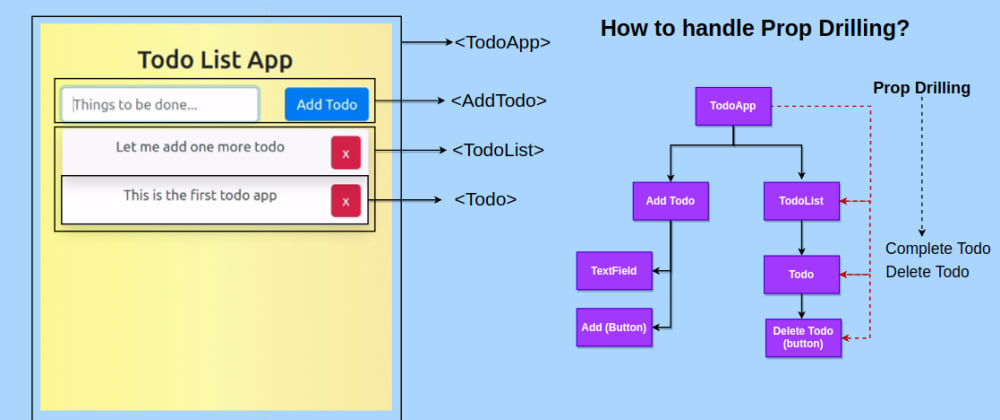
**04/12 – COMO FUNCIONA O REDUX? (REACT)**

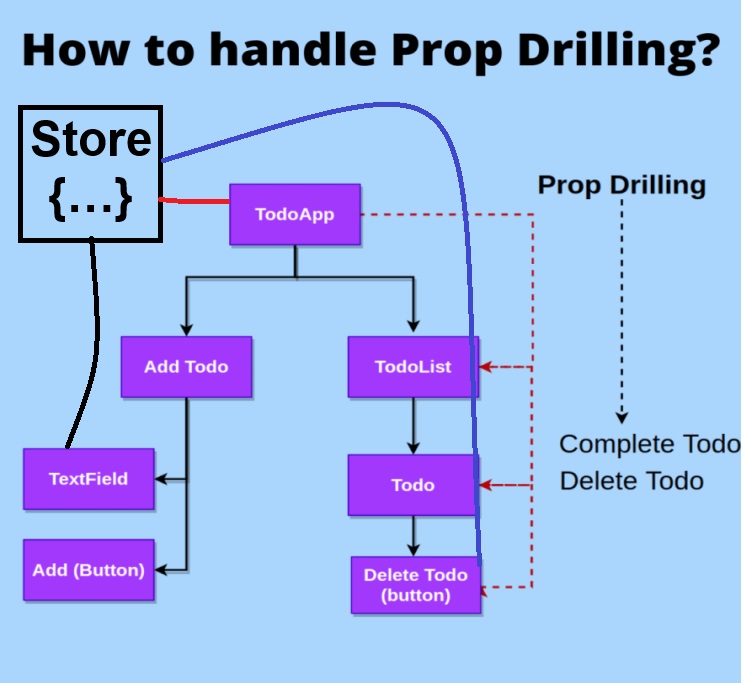
O Redux é uma biblioteca de **gerenciamento de estado** no React. Ele permite o gerenciamento do estado da aplicação de maneira centralizada e previsível. Veja a seguir alguns conceitos importantes para a compreensão do Redux:

**Problema:** Quando uma aplicação fica muito grande, você acaba tendo uma **árvore de componentes** maior. Com isso, fica difícil de transmitir dados de um componente para outro, principalmente se você deve envolver outros componentes nessa comunicação de maneira desnecessária. Isso é chamado de **prop drilling.**



Imagine a situação acima. Em um contexto de desenvolvimento de uma lista de tarefas, como faríamos para passar uma função de deletar tarefa, que está no componente TodoApp, para o botão? Nesse caso, faríamos uma prop drilling, usando os componentes TodoList e Todo como intermediários, para que o componente de botão receba a função do componente TodoApp. Isso complica o desenvolvimento, já imaginou isso em uma escala maior ainda?

**Solução:** o Redux entra nesse cenário por meio do **store**, um objeto descentralizado de estados. Com isso, cada componente pode realizar uma comunicação com o store de maneira individual e obter os valores do estado, sem a necessidade de se comunicar com outros.



