

Local Storage e Ciclo de Vida

Labenu_



O que vamos ver hoje?

- Local Storage
- O que é um Ciclo de Vida?
- Como usar métodos de ciclo de vida



Local Storage

Labenu_



Motivação

- Até agora, só lidamos com informações criadas **durante a vida da aplicação**
- Ao atualizar a página, **perdemos** todos os dados!
- Não conseguimos **persistir** dados
- Uma possível solução para isso é o **Local Storage**



Local Storage

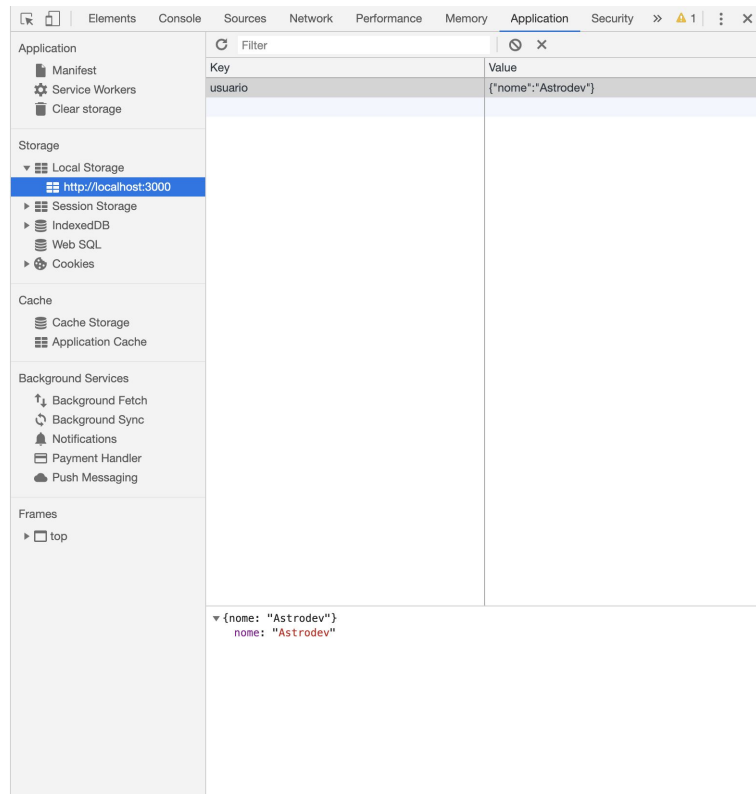
- É uma funcionalidade **do navegador** que permite que dados sejam **salvos** e **lidos**
- Dados são armazenados associados a um **domínio** (URL) e ficam guardados mesmo que o navegador seja fechado
- Está disponível globalmente no navegador por meio do objeto **localStorage**



Vendo os dados salvos



- Na aba **Application** do DevTools, temos acesso aos dados salvos no localStorage
- Usamos isso para ver o funcionamento do código



Guardando dados

- Para guardar dados, usamos a função `setItem()`
- Ela recebe dois parâmetros:
 - **Chave:** identificador do que guardaremos
 - **Dados:** dados a guardar. Devem ser uma string

```
localStorage.setItem("usuario", "Astrodev")
```

Vamos ver na prática! 



Buscando dados

- Para buscar dados, usamos a função `getItem()`
- Ela recebe um parâmetro:
 - **Chave:** identificador do que estamos buscando. Deve ser **a mesma usada para guardar** o dado

```
localStorage.getItem("usuario")
```

Vamos ver na prática! 



Problema

- Local Storage **só armazena strings**
- Mas frequentemente vamos querer guardar outros tipos de valores, como **arrays** e **objetos**
- Para isso, podemos **transformá-los** em string na hora de salvar e voltar ao formato original na hora de pegá-los de volta



Solução

- **Array/Objeto \Rightarrow String**
 - `JSON.stringify()` transforma objetos e arrays em string
- **String \Rightarrow Array/Objeto**
 - `JSON.parse()` transforma string em objetos e arrays

Vamos ver na prática! 



Como usar Local Storage

```
1 const novoUsuario = {nome: "Astrodev", idade: 30}
2
3 localStorage.setItem("usuario", JSON.stringify(novoUsuario))
4
5 const usuarioString = localStorage.getItem("usuario")
6
7 const usuarioObjeto = JSON.parse(usuarioString)
8
9 console.log(usuarioObjeto.nome) //Astrodev
```





Exercício 1

- Crie um componente de formulário com 3 inputs controlados
- Esses inputs representam a **edição do perfil** de um usuário e devem ser: **nome**, **email** e **idade**

Nome:

Email:

Idade:





Exercício 2

Crie dois botões no seu formulário:

- Um para **salvar** os dados no localStorage
- Um para **pegar** os dados salvos e mostrar nos inputs



