosso modo, o Redux é responsável em **controlar o estado geral da sua aplicação**. Ele consegue isso graças a três elementos fundamentais de sua arquitetura:
 - O **store**, o conjunto de dados da sua aplicação.
 - Os **reducers**, que agem sobre os dados armazenados no store, alterando o estado da aplicação, através de ações (**actions**).
 - As **actions**, que enviam dados para os reducers (que podem ou não alterar os dados da aplicação). São implementadas como funções puras (sem lógica).

Redux

- Instalação
 - De posse do projeto ??, entre com os comandos:
 - `npm install --save redux`
 - `npm install --save react-redux`

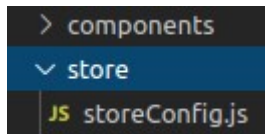
Parte 1

O Store e os Reducers

(Estado Geral da Aplicação)

Redux – O Store Config

- Primeiramente, você deve criar o estado geral da sua aplicação, para isso, crie a seguinte pasta:
 - src/store



- Dentro de store, crie o arquivo **storeConfig.js**
- **storeConfig.js** irá representar o estado geral da aplicação, retornando uma função javascript que cria esse estado.
- Para o seu código, precisamos apenas da biblioteca **redux**.

storeConfig.js

```
import { createStore, combineReducers } from 'redux'
```

```
const reducers = combineReducers({  
  pokemonId: function(state, action){  
    return {  
      id:10  
    }  
  },  
  pokemonNames: function(state,action){  
    return [  
      'Bulbasaur',  
      'Pikachu',  
      'Meow'  
    ]  
  }  
})
```

```
//criando o store a partir dos reducers
```

```
function storeConfig(){  
  return createStore(reducers)  
}
```

```
export default storeConfig
```

storeConfig

- O storeConfig.js armazena **reducers**.
- Cada **reducer** é implementado por uma **função**:

```
function(state, action){  
  return {  
    id:10  
  }  
}
```

- Cada função (reducer) é **especificada** por uma **chave**:

```
pokemonId: function(state, action){  
  return {  
    id:10  
  }  
}
```


storeConfig

- Cada função, que representa a implementação de um reducer, recebe como parâmetro um **state**, e uma **action**
 - **state**: estado anterior que pode ou não ser modificado pela action.
 - **action**: traz informações sobre a modificação do state. São elas (veremos adiante em detalhes):
 - type
 - payload

storeConfig

- No nosso exemplo temos então dois reducers:
 - **pokemonId** : responsável em alterar o **id**
 - **pokemonNames**: apenas para fins didáticos, não faz nada para a aplicação
- A variável **reducers**, criada a partir das funções dos reducers, é então armazenada no estado geral da aplicação (o store).

```
function storeConfig(){  
  return createStore(reducers)  
}
```

storeConfig - Publicando

- Uma vez criado o store (a partir dos reducers) o próximo passo é tornar o **store** visível ao resto da aplicação. Aí que entra a biblioteca **react-redux**.
- Iremos alterar então o código de **index.js** para que o mesmo chame a função de criação do store (em storeConfig.js) e publique para o resto dos componentes.

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import 'bootstrap/dist/css/bootstrap.min.css'
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

import { Provider } from 'react-redux'
import storeConfig from './store/storeConfig'

//criando uma única instância do store pra toda a aplicação.
ReactDOM.render(
  <Provider store={storeConfig()}>
    <React.StrictMode>
      <App />
    </React.StrictMode>
  </Provider>,
  document.getElementById('root')
);
```

index.js

- A classe **Provider** do react-redux é responsável em disponibilizar um store (estado geral) para a aplicação. Como?
- Ao usar a tag **<Provider store={storeConfig()}>**, é chamada a função **storeConfig()** para criação do estado compartilhado com todos os componentes internos à tag **<Provider>**

Store - Consumindo

- Para ter acesso ao store, cada um dos componentes **interessados** deve importar o **react-redux** e se conectar (**connect**) com o estado da aplicação.

