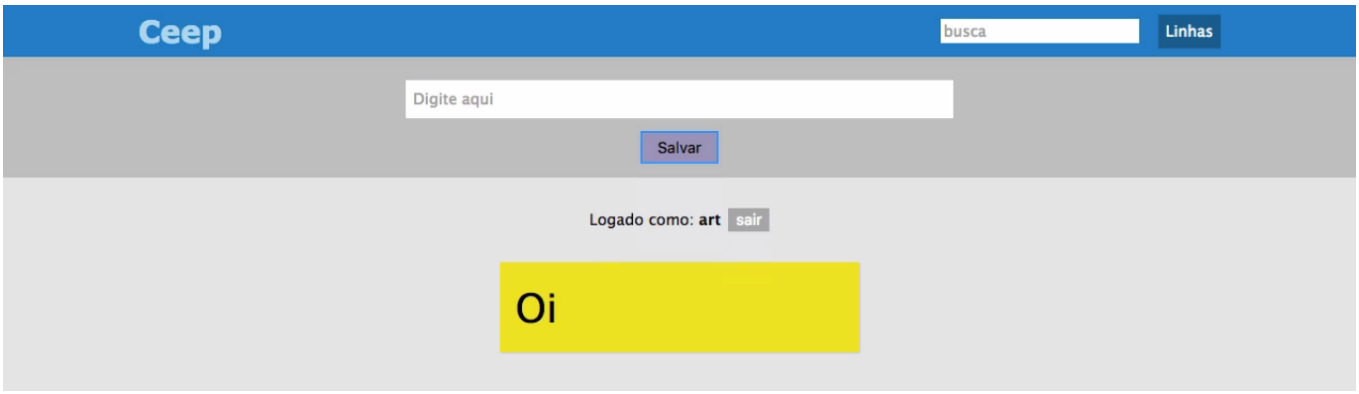
01

Salvando cartões com localStorage na adição

Transcrição

Continuaremos o nosso sistema. Agora que já conseguimos fazer o login do usuário, podemos adicionar os cartões associados ao nome de usuário.



Mas temos um problema: ao recarregar a página, os cartões somem.

Uma das principais funcionalidades que precisamos criar é o salvamento offline dos cartões. Precisamos poder recarregar a página sem perder os cartões, para poder vê-los e mudar sua cor e conteúdo. Assim, precisamos ver em que ponto do código os cartões são adicionados, e onde eles ficam armazenados enquanto a página está carregada.

Voltaremos então para o nosso `Mural.js` , para a função `adiciona` .

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)

  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)
      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
```

```
    render()
        return true
      } else {
        alert("Você não está logado")

      }

  }
```

Essa função verifica prontamente se o usuário está logado, e, se não estiver, emite o alerta: "Você não está logado". Caso ele já estivesse logado, o programa segue toda a lógica para adicionar o cartão.

Uma das coisas necessárias para que o cartão seja adicionado na página, está na linha `cartoes.push(cartao)`. E `cartao` é o que recebo como parâmetro, e quando alguém digita algo no campo, como `teste`, estará chamando a função `adiciona()`.

Temos uma lista de cartões no mural, e é ela que armazena os cartões na memória. Quando recarregamos a página, a lista fica novamente vazia(`let cartoes = []`) e perdemos os cartões criados. A ideia é que a lista não fique vazia, mas que peguemos os cartões do usuário que estiver logado.

Sabendo que a lista precisa ter algo que estará salvo, usaremos o `localStorage`. Diremos que queremos pegar um item (`getItem`) chamado `"cartoes"`. E esse item ainda não foi salvo.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = localStorage.getItem("cartoes")
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)
```

E o que será que volta do `localStorage` sem ter a sua propriedade definida? Abriremos novamente o console no navegador (`Botão direito > Inspect`). E lá, digitaremos:

```
> localStorage.getItem("cartoes")
```

O que o console nos devolve é:

```
> localStorage.getItem("cartoes")
< null
```

Ou seja, agora não é mais uma lista vazia. Ela dará problema em nossa página. Então, precisamos comunicar que, quando não houver cartão nenhum, precisamos que essa seja uma lista vazia. O que podemos fazer com o JavaScript é uma sacada bem legal. Ele será ou o que vier do `localStorage` , ou (`||`) uma lista vazia (`[]`).

