ReactJS
Redux

1 Introdução

A documentação oficial do Redux o define como

"Um contêiner de estado previsível para aplicações Javascript"

Veja o Link 1.1.

Link 1.1
https://redux.js.org/

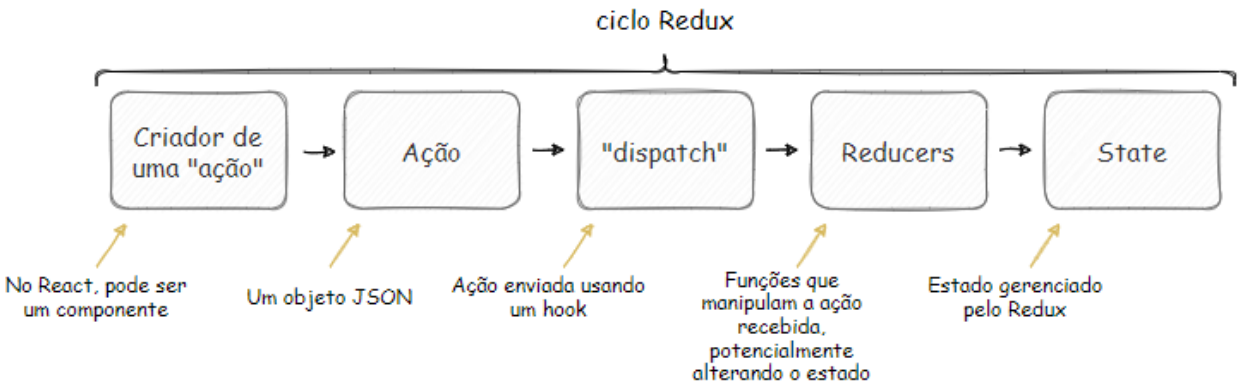
Vale destacar os seguintes pontos da documentação.

- Trata-se de uma biblioteca para manipulação de **estado centralizado**.

- Embora seja muito utilizado em aplicações React, não foi produzido com esse único propósito.

1.1 (Ciclo básico do Redux) O Redux possui um ciclo composto por algumas partes que levam um nome um tanto específico. Veja a Figura 1.1.1.

Figura 1.1.1

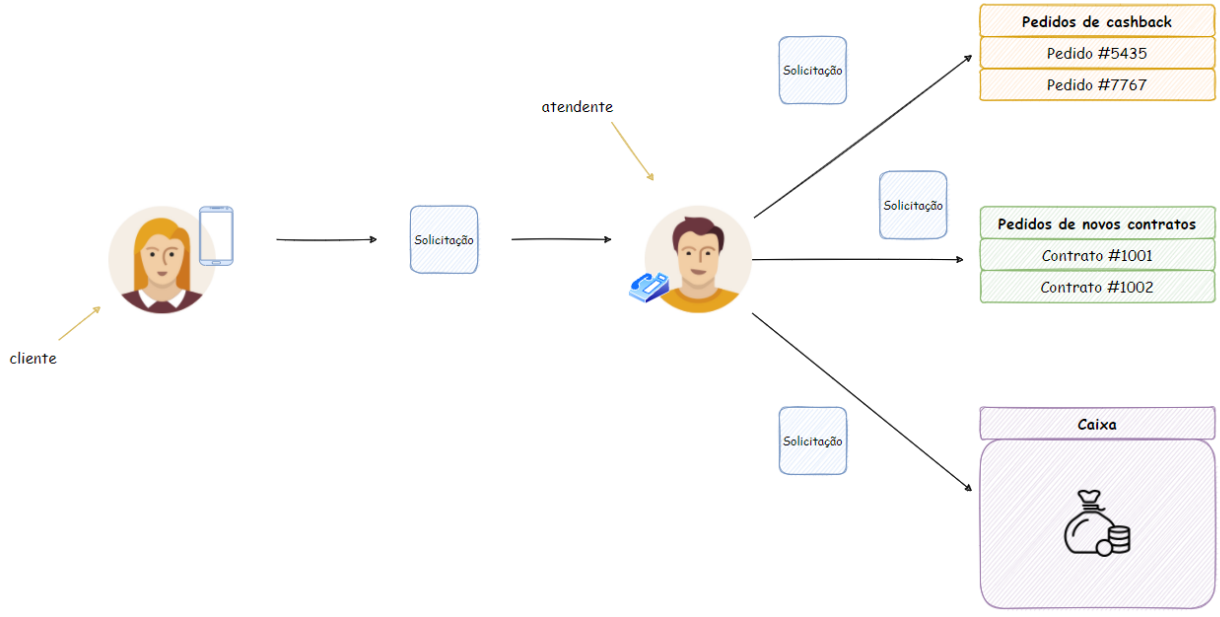


1.2 (Cashback oferecido por uma empresa de cartões de crédito: uma analogia) Para simplificar o entendimento de cada item do ciclo ilustrado, considere a seguinte analogia.

I. Uma empresa de cartão de crédito oferece "**cashback**" para compras feitas utilizando seus cartões.
II. Pessoas podem adquirir seus cartões de crédito. Para isso, assinam um **contrato** com a empresa. Elas pagam uma **taxa** por isso.
III. Após acumular uma quantidade de compras, os clientes podem realizar **pedidos** para obter seu cashback.
IV. A empresa paga os pedidos em **dinheiro**.
V. A empresa armazena o **histórico de pedidos de cashback**.
VI. A empresa armazena o **histórico de contratos assinados**.
VII. A empresa possui um **caixa** a partir do qual os pagamentos acontecem.

Veja a Figura 1.2.1.