**Fluxo do Redux:**

**1. Evento:** um evento é disparado por alguma ação do usuário. (exemplo: clique de um botão).

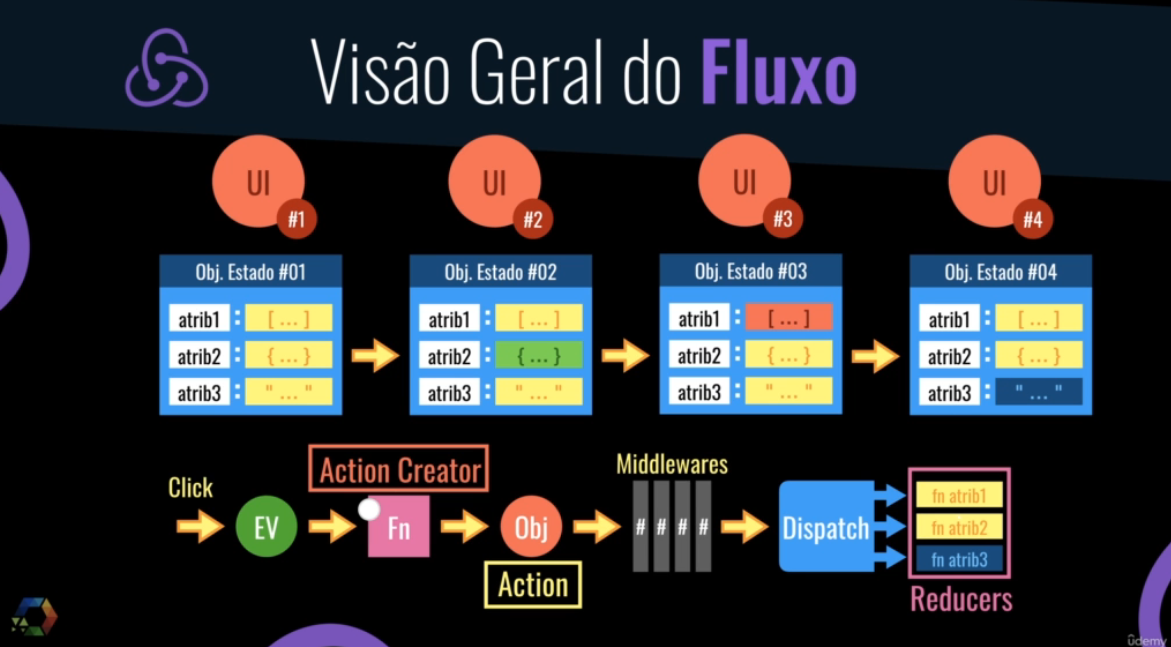
**2. Action Creator:** é executada uma função que retorna um objeto, comumente chamado de **action**. Esse objeto possui pelo menos 1 atributo chamado **type**, que indica o tipo da ação que acabou de ser realizada.

**3. Dispatch:** assim que esse objeto é criado, ele é despachado pela função **dispatch.** Isso notifica o Redux de que ocorreu uma ação.

**4. Reducers:** a execução do Dispatch aciona os reducers. Eles são funções que especificam como o estado deve ser alterado em resposta a uma ação. Essas funções recebem como parâmetro o estado anterior e o objeto da ação (action). Com isso, verificam a ação que foi realizada, e com base nela, retornam um novo estado.

**5. Store:** o estado é alterado no store. O store modifica automaticamente todos os componentes interessados sobre a mudança no estado.

**6. Componentes atualizados:** os componentes que dependem do estado modificado são automaticamente atualizados com os novos dados, refletindo a mudança na interface do usuário.



**RESUMO**

De maneira simples, fazendo o uso do Redux, todos os estados ficarão armazenados em uma árvore de objetos através do store. Para que isso aconteça, o Redux utiliza 3 recursos:

**Store:** um container ou um grande centro de informações, que tem disponibilidade para receber e entregar o que o seu componente requisita. A store armazena de forma centralizada todos os estados da aplicação. Vale ressaltar que a store é imutável.

**Actions:** São objetos que representam ações disparadas da aplicação para o store. Elas são criadas através das action creators. As actions são a única forma de acionar uma mudança de estados no store.

**Reducers:** Cada dado da store deve ter o seu próprio reducer. Ele é encarregado de lidar com todas as ações e especificar como o estado da aplicação irá mudar de acordo com a action que foi enviada para o store.

O fluxo geralmente funciona da seguinte forma: um componente gera uma interação através de um clique dado pelo usuário na interface - por exemplo, assim, um action creator é acionado e dispara uma ação para o store. Essa ação chega até um reducer que irá processar e fazer a alteração do estado no store. Assim um novo estado será disponibilizado para o componente.

**EXEMPLO DE INTEGRAÇÃO DO REDUX EM UM APP**

Para exemplo de como integrar o Redux em uma aplicação, o professor apresentou o seguinte modelo.



Trata-se de um programa que recebe dois números em um componente: mínimo e máximo. A partir disso, outros três componentes consomem o valor desses números para fazer alguma lógica com eles. O primeiro retorna a **média** dos dois valores, o segundo os **soma** e o terceiro **sorteia** um número aleatório entre esses números.