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = localStorage.getItem("cartoes") || []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)
```

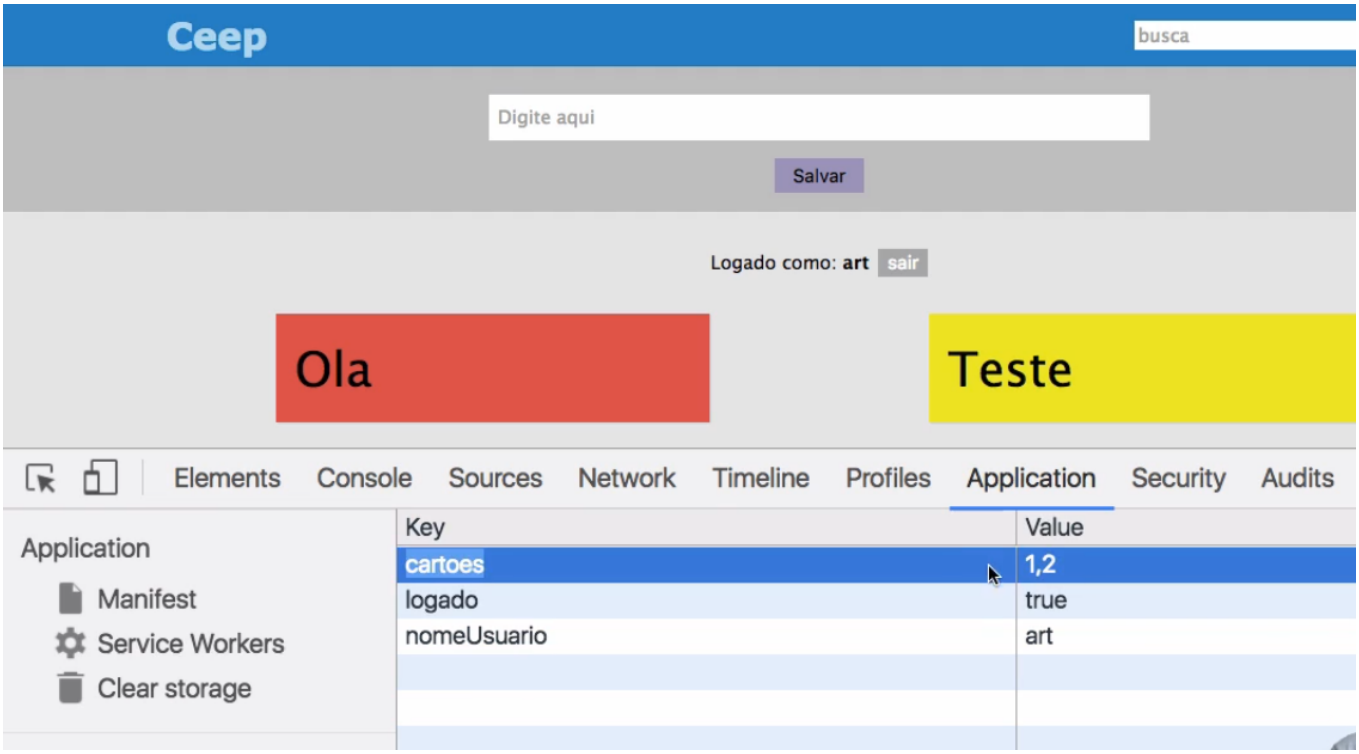
O próprio programa decidirá. Se a listar der `null` , ele poderá escolher a lista vazia. Mas se realmente houver uma lista, ele puxará os cartões do `localStorage` . Já sabemos que o `getItem` do `localStorage` não devolve exatamente uma lista. Vamos testar no console, usando o `setItem` para estabelecer uma lista de números.

```
> localStorage.getItem("cartoes")
< null
> localStorage.setItem("cartoes", [1,2])
< undefined
```

O que será que voltará quando usarmos o `getItem` ?

```
> localStorage.getItem("cartoes")
< null
> localStorage.setItem("cartoes", [1,2])
< undefined
> localStorage.getItem("cartoes")
"1,2"
```

Continua a voltar como texto, que representa a lista. Mas note que não tem colchetes. Se formos ao `Local Storage` , veremos que os cartões não são uma lista do JavaScript, então não conseguiríamos fazer o `JSON.parse` .



Mas podemos testar no console, para ver o que acontece.

```
> localStorage.getItem("cartoes")
< null
> localStorage.setItem("cartoes", [1,2])
< undefined
> localStorage.getItem("cartoes")
"1,2"
> JSON.parse(localStorage.getItem("cartoes"))
> Uncaught SyntaxError: Unexpected token , in JSON at position 1(...)
```

Deu erro! Esse é um texto com vírgula, que é inválido. O `JSON.parse` funciona melhor se deixarmos esse texto com mais cara de lista, assim: `"[1, 2]"`.