Figura 1.2.1

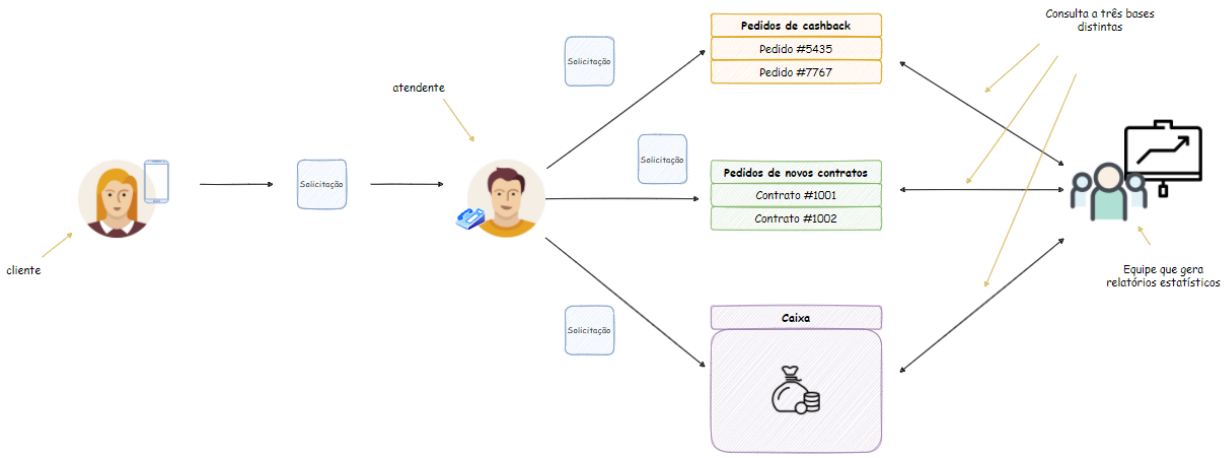


Na Figura 1.2.1, o que ocorre é o seguinte:

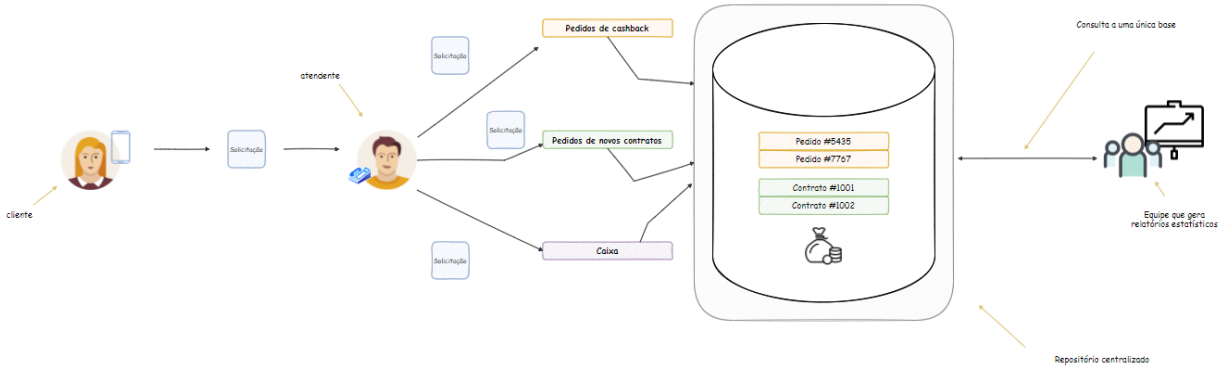
I. Um cliente tem uma solicitação. Ela pode ser de tipos diversos.
II. O cliente entrega a sua solicitação a um atendente.
III. O atendente entrega uma cópia da solicitação **para cada** departamento da empresa.
IV. Cada departamento que recebe a solicitação decide como manipulá-la.
V. Dependendo do tipo da solicitação, um departamento pode decidir ignorá-la. Por exemplo, o departamento de pedidos de cashback não está interessado em solicitações de novos contratos.

1.3 (Uma equipe deseja desenvolver relatórios estatísticos) Suponha que há uma equipe nesta empresa que deseja desenvolver relatórios estatísticos sobre os contratos, pedidos de cashback e fluxo de caixa. Para tal, seria necessário consultar as bases de cada um dos departamentos a fim de obter os dados de interesse. Veja a Figura 1.3.1

Figura 1.3.1



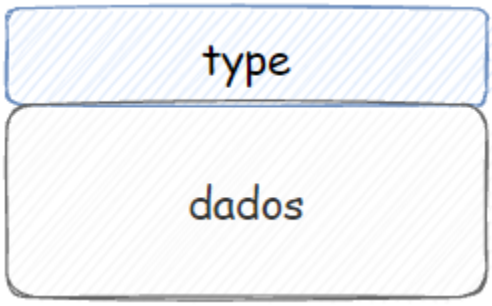
A Figura 1.3.2 mostra uma alternativa: armazenar os dados de todos os departamentos em um único **repositório centralizado**.



1.4 (Com o que se parece uma solicitação?) Neste exemplo, teremos três tipos de solicitações:

- Criação de novo contrato. Vamos supor que há uma taxa inicial a ser paga pelo contrato. Além disso, a pessoa interessada deve informar o seu nome.
- Pedido de cashback. Uma pessoa informa seu nome e o valor que deseja obter como cashback.
- Cancelamento de contrato. Uma pessoa pode cancelar seu contrato a qualquer momento. Para tal, ela informa seu nome. A Figura 1.4.1 mostra a estrutura básica de uma solicitação.

Figura 1.4.1



Nota. O nome "type" é obrigatório. As demais partes são decididas pelo desenvolvedor. Veja o que diz a documentação.