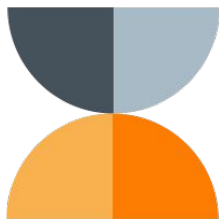
Exemplo do Formulário

- Esse comportamento não é o melhor para um formulário de perfil!
- Algo mais interessante seria eliminar os botões:
 - Salvar os dados **automaticamente** conforme eles são escritos
 - Assim que a tela abrir, **automaticamente** o formulário é preenchido com os dados corretos



Pausa para relaxar 🧘

10 min



```
1 const novoUsuario = {nome: "Astrodev", idade: 30}
2
3 localStorage.setItem("usuario", JSON.stringify(novoUsuario))
4
5 const usuarioString = localStorage.getItem("usuario")
6
7 const usuarioObjeto = JSON.parse(usuarioString)
8
9 console.log(usuarioObjeto.nome) //Astrodev
```



O que é um Ciclo de Vida?

Labenu_






Aviso

- O conteúdo a seguir (ciclo de vida) será abstrato e pouco prático **por enquanto**
- Ele será necessário para outro conteúdo que veremos a partir da **quinzena que vem**
- Não tem problema algum não entender tudo e/ou não ver aplicação do que for feito hoje
 - Tudo que for relevante será explicado novamente no momento de aplicar



Ciclo de vida (ou lifecycle)

- São as **fases da vida** de um componente
 - Montagem 
 - Atualização 
 - Desmontagem 
- Cada uma dessas fases podem ser acessadas por **métodos especiais** de um componente de **classe**



Ciclo de vida (ou lifecycle)

- Esses métodos de ciclo de vida são um pouco diferentes dos métodos que trabalhamos até agora
- Não somos nós que invocamos eles, mas sim os **momentos específicos** do ciclo de vida
- Usamos esses métodos para fazer ações **quando** um determinado ciclo de vida do componente acontecer



Momentos do ciclo de vida

- **Montagem:** assim que o componente é renderizado pela primeira vez na tela
- **Atualização:** assim que alguma informação é alterada no estado ou nas props do componente
- **Desmontagem:** logo antes de o componente deixar de estar renderizado na tela



Montagem: *componentDidMount()* 🧒

- Invocado logo após o método **render()** ter sido executado **pela primeira vez**
- Utilizado normalmente para fazer **requisições externas**, como uso de um banco de dados, com recursos que o componente precisará
- **Ex:** quando você entra em uma rede social, você precisa apertar um botão para pegar os posts?



Atualização: *componentDidUpdate()*



- Invocado em **todas as atualizações** de informação que o componente recebe (seja de estados ou de props)
- Útil para verificarmos **se alguma prop ou estado mudou** para realizar uma ação após cada atualização
- **Ex:** quando você começa a digitar na busca e os resultados vão aparecendo ao vivo



Montagem & Atualização: *render()*

- É invocado tanto na **montagem** do componente quanto em sua **atualização**
- É o responsável por renderizar os **elementos do JSX** na tela
- Por isso, faz todo sentido ele acontecer tanto na montagem quanto na atualização, pois **coisas mudarão na nossa tela**