```
> localStorage.getItem("cartoes")
< null
> localStorage.setItem("cartoes", [1,2])
< undefined
> localStorage.getItem("cartoes")
"1,2"
> JSON.parse(localStorage.getItem("cartoes"))
> Uncaught SyntaxError: Unexpected token , in JSON at position 1(...)
> JSON.parse("[1,2]")
[1, 2]
```

Assim, temos que armazenar esses valores de uma forma diferente, como texto. E o recuperaremos com o `JSON.parse`. Colocaremos o `JSON` no código, portanto.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
```

```
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)
```

Manteremos a lista vazia como alternativa, caso não venha nada. No `JSON.parse`, o que retornaria se não não houver nada(`null`)?

```
> localStorage.getItem("cartoes")
< null
> localStorage.setItem("cartoes", [1,2])
< undefined
> localStorage.getItem("cartoes")
"1,2"
> JSON.parse(localStorage.getItem("cartoes"))
> Uncaught SyntaxError: Unexpected token , in JSON at position 1(...)
> JSON.parse("[1,2]")
< [1, 2]
> JSON.parse(null)
< null
```

Como ele retorna `null`, dá certinho com o que a gente precisa. A recuperação dos dados já está funcionando. Só precisamos salvar os dados no `localStorage`.

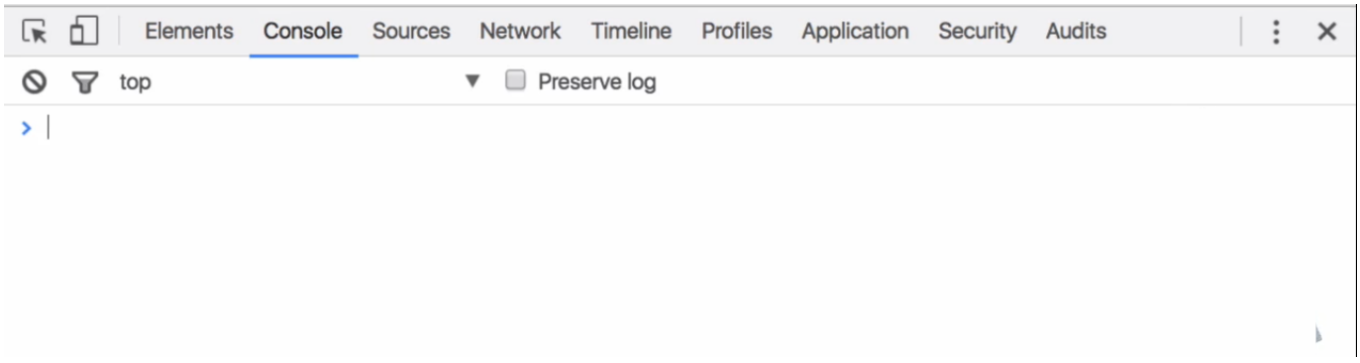
Se todo cartão passa pela função `adiciona`, podemos adicioná-los ao `localStorage` assim que a função for chamada. Assim, se o usuário estiver logado e o cartão for adicionado, já será salvo também no `localStorage` com o `setItem()`, que é a função que acrescenta itens.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)

  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)

      localStorage.setItem("cartoes", cartao)

      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
  render()
      return true
    } else {
      alert("Você não está logado")