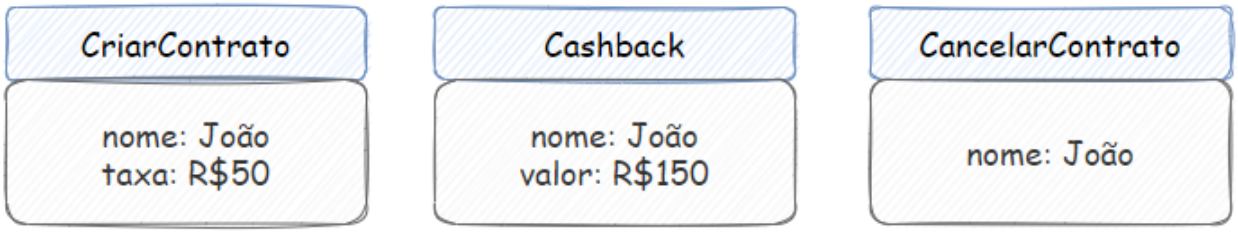
"Actions must have a 'type' field that indicates the type of action being performed. Types can be defined as constants and imported from another module. It's better to use strings for type than Symbols because strings are serializable. Other than type, the structure of an action object is really up to you."

Veja mais na página acessível por meio do Link 1.4.1.

Link 1.4.1
https://redux.js.org/api/store

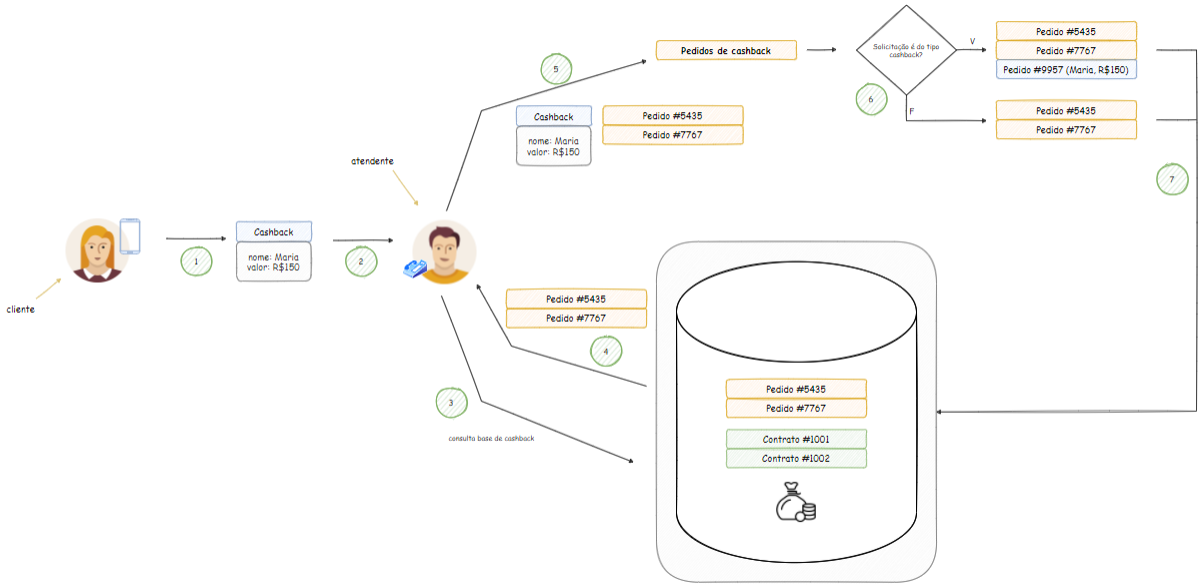
A Figura 1.4.2 mostra os três tipos de solicitações que nos são de interesse.

Figura 1.4.2



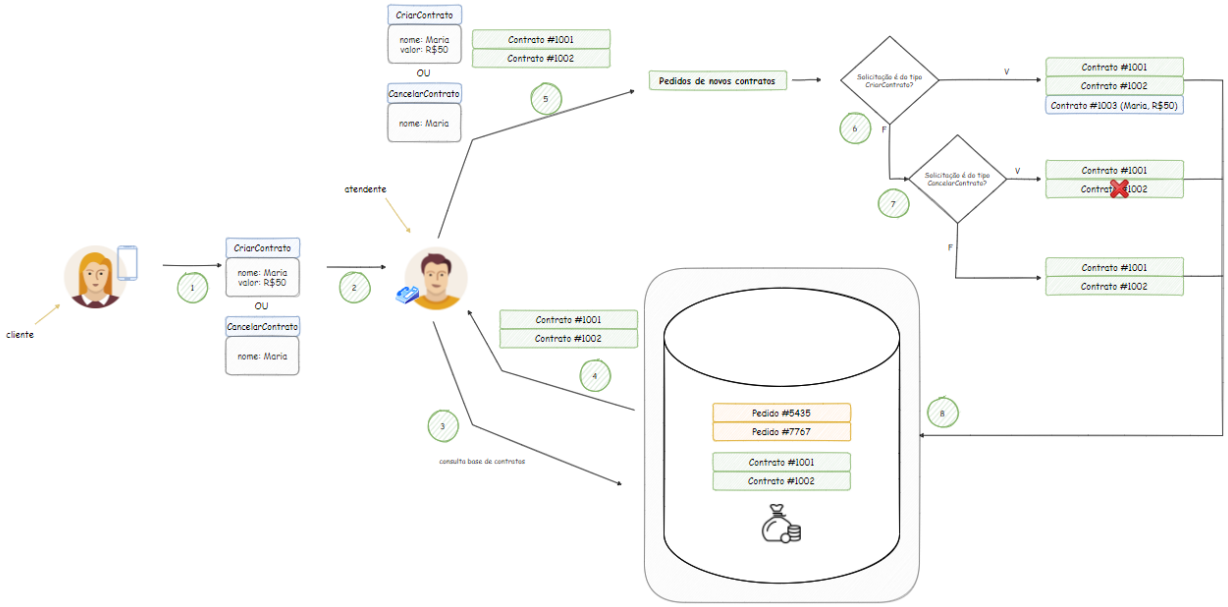
1.5 (Detalhes sobre um pedido de cashback: funções do atendente e o departamento de "Pedidos de cashback") Como vimos, cabe ao atendente receber as solicitações e direcioná-las ao departamento adequado. Dada a existência da base centralizada, caberá a ele, também, consultá-la previamente e entregá-la ao departamento alvo da solicitação. Por exemplo, quando ele recebe uma solicitação de cashback, ele faz uma consulta à base e obtém todos os registros referentes a cashback. Depois disso, entrega a lista de registros de cashback e a nova solicitação ao departamento de pedidos de cashback que, por sua vez, decide se a base centralizada deve ou não ser atualizada. Veja a Figura 1.5.1.