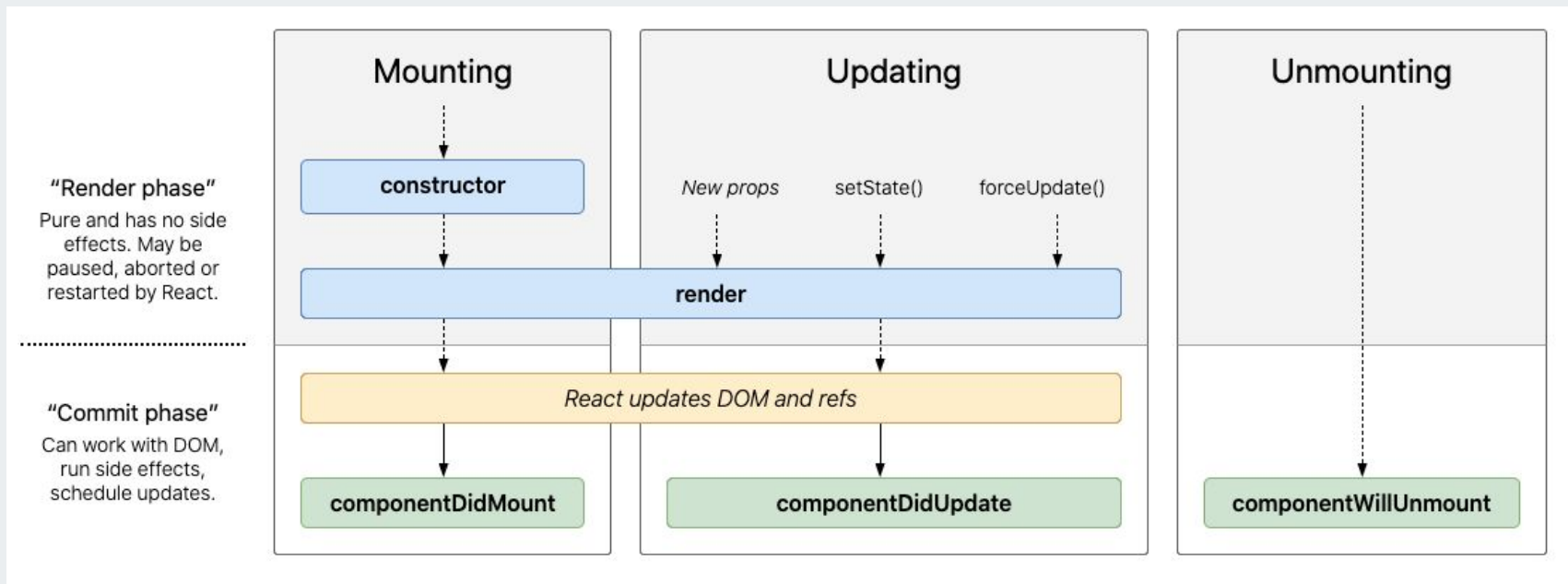


Desmontagem: *componentWillUnmount()* 🧑

- Invocado logo **antes** do componente ser desmontado
- Útil para desfazer operações que fizemos no `componentDidMount()`
- Não será muito utilizado, mas é interessante saber que ele existe caso encontrem por aí algum dia



Ciclo de Vida do Componente



Fonte: <http://projects.wojtekma.pl/react-lifecycle-methods-diagram/>



Como usar métodos de Lifecycle

Labenu_



Como usar? 🤔

- Para usar os métodos de lifecycle, basta declararmos eles como **métodos de classe**
- **Eles só funcionam em componentes de classe**
- A forma de declarar é **a mesma do método `render()`**



Como usar Métodos de Lifecycle

```
1 class App extends React.Component {  
2   componentDidMount() {  
3     // Executa após a montagem do componente  
4   }  
5  
6   componentDidUpdate() {  
7     // Executa após a atualização do componente  
8   }  
9  
10  componentWillUnmount() {  
11    // Executa antes da desmontagem do componente  
12  }  
13  
14  render() {  
15    return <div>Componente</div>  
16  }  
17 }
```



componentDidMount 🧒

- Chamaremos dentro deste método as ações que quisermos que sejam executadas **automaticamente assim que a tela abrir**
- **Ex:** posts das redes sociais **simplesmente aparecem** para você na tela, sem apertar botões
- **Ex:** queríamos pegar os dados para preencher o form de perfil **automaticamente** no exercício



componentDidMount

```
1 import React from "react"
2
3 export default class App extends React.Component {
4   // Vamos supor que temos uma função que pega os
5   // posts de uma rede social de um banco de dados
6
7   pegarPosts = () => {
8     // Semana que vem aprenderemos como de fato pegar esses dados
9     // Então não se preocupem com isso agora, penas suponha que
10    // essa função é uma caixinha preta que pega os dados dos posts
11  }
12
13  componentDidMount() {
14    this.pegarPosts()
15  }
16
17  render() {
18    return (
19      <div>Aqui vem o layout da tela</div>
20    )
21  }
22 }
```



componentDidUpdate 🧑

- Chamaremos dentro deste método as ações que quisermos que sejam executadas automaticamente **quando algum estado ou prop for atualizado**
- **Ex:** queríamos salvar os dados do nosso formulário automaticamente conforme a pessoa for escrevendo
- Entretanto, essa função é executada quando QUALQUER **estado** ou **prop** muda





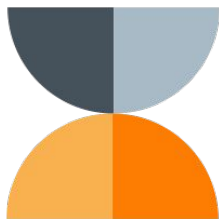
Exercício 3

- Agora vamos fazer com que esse formulário salve em tempo real o que está sendo escrito! Usaremos o Local Storage para persistir o estado do formulário (verifique usando a aba Application)
- **Assim que a tela abrir**, preencha os dados iniciais do formulário com os dados do Local Storage
- **Toda vez que os dados forem alterados**, salve-os no Local Storage



Pausa para relaxar 🧘

5 min



- **Ciclos de Vida:** são métodos executados em momentos diferentes da vida de um componente
- **Montagem:** `componentDidMount()`
- **Atualização:** `componentDidUpdate()`
- **Desmontagem:** `componentWillUnmount()`



componentDidUpdate 🧑

- As funções no componentDidUpdate são executadas quando QUALQUER **estado** ou **prop** muda, porém nem sempre queremos fazer isso!
- Vamos supor que uma tela tem um input de busca e um botão que muda o tema para dark mode
 - estados: searchInput e darkMode
- Toda vez que a pessoa atualizar **qualquer** um desses dois estados, a função do didUpdate vai acontecer



componentDidUpdate 🧑

- Para prevenir isso, o método `componentDidUpdate` dá para a gente dois parâmetros úteis:
 - `prevProps`
 - `prevState`
- Com eles eu consigo fazer **condicionais** que permitem ver se uma determinada prop ou estado mudou e só executar a ação em caso positivo