    }

}
```

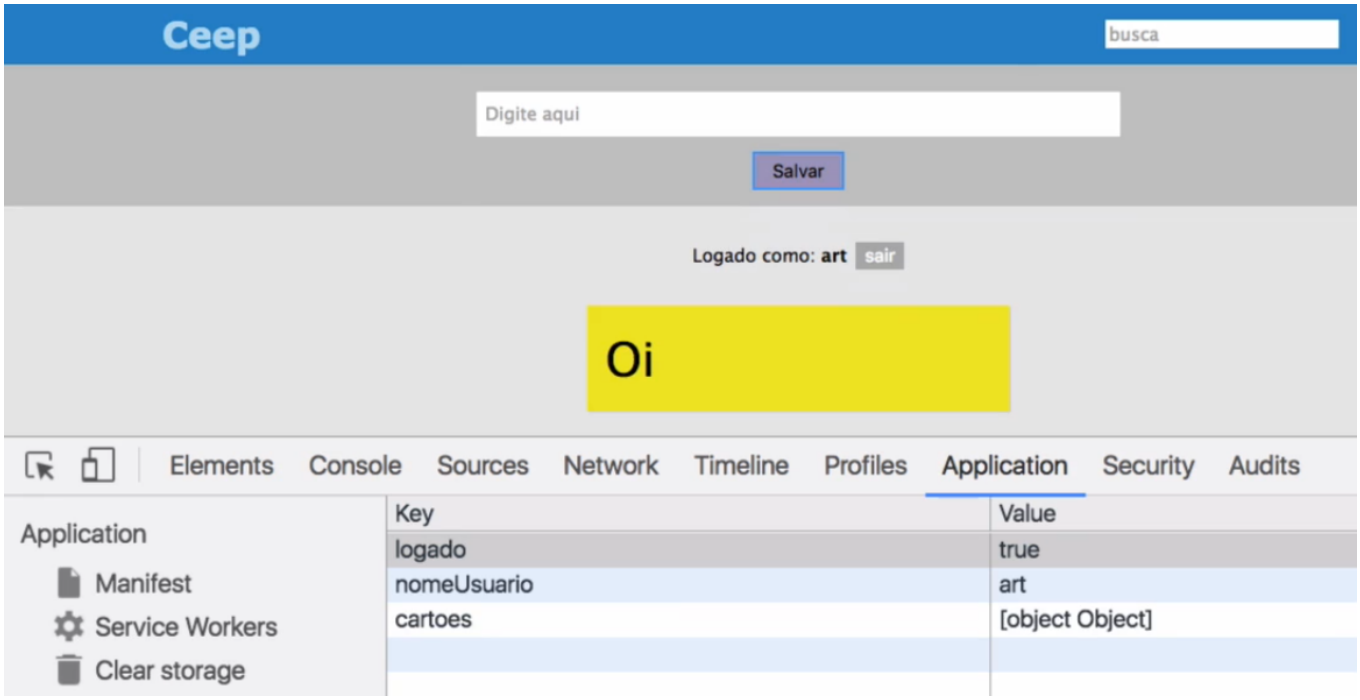
Tem algo estranho na linha que adicionamos. Estaremos adicionando apenas um dado. Essa não é uma lista de todos os cartões da página. E a cada novo cartão, estaríamos substituindo o anterior, e não acrescentando ao que já havia antes. Isso significa que não estamos conseguindo adicionar itens a uma lista do `localStorage`. Precisamos adicionar a lista inteira, de novo, como o novo cartão adicionado. Assim, o que precisamos adicionar não é `cartao`, mas `cartoes`.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)

  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)

      localStorage.setItem("cartoes", cartoes)

      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
  render()
      return true
    } else {
      alert("Você não está logado")

    }

}
```

Agora sim estamos armazenando a lista de cartões. Pelo console, parece que está tudo certo.



Podemos dar uma olhada no `Local Storage`, para ver como os cartões estão sendo armazenados.