Figura 1.5.1


cliente
Cashback
nome: Maria
valor: R$150
1
2
atendente
3
consulta base de cashback
Pedido #5435
Pedido #7767
4
5
Pedidos de cashback
Cashback
nome: Maria
valor: R$150
Pedido #5435
Pedido #7767
Solicitação é do tipo cashback?
V
F
6
Pedido #5435
Pedido #7767
Pedido #9957 (Maria, R$150)
Pedido #5435
Pedido #7767
7
Pedido #5435
Pedido #7767
Contrato #1001
Contrato #1002

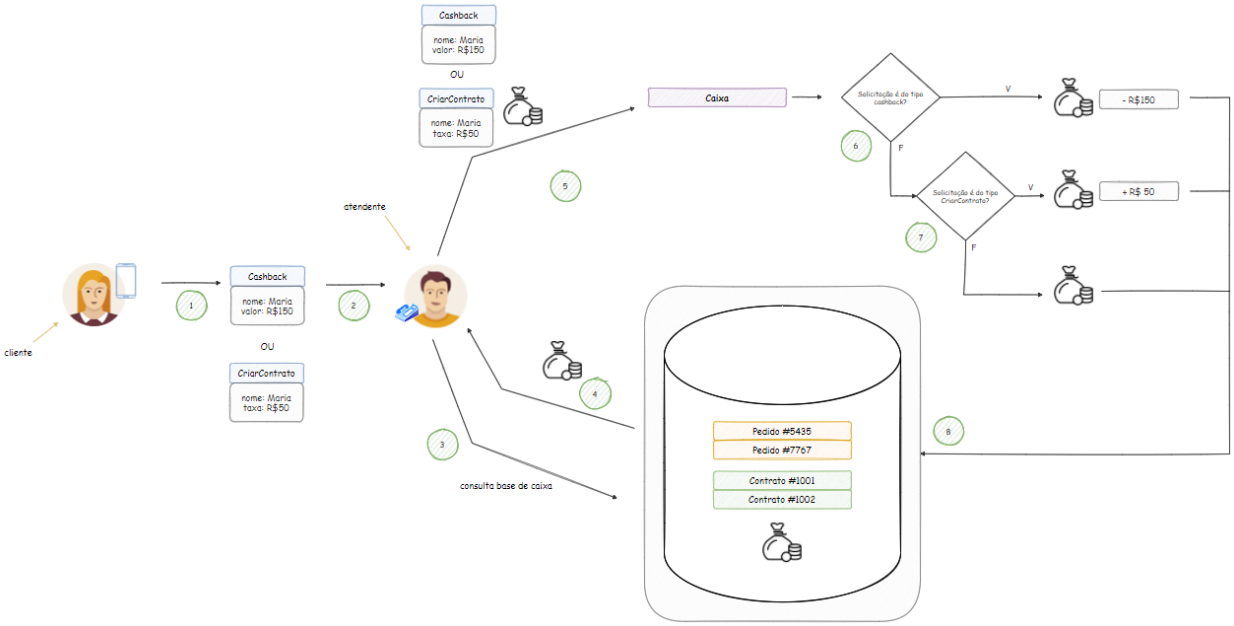
1.6 (Detalhes sobre um pedido de novo contrato: funções do atendente e o departamento de "contratos") Ao receber solicitações dos tipos NovoContrato ou CancelarContrato, o atendente consulta a base de contratos existentes e a entrega ao departamento de novos contratos, incluindo a nova solicitação. Veja a Figura 1.6.1.