Como usar prevProps/prevState

```
1 import React from "react"
2
3 export default class App extends React.Component {
4   state = {
5     darkMode: false,
6     searchInput: ""
7   }
8
9   componentDidUpdate(prevProps, prevState) {
10     // Vamos verificar se o valor do input mudou
11     if (prevState.searchInput !== this.state.searchInput){
12       // Caso tenha mudado, fazemos a operação de busca
13       // Se não mudou, não fazemos nada
14       executeSearch()
15     }
16   }
17
18   render() {
19     return (
20       <div>Aqui vem o layout da tela</div>
21     )
22   }
23 }
```





Exercício 4

- Salve no localStorage apenas os dados que foram alterados naquele momento (nome, email, idade) e não todos ao mesmo tempo



Resumo

Labenu_



Resumo

- Local Storage **persiste** informações mesmo quando o navegador é fechado
- `setItem(chave, dado)` guarda dados
- `getItem(chave)` busca dados
- A chave deve ser a mesma ao **salvar** e **ler** o mesmo dado
- Local Storage **guarda apenas strings**



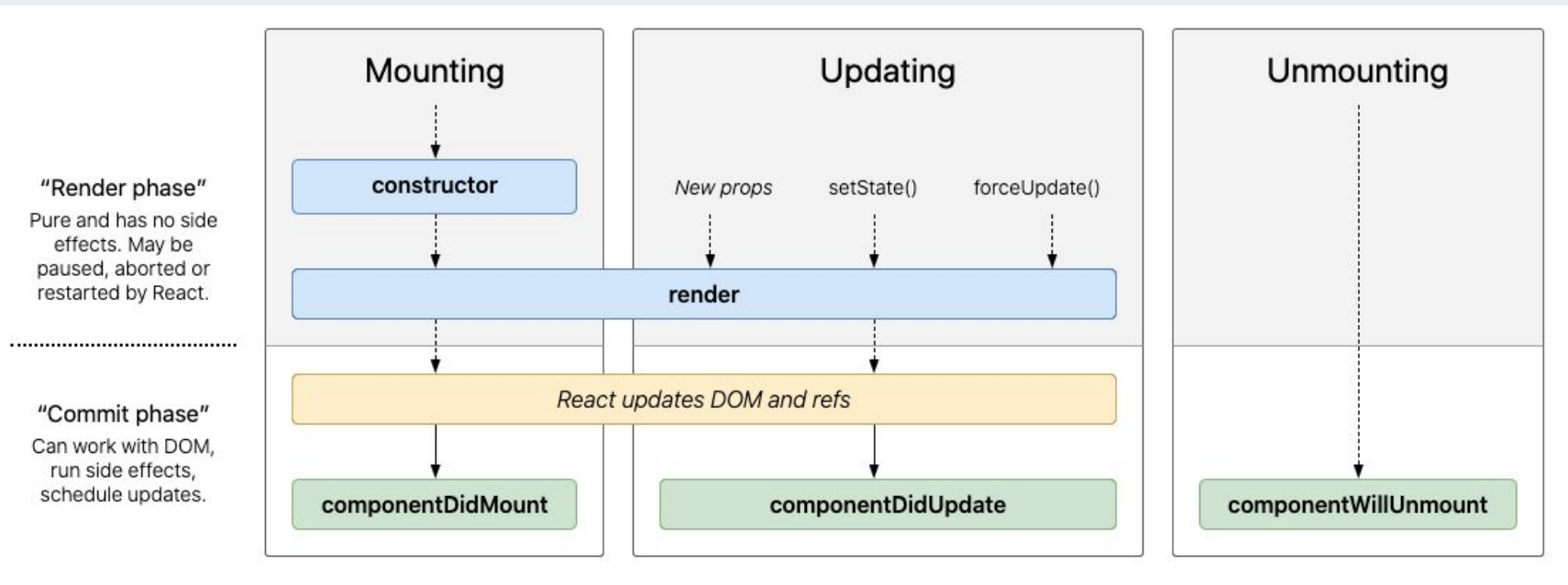
Resumo

- **Ciclos de vida** são métodos executados em momentos diferentes da vida de um componente

Montagem	<code>componentDidMount()</code>
Atualização	<code>componentDidUpdate()</code>
Desmontagem	<code>componentWillUnmount()</code>



Resumo



Dúvidas? 🧐

Labenu_





Obrigado(a)!