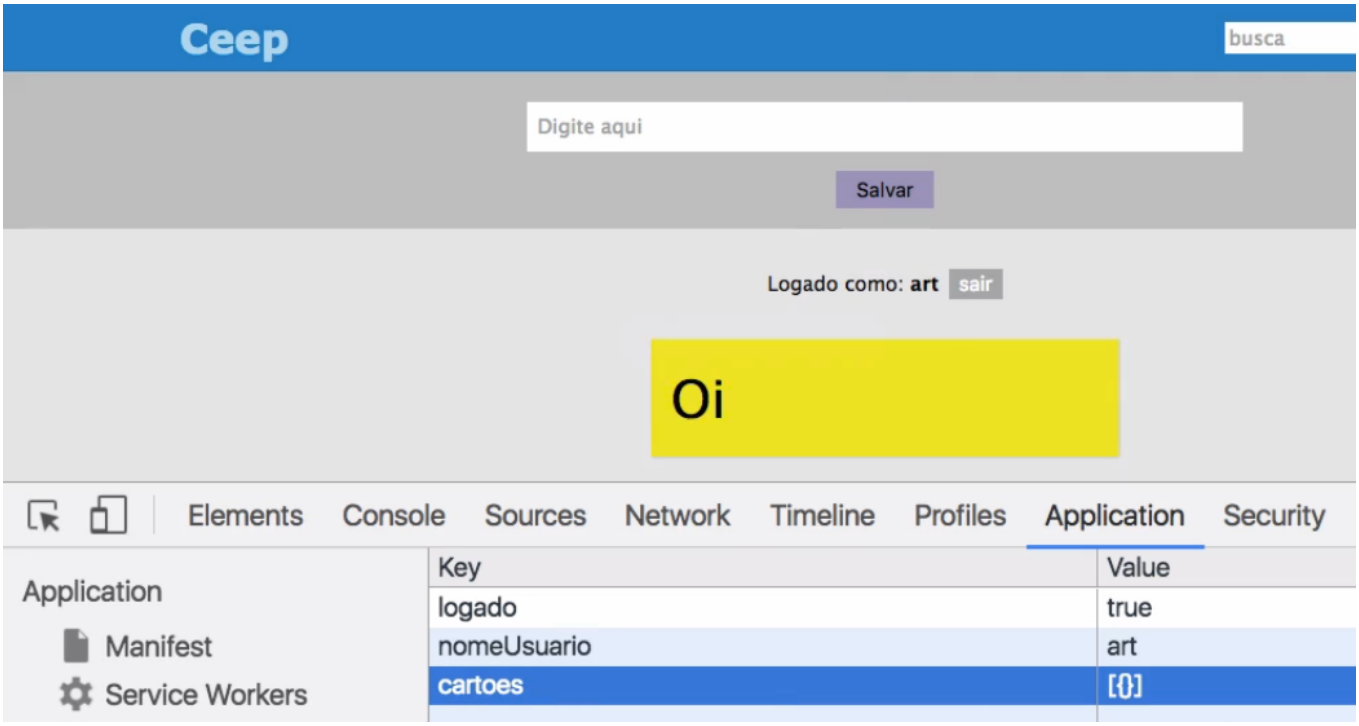


O valor de `cartoes` está sendo armazenado como `[object Object]`, que não é uma lista válida. Temos que salvar o texto dos cartões com colchetes e vírgulas, como se estivéssemos escrevendo uma lista à mão. E como transformar isso que vemos em um texto? Estaremos fazendo o contrário de quando pegamos um texto e o transformamos em JavaScript, boleano ou lista. É o contrário do `JSON.parse`.

Antes de salvar os cartões no `setItem`, precisamos transformá-los em texto. Para isso, usaremos o `JSON.stringify`, que fará essa transformação.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)

  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)

      localStorage.setItem("cartoes", JSON.stringify(cartoes))

      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
  render()
      return true
    } else {
      alert("Você não está logado")

    }

}
```

Vamos salvar e limpar o `Local Storage`, o que é essencial justamente porque estávamos salvando os cartões como objetos. Agora poderemos voltar ao navegador para ver como os cartões estão sendo salvos.



Esse `[{}]` é uma lista, mas o objeto que vemos aí não tem nenhuma propriedade. O que acontece é que, pela forma que os cartões foram criados, o `stringify` não atribui propriedades que possam ser lidas a eles. São propriedade não enumeráveis, questões de orientação de objetos do JavaScript.

Assim, precisaremos transformar esse objeto cartão, que é bem complexo, em um objeto com propriedades. Precisamos salvar o conteúdo e o tipo do cartão, que inclui sua cor. E é isso que precisamos dizer na hora de usar o `stringify`, pois não queremos uma lista de cartões usando o objeto `cartao` recebido como parâmetro no `adiciona`. Queremos um cartão especial para o `localStorage`.

Transformaremos a lista de cartões em uma lista de objetos que queremos adicionar, o que exige um código um pouco mais complexo que o `setItem`. Assim, deletaremos a sua linha e chamaremos a função `salvaCartoes`.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)

  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)

      salvaCartoes()

      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
```

```
    render()
        return true
      } else {
        alert("Você não está logado")

      }

  }
```

E criaremos a função nesse arquivo mesmo, logo depois de `Filtro` .

```
  Const Mural = (function(_render, Filtro){
    "use strict"
    let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
    const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

    Filtro.on("filtrado", render)

    function salvaCartoes (){

    }

    function adiciona(cartao){
      if(logado){
        cartoes.push(cartao)
        salvaCartoes()
        cartao.on("mudanca.**", render)
        cartao.om("remocao", ()=>{
          cartoes = cartoes.slice(0)
          cartoes.splice(cartoes.index0f(cartao),1)
        render()
    )}
    render()
        return true
      } else {
        alert("Você não está logado")

      }

  }
```