Figura 1.6.1



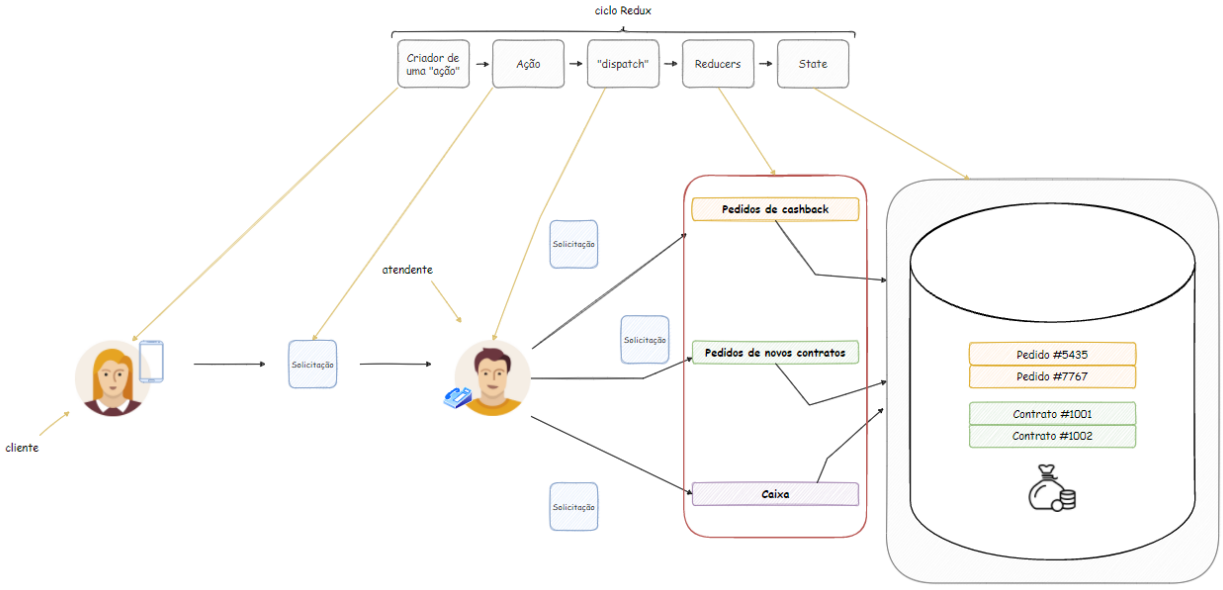
1.7 (Detalhes de um pedido por cashback ou novo contrato: funções do atendente e o departamento "caixa") Pedidos de novos contratos e por cashback também são de interesse para o departamento "caixa". Veja a Figura 1.7.1.

Figura 1.7.1



1.8 (Associando os nomes do ciclo Redux com as partes retratadas na simulação da empresa de cashback) Cada parte retratada nas simulações feitas até então está associada a um conceito do ciclo Redux. Veja a Figura 1.8.1.

Figura 1.8.1



2 Desenvolvimento

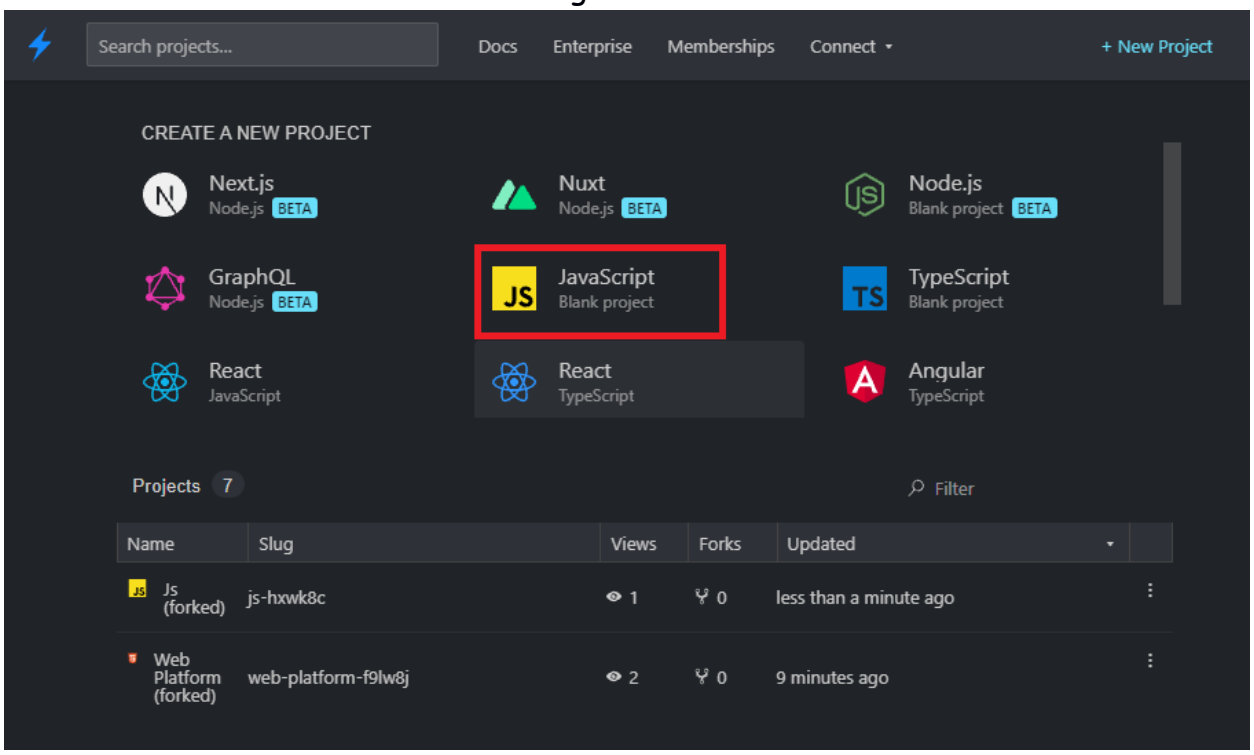
Nesta seção, implementaremos as funcionalidades da empresa descrita. Para tal, utilizaremos o ambiente StackBlitz. Encontre a sua página oficial por meio do Link 2.1.

Link 2.1
https://stackblitz.com/

Pode ser de interesse fazer login com o seu Github, assim você poderá armazenar seus projetos.