```
import { connect } from 'react-redux'
```

```
class Navigate extends Component {
```

```
  ...
```

```
  ...
```

```
}
```

Navigate.jsx

```
//mapeia o estado geral pra o props deste componente
```

```
function mapStateToProps(state) {
```

```
  return {
```

```
    id: state.pokemonId.id,
```

```
  }
```

```
}
```

```
export default connect(mapStateToProps)(Navigate)
```

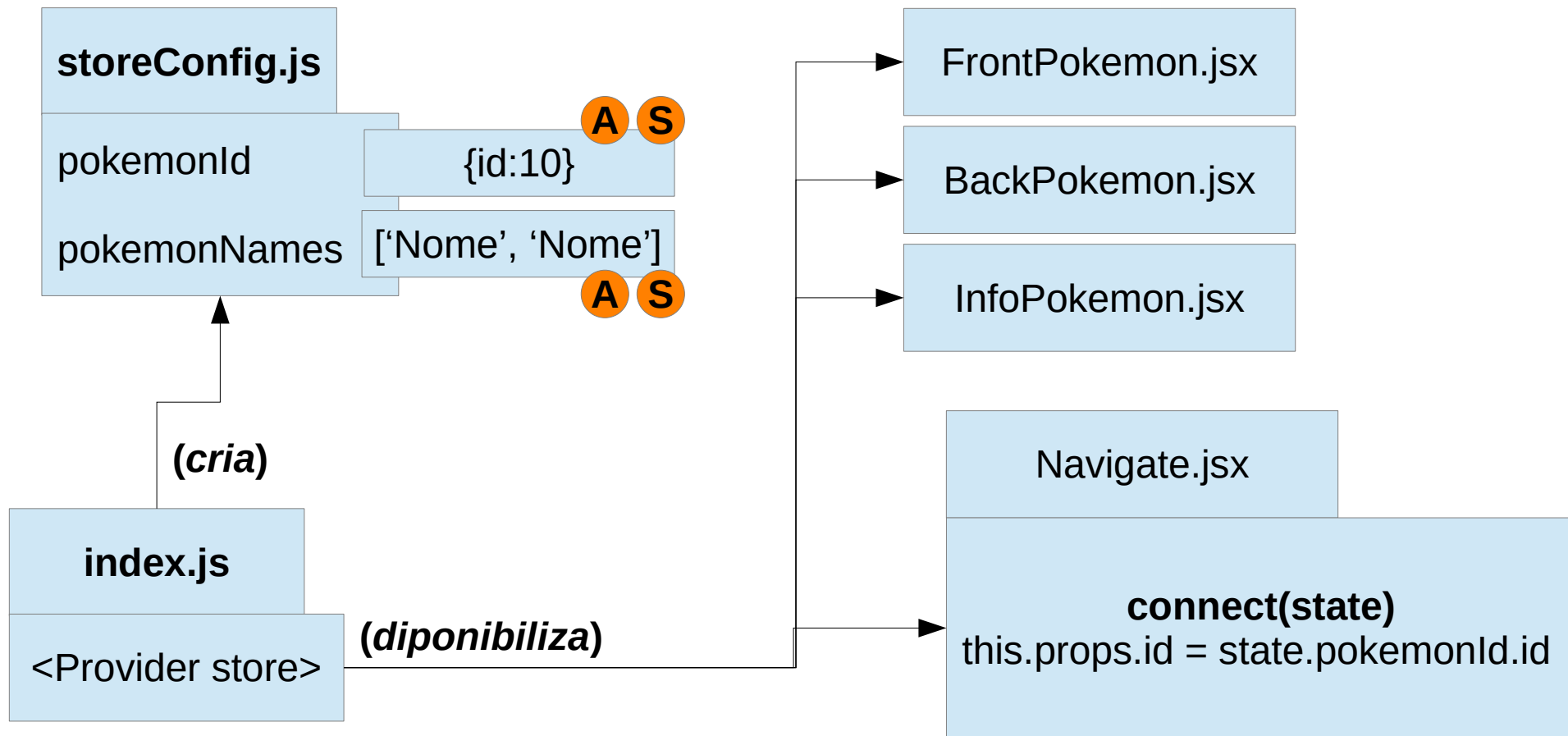
Store - Consumindo

- Deve-se importar o connect
 - `import { connect } from 'react-redux'`
- A função `mapStateToProps(state)` mapeia o estado geral (`stateConfig.js`) para o props do componente que está sendo conectado (no caso o `Navigate`).

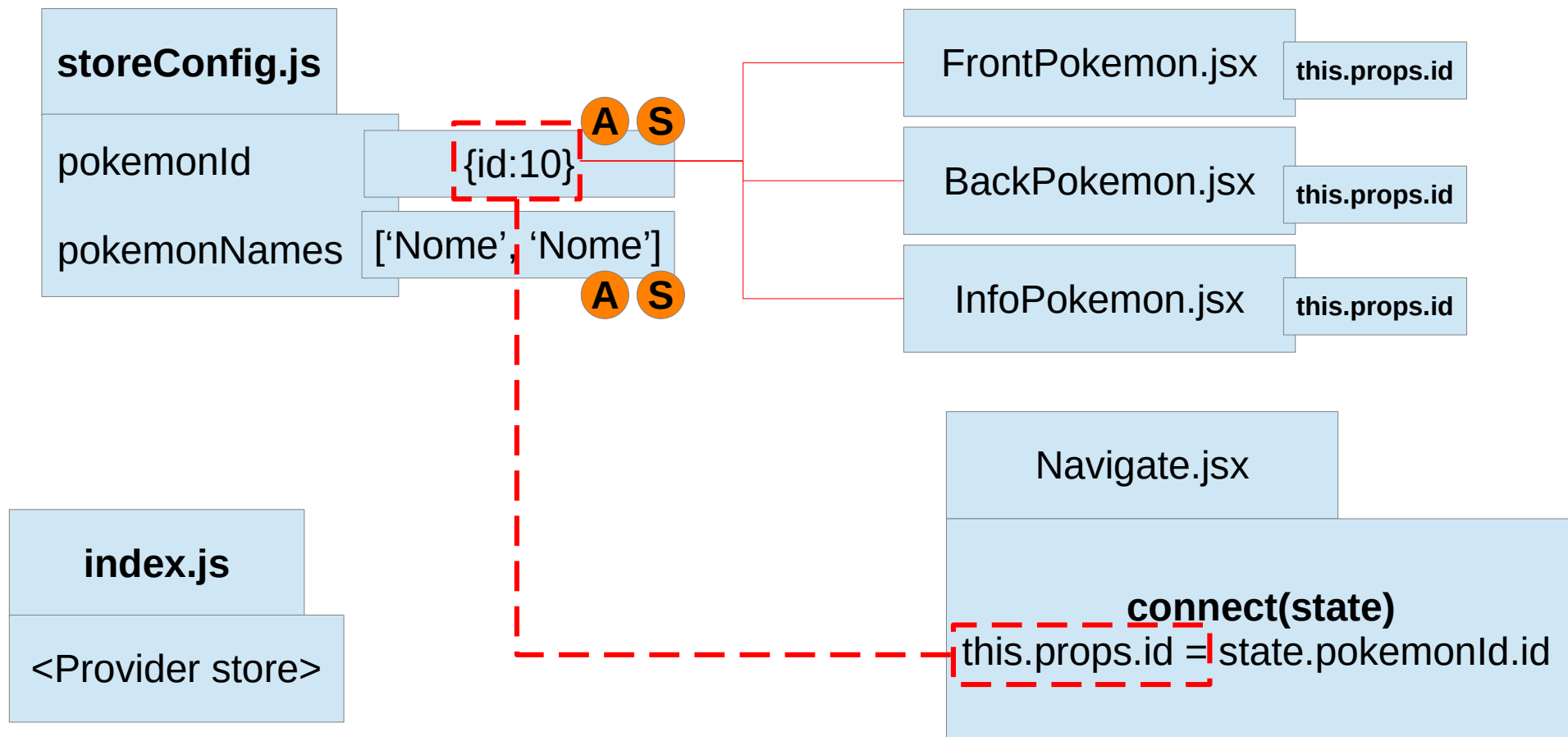
Store - Consumindo