Ela fará aquilo que já estávamos fazendo com a linha `localStorage.setItem("cartoes", JSON.stringify(cartoes))` , mas um pouco melhor.

```
  Const Mural = (function(_render, Filtro){
    "use strict"
    let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
    const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

    Filtro.on("filtrado", render)

    function salvaCartoes (){
      localStorage.setItem("cartoes", JSON.stringify(cartoes))
    }
```

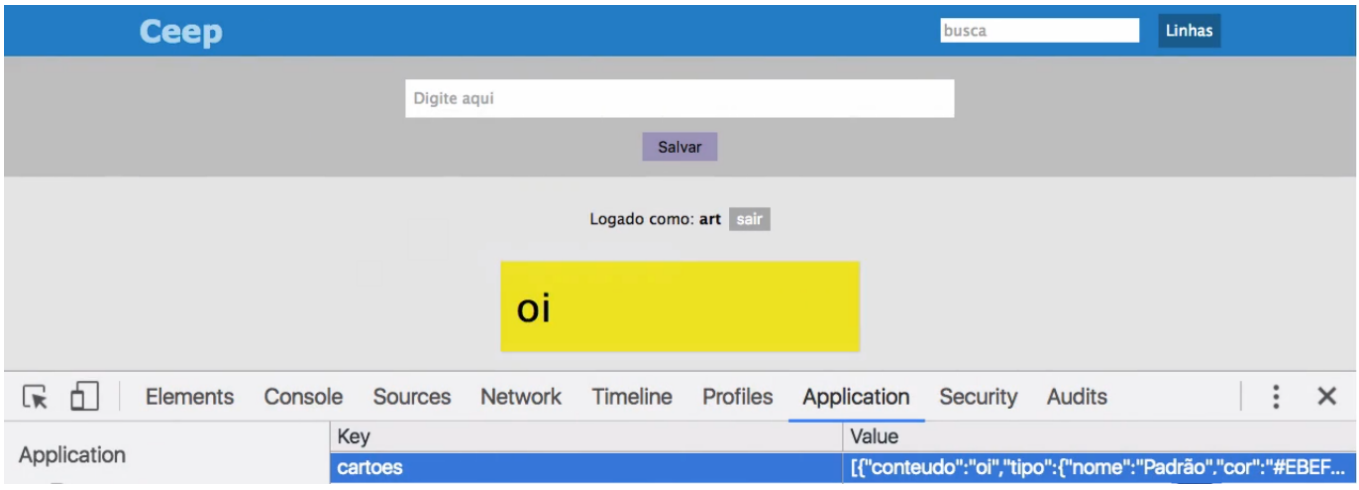
```
  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)
      salvaCartoes()
      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
  render()
      return true
    } else {
      alert("Você não está logado")

    }

}
```

A nova lista de cartões que iremos armazenar terá o mesmo tamanho da anterior, mas será uma lista mapeada. Para cada cartão que tivermos dentro dessa lista, retornaremos um objeto que terá a propriedade `conteudo`, cujo valor será `cartao.conteudo`, e a propriedade `tipo`, cujo valor será `cartao.tipo`. Esse será um .objeto literal do JavaScript.

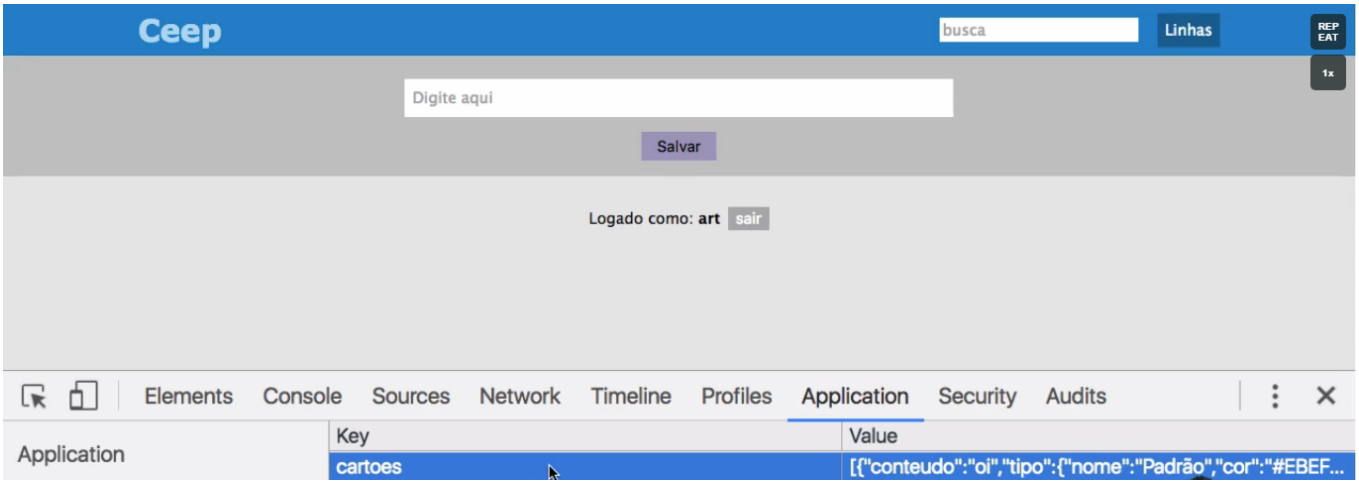
```
function salvaCartoes (){
  localStorage.setItem("cartoes", JSON.stringify(
    cartoes.map(cartao => ({conteudo: cartao.conteudo, tipo: cartao.tipo})
    ))
}
```

