Redux

Criando uma Store — armazenamento disponibiulização

```
const store = createStore( reducer );
```

```
const store = createStore(
    rootReducer,
    composeWithDevTools()
);

const store = createStore(
    rootReducer,
    composeWithDevTools( applyMiddleware( thunk ) )
);


// Dica 1: O ReduxDevTools é uma ferramenta incrível
// para entender o comportamento de uma aplicação
// Redux

// Dica2: Podemos adicionar múltiplos middlewares na
// nossa aplicação redux, sendo redux-thunk um dos mais
// utilizados
```

Criando um reducer — escrita

```
const INITIAL_STATE = { name: '' };
const reducer = ( state=INITIAL_STATE, action ) => {
    switch( action.type ){
        case CHANGE_NAME:
            return { ...state, name: action.payload }
        default:
            return state;
    }
}

// Dica 1: Será considerado um REDUCER qualquer função
// que receba um state e uma action e retorne um novo
// state.

// Dica 2: Redux exige a IMUTABILIDADE do state. O que
// significa que não podemos mudar o state atual e, por
// isso, temos que criar um novo.
```

Combinando reducers — escrita

```
const rootReducer = combineReducers(
    { reducer1, reducer2 }
);


const rootReducer = combineRecucers(
    { user: userReducer, character: characterReducer }
);
```

Actions — escrita

```
{ type: 'CHANGE_NAME' }

{ type: 'CHANGE_NAME', payload: 'novo nome' }

const actionChangeName = {
    type: 'CHANGE_NAME',
    payload: 'novo nome'
};

// Dica: Será considerada uma ACTION qualquer objeto
// que tenha a chave type.

// Dica2: Podemos colocar qualquer outra propriedade
// no objeto ACTION, desde que não esqueçamos o type.

// Dica3: É comum usar a chave payload em actions. E
// ela irá conter todos os dados necessários para a
// ação
```

Action Creator — escrita

```
const changeName = () => ({ type: 'CHANGE_NAME' });


const changeName = (name) => {
    return { type: 'CHANGE_NAME', payload: name };
}
```

Dispatch — escrita

```
store.dispatch( { type: CHANGE_NAME } );

store.dispatch( changeName( 'New Name' ) );

// Dica 1: O dispatch é uma operação síncrona e por
// isso proíbe o uso de side effects no reducer

// Dica 2: Para efetuar dispatches de operações
// assíncronas, como chamadas em apis, podemos esperar
// o resultado da operação assíncrona e usar o dispatch
// de forma síncrona

// Dica 3: Outra forma de efetuar um dispatch de uma
// operação assíncrona é utilizando o middleware
// redux-thunk
```

getState — leitura

```
store.getState();
```

Subscribe — leitura

```
store.subscribe( () => {
    const newStateWithChanges = store.getState();
});


// Dica1: O subscribe é a função que será executada
// sempre que o state mudar.
```

React-Redux

Provider — disponibilização

```
import { Provider } from 'react-redux';

<Provider store={store}>
    <App />
</Provider>

// Dica1: O Provider é obrigatório e irá disponibilizar
// o uso do redux para a hierarquia abaixo de si
```

Connect — leitura

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


export default connect()( MyComponent );

export default connect( mapStateToProps )( MyComponent );

export default connect(
    mapStateToProps,
    mapDispatchToProps
)( MyComponent );


// Dica 1: O connect vai trabalhar por debaixo dos
// panos com o subscribe. Por isso nós não precisamos
// usar o subscribe quando usamos o read-redux

// Dica 2: O connect vai adicionar no props do seu
// componente tudo o que estiver no mapStateToProps
// e o que tiver no seu mapDispatchToProps.
// Podemos acessar os dados dentro do this.props

// Dica 3: quando não passamos o mapDispatchToProps
// o connect vai passar todo o mapStateToProps
// e adicionar dispatch como atributo na prop.
// Podemos acessar ele com um this.props.dispatch
```

mapStateToProps — leitura

```
const mapStateToProps = ( state ) => ({
    name: state.userReducer.name
});

const mapStateToProps = ( state ) => {
    return {
        name: state.userReducer.name
    };
};
```

mapDispatchToProps — escrita

```
// Dica 1: Precisamos importar nossos ActionCreators
import { changeName } from './store/actions'

const mapDispatchToProps = (dispatch) => ({
    dispatchChangeName: () => {
        dispatch( changeName() );
    }
});
```

Async

Redux Thunk — escrita

```
import thunk from 'redux-thunk';

const store = createStore(
    rootReducer,
    composeWithDevTools( applyMiddleware( thunk ) )
);

// Dica 1: Para usar o Thunk basta adicionar como
// middleware
```

Action Thunk — escrita

```
const changeCharacters = {
    type: 'CHANGE_CHARACTERS'
}

const getCharacters = async () => {
    const apiCharacters = await fetch('http://myapi');
    const characters = await apiCharacters.json();
    return ( dispatch ) => {
        dispatch( changeCharacters( characters ) )
    }
};

// Dica 1: Ações thunk são funções que retornam
// uma função onde o primeiro parâmetro é o dispatch.
// Com ele podemos fazer uma chamada normal do dispatch
// de uma action síncrona
```