```
function mapStateToProps(state) {  
  return {  
    id: state.pokemonId.id,  
  }  
}
```

- O **state**, parâmetro de mapStateToProps está conectado com **TODOS** os reducers de stateConfig.js.
- Logo, a partir dele, eu posso acessar:
 - state.pokemonId.id
 - state.pokemonNames[0] //por exemplo
- Dentro do componente Navigate, eu acesso a variável **id**, via props, inicializada no retorno de mapStateToProps.



A S *Action e State, passados como parâmetros para cada reducer.*



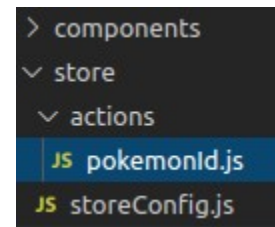
***id** do reducer **pokemonId** está conectado com **this.props.id** de **Navigate**, ou seja, quaisquer mudança no estado de **id**, refletirá em **this.props.id**. O mesmo vale para os outros componentes.*

Parte 2

ActionCreate e Action (Ações)

Action

- Até o momento, todos os componentes tem acesso ao estado geral (**reducers**).
- No entanto, queremos modificar o estado geral (no caso o id, em pokemonId) a partir de um **evento** disparado pelo **Navigate.jsx**.
- Sendo assim, devemos criar uma **ação (action)** que será chamada a partir de Navigate.jsx
- Dentro de **store**, crie o seguinte arquivo:



pokemonId.js

- **pokemonId.js** é um arquivo responsável em criar uma ação (**action**) que tem como objetivo trabalhar sobre o reducer **pokemonId**, em **storeConfig.js**.

```
export function alterarId(novold){  
  //retorna uma Action (objeto javascript)  
  return {  
    type: 'NOVO_ID',  
    payload: novold  
  }  
}
```

pokemonId.js

- A função **alterarId** recebe um parâmetro (**novoid**) que é o novo valor de id a ser atualizado em storeConfig.js. Note que essa função retorna um objeto javascript (uma **action**), formado por:
 - **type**: um nome único para essa ação, o qual irá ajudar o store decidir qual **reducer** disparar.
 - **payload**: o dado atualizado a ser repassado para os reducers, em storeConfig.js.

Navigate.jsx

- Em Navigate.jsx, devemos conectar o nosso `actionCreator` “`alterarId(novold)`”, de `pokemonId.js` com o props do componente `Navigate` (assim como fizemos com o state).
- Lembrando que só fazemos esse procedimento aos componentes que disparam eventos, pois eles tem interesse em **modificar** o estado.

Navigate.jsx

```
import { alterarId } from '../store/actions/pokemonId'