Quando o `stringify` for chamado, conseguirá entender a propriedade `conteudo` e a propriedade `tipo`. É diferente de fazer o `stringify` diretamente da lista de cartoes, pois cada cartão não é um objeto literal do JavaScript; eles foram criados de outra maneira. Desse jeito conseguiremos salvar o conteúdo do cartão.

Voltaremos ao navegador e deletaremos o `cartoes` que está armazenado com o valor `[{}]`. Recarregaremos a página e criaremos um novo cartão, com o texto `oi`.



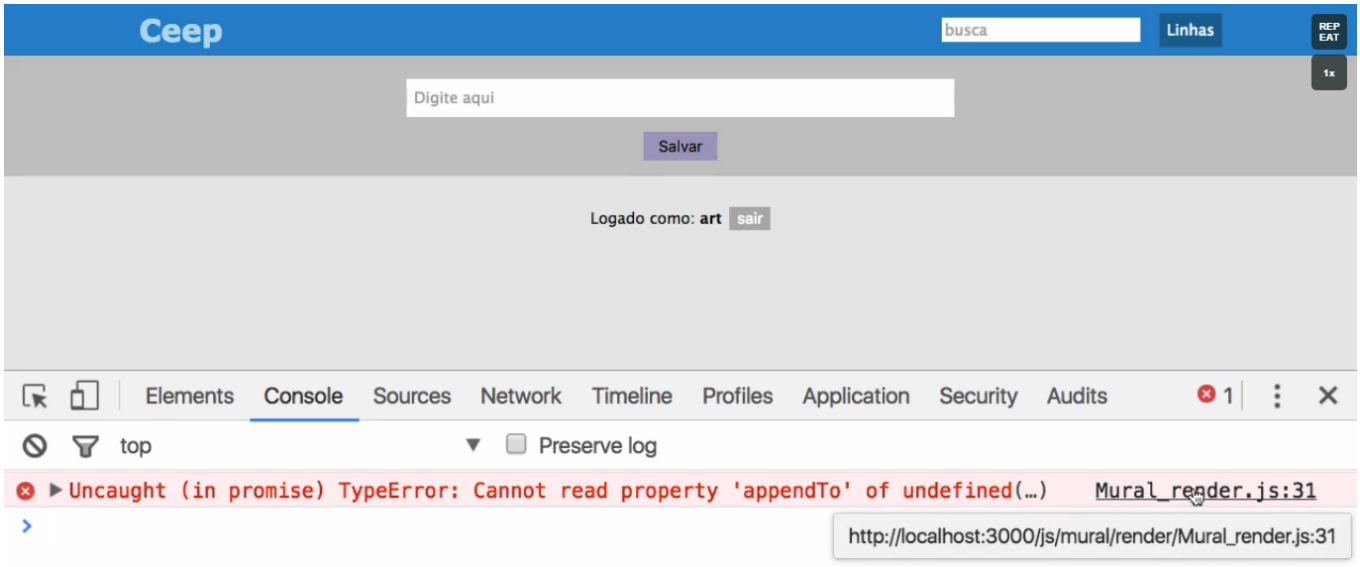
As propriedades estão armazenadas corretamente! Mas, quando recarregamos a página...