Essa aplicação é bem simples e ótima para quem está iniciando. Sem Redux, o desenvolvimento é bem simples. Basta seguir os passos abaixo:

**1.** Criar duas variáveis no componente App com useState(), representando os dois números.

**2.** Passá-las como props para todos os componentes. No componente Intervalo, além das variáveis, é preciso também passar as funções que as alteram.

function App() {

  const [min, setMin] = useState(1)

  const [max, setMax] = useState(10)

  return (

    <div *className*="App">

      <h1>Exercício React + Redux</h1>

      <div *className*="linha">

        <*Intervalo* *max*={max} *min*={min} *setMax*={setMax} *setMin*={setMin} />

      </div>

      <div *className*="linha">

        <*Média* *max*={max} *min*={min}></*Média*>

        <*Soma* *max*={max} *min*={min}></*Soma*>

        <*Sorteio* *max*={max} *min*={min}></*Sorteio*>

      </div>

    </div>

  );

}

**3.** No componente Intervalo, aplicar a técnica de controlled inputs, usando o atributo value (variável) e evento onChange (função que altera o valor da variável) dos inputs.

<strong>Mínimo:</strong>

<input *type*="number" *value*={*min*} *onChange*={(*e*) => setMin(*e*.target.value)} />

<strong>Máximo:</strong>

<input *type*="number" *value*={*max*} *onChange*={(*e*) => setMax(*e*.target.value)} />

**4.** Realizar a lógica de cada componente.

Agora, para aplicar o Redux nisso, precisaremos seguir passos mais complexos, os quais estão listados a seguir:

**1. Criar a store**

Para isso, precisaremos criar uma pasta chamada **store** e um arquivo com o nome **storeConfig.js.** Dentro desse arquivo, iremos usar as funções **createStore()** e **combineReducers()** para criar a store.

import {createStore, combineReducers} from 'redux'

import numerosReducer from './reducers/numerosReducer'

const reducers = combineReducers({

   numeros: numerosReducer

})

function storeConfig(){

    return createStore(reducers)

}

export default storeConfig

**2. Criar reducers**

Os reducers especificam como os estados são modificados, de acordo com as ações que ocorrem. Precisamos criar esses reducers primeiro, afinal eles estão inclusos no passo anterior.

Para isso, criamos uma pasta dentro de store chamada **reducers** e ali, incluímos reducers para cada parte da store. Como nossa aplicação envolve números, criei apenas um arquivo com o nome de **numerosReducer.**

Nele, deve haver um **initialState**, que representa um objeto com os valores iniciais do estados.

A seguir, haverá o reducer: uma função que recebe **state** (estado atual) e **action** (objeto gerado com um evento que possui o atributo type, com o tipo da ação). Deve ser feito um switch em **action.type** e para cada ação, deverá ser retornado um novo objeto, com spread do estado atual + o que será mudado.

const initialState = {

    min: 10,

    max: 100

}

export default function numerosReducer(*state* = initialState, *action*){

    switch(*action*.type){

        case MIN\_ALTERADO:

            return {

                ...*state*,

                min: *action*.payload

            }

        case MAX\_ALTERADO:

            return {

                ...*state*,

                max: *action*.payload

            }

        default:

            return *state*

    }

}

**3. Criar Action Creators**

Após termos criados reducers que leem o atributo type do objeto action, precisamos criar funções que geram esses objetos. Ao final, essas funções devem ser exportadas/ Veja como é simples:

function alterarMin(*newMin*){

    return {

        type: MIN\_ALTERADO,

        payload: +*newMin*

    }

}

function alterarMax(*newMax*){

    return {

        type: MAX\_ALTERADO,

        payload: +*newMax*

    }

}

export {

    alterarMax, alterarMin

}

**4. Usar a função connect()e mapear o estado da store como props para cada componente**

Nesse estágio, devemos criar funções em cada componente que convertam o estado da store como props dos componentes, desse jeito:

**Componentes somente leitura**

function mapStateToProps(*state*){

    return{

        min: *state*.numeros.min,

        max: *state*.numeros.max

    }

}

**Componentes que alterem o estado**

function mapDispatchToProps(*dispatch*){

    return {

        alterarMaximo(*newMax*){

            dispatch(alterarMax(*newMax*))

        },

        alterarMinimo(*newMin*){

            dispatch(alterarMin(*newMin*))

        }

    }

}

Ao final, você deve importar a função connect() de redux e usá-la em todos os componentes, dessa maneira.

export default connect(mapStateToProps, mapDispatchToProps)(Intervalo)

**5. Receber as props no componente e aplica-las**

Após mapearmos as props, podemos recebe-las dentro do objeto **props** do componente e então fazer uso delas, como mostrado a seguir:

function Intervalo({*min*, *max*, *alterarMaximo*, *alterarMinimo*}){

    return(

    <*Card* *title*="Intervalo de números" *red*>

       <div *className*="intervalo">

           <span>

               <strong>Mínimo:</strong>

               <input *type*="number" *value*={*min*} *onChange*={(*e*) => alterarMinimo(*e*.target.value)} />

           </span>

           <span>

              <strong>Máximo:</strong>

               <input *type*="number" *value*={*max*} *onChange*={(*e*) => alterarMaximo(*e*.target.value)} />

           </span>

       </div>

    </*Card*>

    )

}