class Navigate extends Component {

  constructor(props) {
    super(props)
    this.props.alterarPokemonId(311)
  }
  ...
}
...
function mapActionCreatorToProps(dispatch){
  return {
    alterarPokemonId(novold){
      const action = alterarId(novold)
      dispatch(action)
    }
  }
}
connect(mapStateToProps,mapActionCreatorToProps)(Navigate)
```

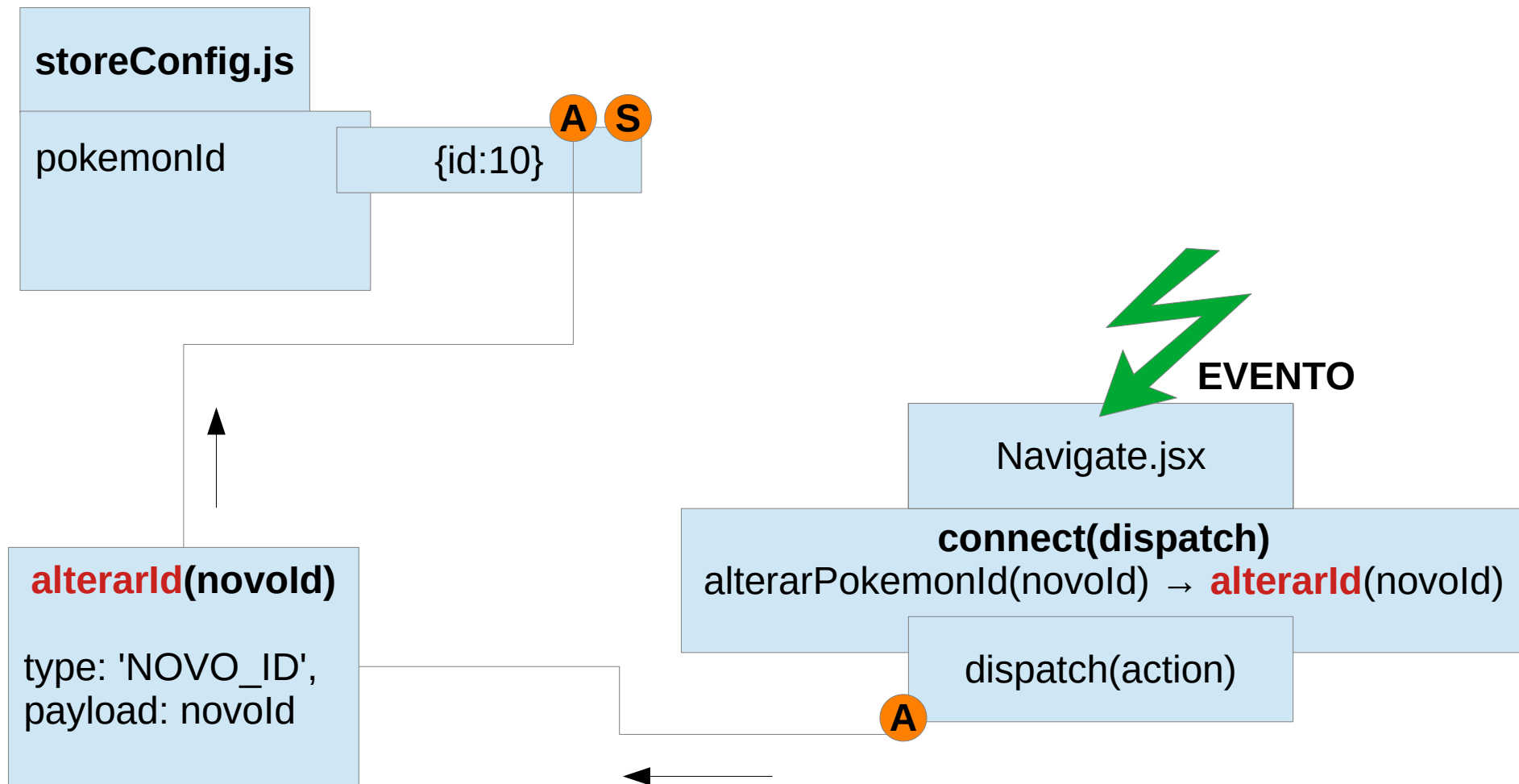

Navigate.jsx

- Antes de mais nada, importar a função de pokemonId.js em Navigate.jsx
 - `import { alterarId } from '../store/actions/pokemonId'`
- A função `mapActionCreatorToProps` irá retornar a função `alterarPokemonId(novold)` a qual, internamente, acessa a função `alterarId(novold)` de `pokemonId.js`.
- Finalmente, `mapActionCreatorToProps` é conectado ao `Navigate` com o seguinte comando:
 - `connect(mapStateToProps, mapActionCreatorToProps)(Navigate)`

Navigate.jsx

- De certa forma, podemos dizer agora que para o Navigate, a função `alterarPokemonId(novold)` está conectada com `alterarId(novold)`.
- Internamente, em Navigate, via props, é possível acessar `alterarPokemonId(novold)`