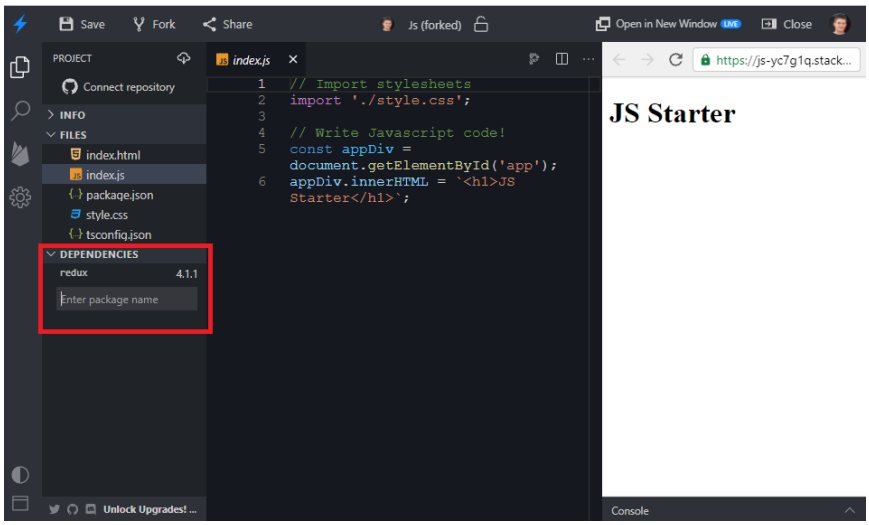
2.1 (Novo projeto Javascript) Na tela inicial do StackBlitz, clique **Javascript – Blank Project**, como destaca a Figura 2.1.1.

Figura 2.1.1



2.2 (Adicionando o Redux como dependência) O Redux é uma biblioteca Javascript e precisa ser adicionado como dependência do projeto. Para tal, clique sob "DEPENDENCIES', digite "redux" e aperte Enter. Veja a Figura 2.2.1.

Figura 2.2.1



2.3 (Um criador de ação para a criação de contratos) Uma das solicitações previstas no sistema é a criação de contratos. Isso é responsabilidade de um criador de ações que, neste caso, será uma simples função. Ela recebe nome e valor da taxa de criação de contrato e devolve uma **ação**. A ação é um simples objeto JSON contendo tipo e os dados de interesse. Veja o Bloco de Código 2.3.1. Estamos no arquivo **index.js**. Caso ele possua algum conteúdo inicial, apague tudo.

Nota: Lembre-se de clicar **Save** esporadicamente no canto superior esquerdo da tela.

Bloco de Código 2.3.1

```
//essa função é criadora de um tipo de ação
const criarContrato = (nome, taxa) => {
 //esse JSON que ela devolve é uma ação
 return {
  type: "CRIAR_CONTRATO",
  dados: {
   nome, taxa
  }
 }
}
```

2.4 (Um criador de ação para o cancelamento de contratos) O Bloco de Código mostra uma função que desempenha o papel de criadora de ações. Ela cria ações para o cancelamento de contratos.

Bloco de Código 2.4.1

```
//esta função é criadora de um tipo de ação
const cancelarContrato = (nome) => {
 //esse JSON que ela devolve é uma ação
 return {
  type: "CANCELAR_CONTRATO",
  dados: {
   nome
  }
 }
}
```

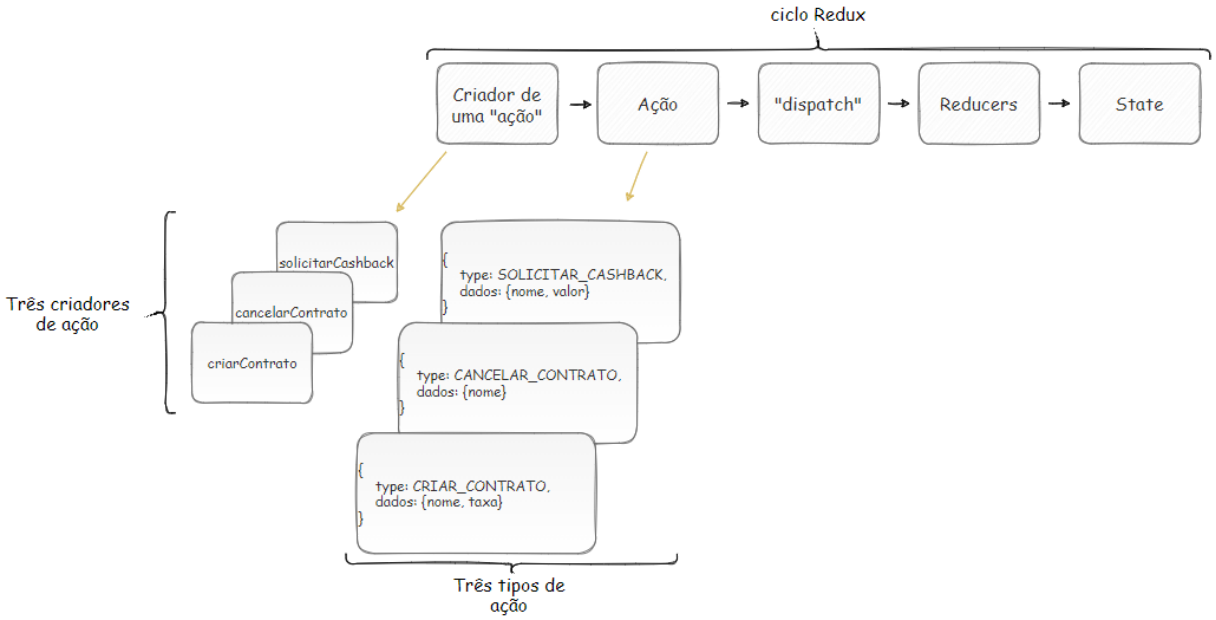
2.5 (Um criador de ações para solicitações de cashback) Por sua vez, a função exibida pelo Bloco de Código 2.5.1 cria ações para solicitações de cashback.

Bloco de Código 2.5.1

```
//esta função é criadora de um tipo de ação
const solicitarCashback = (nome, valor) => {
 //esse JSON que ela devolve é uma ação
 return {
  type: "CASHBACK",
  dados: {
   nome, valor
  }
 }
}
```

2.6 (Partes do ciclo Redux implementadas até então) A Figura 2.6.1 mostra as partes do ciclo Redux que implementamos até então.