```
:
  constructor(props) {
    super(props)
    this.props.alterarPokemonId(311)
  }
```



Parte 3

Discriminando a action

Dentro dos reducers.

storeConfig.js

- Quando uma ação é disparada por um evento, TODOS os reducers são avisados.
- No entanto, nem todos estão INTERESSADOS nessa ação.
- Devemos criar mecanismos de discriminação da ação dentro de cada reducer.

storeConfig.js

```
const reducers = combineReducers({
  pokemonId: function(state, action){
    switch(action.type){
      case 'NOVO_ID':
        return {
          ...state,
          id: action.payload
        }
      default:
        return {
          id: 10
        }
    }
  }
})
```

action.type armazena a string que identifica a ação disparada por um evento.

action.payload armazena o valor a ser repassado para os componentes conectados com esse store.

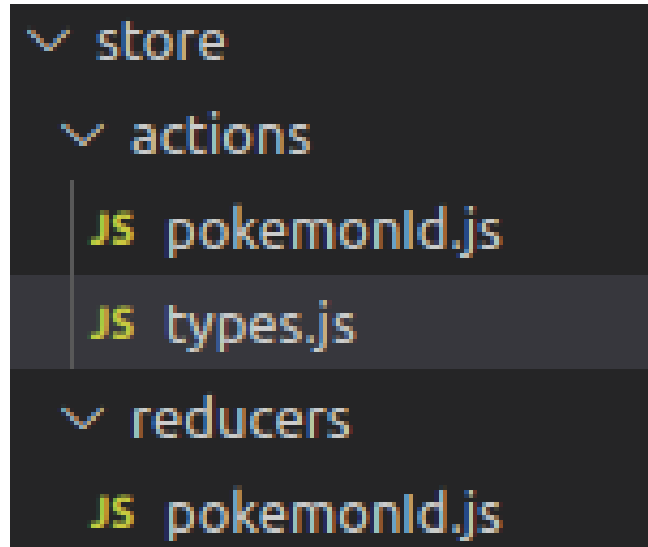
Navigate.jsx

- Agora está tudo conectado. Falta apenas chamar a função que gera a ação nos eventos corretos em Navigate.jsx, por exemplo:

```
...  
proximo() {  
  const novold = (this.props.id + 1 > 500) ? this.props.id : this.props.id + 1  
  this.props.alterarPokemonId(novold)  
  
}  
...
```

Refatoração

- Crie o arquivo types.js em actions
- Crie a pasta reducers dentro store e o arquivo pokemonId.js



Refatoração

- types.js

```
export const NOVO_ID = 'NOVO_ID'
```

- pokemonId.js dentro de actions

```
import {NOVO_ID} from './types'
```

```
export function alterarId(novold){  
  return {  
    type: NOVO_ID,  
    payload: novold  
  }  
}
```

Refatoração

- O arquivo pokemonId.js dentro de reducers:

```
import {NOVO_ID} from '../actions/types'

const initialState = {
  id:10
}

export default function(state = initialState, action){
  switch(action.type){
    case NOVO_ID:
      return {
        ...state,
        id: action.payload
      }
    default:
      return state
  }
}
```

Refatoração

- O arquivo storeConfig.js

```
import { createStore, combineReducers } from 'redux'  
import pokemonIdReducer from './reducers/pokemonId'
```

```
const reducers = combineReducers({  
  pokemonId: pokemonIdReducer  
})
```

```
function storeConfig(){  
  return createStore(reducers)  
}
```

```
export default storeConfig
```

Resultado

Exercício React-Redux Simples

ID do Pokémon

-10 Anterior Próximo +10 130

Frente



Costas



Informações do Pokémon 130

Gyarados

Altura: 65

Peso: 2350

Ordem: 194