Figura 2.6.1



2.7 (Um reducer para o tratamento de solicitações de cashback) Um reducer é uma simples função que recebe partes do estado atual que lhe sejam de interesse e a ação a ser tratada. Cabe a ela devolver o estado atualizado de acordo com os parâmetros recebidos. O reducer para o tratamento de solicitações de cashback aparece no Bloco de Código 2.7.1. Repare que ele representa o departamento da empresa responsável por essa atividade.

Bloco de Código 2.7.1

```
//esta função é um reducer
//quando chamada pela primeira vez, seu primeiro parâmetro será undefined
//já que não existirá histórico algum
//por isso, configuramos uma lista vazia como seu valor padrão
const historicoDePedidosDeCashback = (historicoDePedidosDeCashbackAtual = [], acao) => {
 //se a ação for CASHBACK, adicionamos o novo pedido à coleção existente
 if (acao.type === 'CASHBACK'){
  //uma cópia. Contém todos os existentes + o novo
  //não faça push
  return [
   ...historicoDePedidosDeCashbackAtual,
   acao.dados
  ]
 }
 //caso contrário, apenas ignoramos e devolvemos a coleção inalterada
 return historicoDePedidosDeCashbackAtual
}
```

2.8 (Um reducer para a manipulação do caixa) O reducer do Bloco de Código 2.8.1 implementa a lógica do departamento de caixa. Ele recebe o valor existente no caixa e o altera de acordo com o tipo da ação recebida.

Bloco de Código 2.8.1

```
//caixa começa zerado
const caixa = (dinheiroEmCaixa = 0, acao) => {
 if (acao.type === "CASHBACK"){
  dinheiroEmCaixa -= acao.dados.valor
 }
 else if (acao.type === "CRIAR_CONTRATO"){
  dinheiroEmCaixa += acao.dados.taxa
 }
 return dinheiroEmCaixa
}
```

2.9 (Um reducer para a criação e cancelamento de contratos) O reducer – uma simples função, lembra? – do Bloco de Código 2.9.1 trata a criação e cancelamento de contratos.

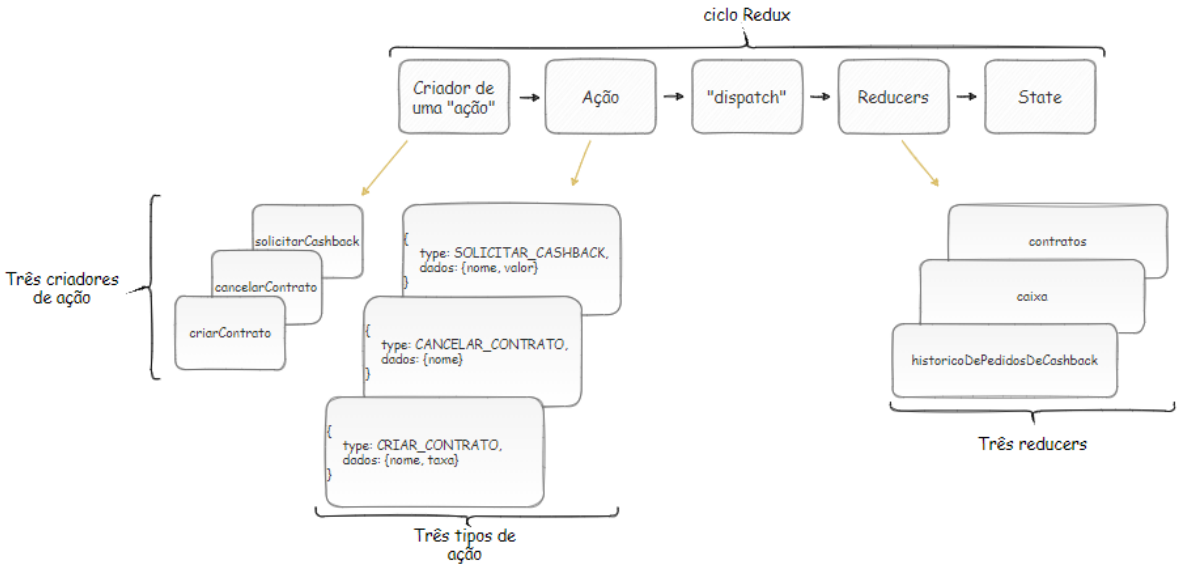
Bloco de Código 2.9.1

```
//lista começa vazia
```

```
const contratos = (listaDeContratosAtual = [], acao) => {
  if (acao.type === "CRIAR_CONTRATO")
    return [...listaDeContratosAtual, acao.dados]
  if (acao.type === "CANCELAR_CONTRATO")
    return listaDeContratosAtual.filter(c => c.nome !== acao.dados.nome)
  return listaDeContratosAtual
}
```

2.10 (Partes do ciclo Redux implementadas até então) A Figura 2.10.1 atualiza as partes do ciclo Redux que implementamos até então. Temos agora os reducers, veja.

Figura 2.10.1



2.11 (Utilizando o Redux) Até o momento sequer fizemos uso do Redux. Escrevemos apenas algumas funções Javascript comuns. Passaremos a utilizá-lo nesta seção. Quando utilizamos o Redux, combinamos coleções de criações de ações com reducers apropriados, obtendo um maquinário capaz de desempenhar o gerenciamento do estado centralizado. O primeiro passo é trazer o Redux para o contexto, como mostra o Bloco de Código 2.11.1. Você pode importá-lo na primeira linha de código do arquivo **index.js**, antes de definir qualquer função.

Bloco de Código 2.11.1

```
const Redux = require ('redux')
...
```

A seguir, após a definição de todas as funções, desestruturamos o objeto Redux a fim de obter as funções **createStore** e **combineReducers**. Veja o Bloco de Código 2.11.2.

Bloco de Código 2.11.2

```
...
//depois da definição de todas as funções
const { createStore, combineReducers } = Redux
```

O próximo passo é combinar todos os reducers utilizando a função **combineReducers**. Veja o Bloco de Código 2.11.3.

Bloco de Código 2.11.3

```
//depois da definição de todas as funções
const { createStore, combineReducers } = Redux

const todosOsReducers = combineReducers({
  historicoDePedidosDeCashback,
  caixa,
  contratos
})
```

Nota: O nome de cada "parte" do estado gerenciado pelo reducer será determinado pela chave escolhida quando **combineReducers** foi chamada. Neste caso, elas serão

historicoDePedidosDeCashback, **caixa** e **contratos**. Se desejar trocar o nome, basta escolher outro nome. Veja o exemplo do Bloco de Código 2.11.4.

Bloco de Código 2.11.4 – Apenas um exemplo

```
//depois da definição de todas as funções
const { createStore, combineReducers } = Redux

const todosOsReducers = combineReducers({
  historicoCashback: historicoDePedidosDeCashback,
  nossocaixa: caixa,
  osContratos: contratos
})
```

A seguir, construímos o chamado **store** do Redux, utilizando a função intuitivamente denominada **createStore**. Veja o Bloco de Código 2.11.5.

Bloco de Código 2.11.5

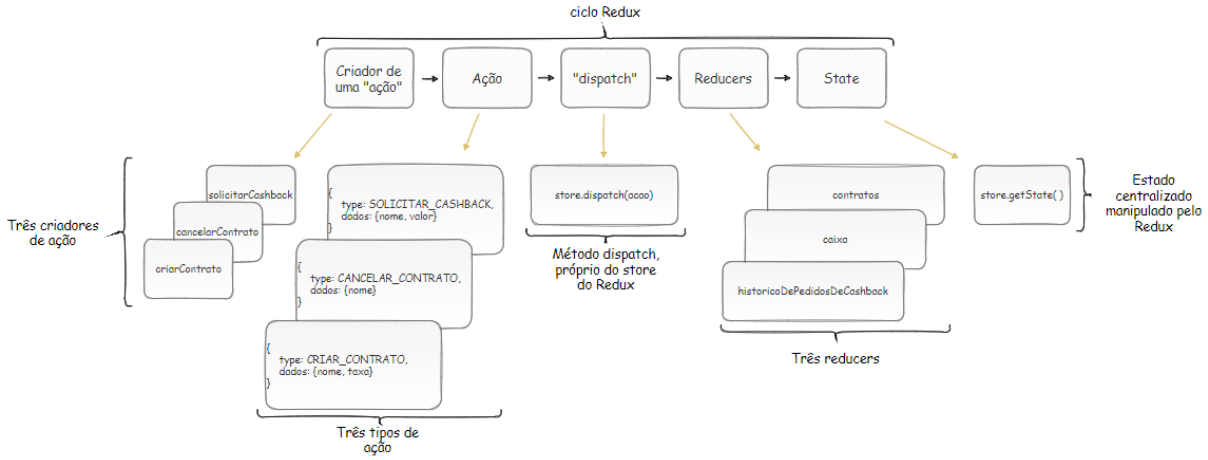
```
//depois da definição de todas as funções
const { createStore, combineReducers } = Redux

const todosOsReducers = combineReducers({
  historicoDePedidosDeCashback,
  caixa,
  contratos
})

const store = createStore(todosOsReducers)
```

2.12 (Partes do ciclo Redux implementadas até então) A Figura 2.12.1 exibe a implementação completa do ciclo Redux. Observe que o objeto **store** que criamos possui um método chamado **getState**. O objeto devolvido por ele desempenha o papel de estado. Ele pode ser usado assim: **store.getState()**. E o item "dispatch"? Ele é um método do objeto store que passaremos a utilizar a seguir. Ele pode ser usado assim: **store.dispatch(acao)**.

Figura 2.12.1



2.13 (Sequência de ações enviadas ao Redux) O programa do Bloco de Código 2.13.1 realiza as seguintes tarefas em ordem:

- Cria um contrato para José
- Cria um contrato para Maria
- Solicita cashback de 10 para Maria
- Solicita cashback de 20 para José
- Cancela o contrato de Maria

Observe que a execução acontece automaticamente assim que o projeto é salvo. Ele pode ser salvo automaticamente ou você pode clicar **Save** no canto superior esquerdo. Abra o console do navegador (CTRL + SHIFT + I no Chrome, por exemplo) para visualizar o resultado.

Bloco de Código 2.13.1

```
const acaoContratoJose = criarContrato('José', 50)
store.dispatch(acaoContratoJose)
console.log(store.getState())
const acaoContratoMaria = criarContrato ('Maria', 50)
store.dispatch(acaoContratoMaria)
console.log(store.getState())
const acaoCashbackMaria = solicitarCashback('Maria', 10)
store.dispatch(acaoCashbackMaria)
console.log(store.getState())
const acaoCashbackJose = solicitarCashback('José', 20)
store.dispatch(acaoCashbackJose)
console.log(store.getState())
const acaoCancelaContratoMaria = cancelarContrato ('Maria')
store.dispatch(acaoCancelaContratoMaria)
console.log(store.getState())
```

Referências

React – A JavaScript library for building user interfaces. 2021. Disponível em <https://reactjs.org/>. Acesso em agosto de 2021.

Redux - A predictable state container for JavaScript apps. | Redux. 2021. Disponível em <https://redux.js.org>. Acesso em outubro de 2021.