O cartão não aparece mais na aplicação. Os cartões ainda estão no `Local Storage`, mas não estão sendo renderizados. Estamos criando a lista de cartões, e o `render` está presente. Mas não estamos chamando o `render` na hora que a página carrega. Vamos chamá-la então, para forçar a renderização dos cartões que estão no `Local Storage`.

```
Const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem("cartoes")) || []
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  render()

  Filtro.on("filtrado", render)

  function salvaCartoes (){
    localStorage.setItem("cartoes", JSON.stringify(
      cartoes.map(cartao => ({conteudo: cartao.conteudo, tipo: cartao.tipo})
      ))
  }

  function adiciona(cartao){
    if(logado){
      cartoes.push(cartao)
      salvaCartoes()
      cartao.on("mudanca.**", render)
      cartao.om("remocao", ()=>{
        cartoes = cartoes.slice(0)
        cartoes.splice(cartoes.index0f(cartao),1)
      render()
  )}
  render()
      return true
    } else {
      alert("Você não está logado")

    }

}
```

Voltamos ao navegador para recarregar a página, forçando o render. Mas, quando abrimos o console, vemos um erro.



E esse erro nos diz que não é possível ler a propriedade `appendTo` em `Mural_render`. O cartão que é adicionado ao `Mural` é o que o `render` usa. Assim, o nosso `cartoes` precisa ser do tipo `cartao`, criado com a função `Cartao`, disponível no arquivo `Cartao.js`.

```
const Cartao = (funciton(_render, EventEmitter){
  "use strict"

  const autoIncremet = (function(){
    let id = 0
    return funciton (){
        return id++
    }
  }) ()

  const globalProps = {
    tipos: {
      padrao: {nome: "Padrao", cor: "#  EBEF40"}
      ,importante: {nome: "Importante", cor: "#F05450"}
      ,tarefa: {nome: "Tarefa", cor: "#92C4EC"}
      ,inspiracao: {nome: "Inspiração", cor: "#76EF40"}
    }
  }

  function Cartao(conteudo, tipo = globalProps.tipos.padrao){

    const render = _render(globalProps)
    function renderAfter(fn){
        return function(){
          fn.apply(this, arguments)
          render(props, state, handlers)
        }
    }
  }
}
})
```

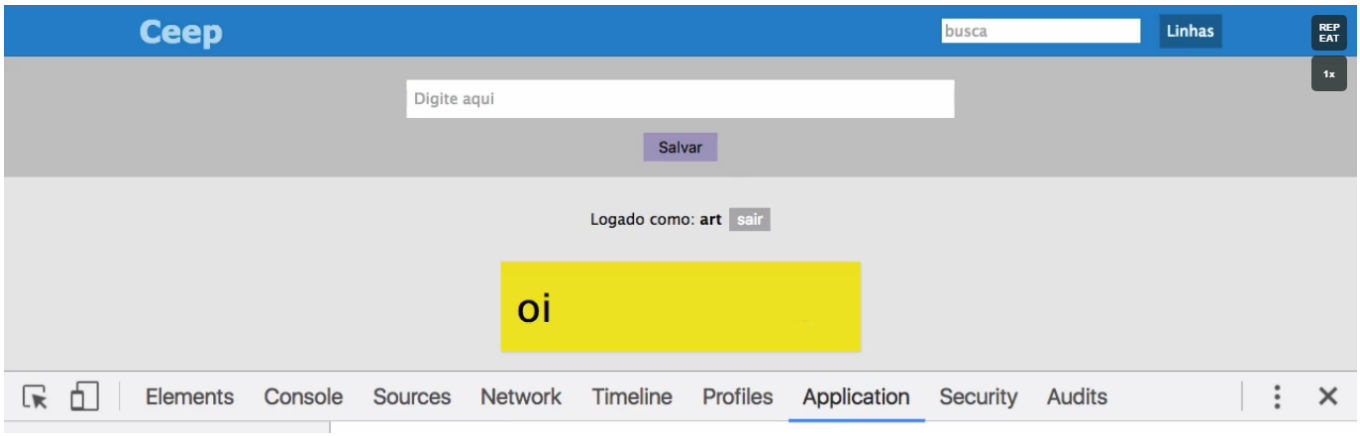
Ela é uma função construtora, que precisa ser chamada passando qual é o `conteudo` e o `tipo`. E a partir disso, ela construirá um objeto cartão do jeito que o `Mural_render` precisa. Não podemos simplesmente pegar os cartões que estão armazenados como objetos literais do JavaScript, precisamos de um objeto do tipo `cartao` da nossa aplicação.

Para cada cartão da lista que está armazenada, precisamos ter um objeto `cartao` gerado por essa função construtora. Precisaremos fazer um novo mapa, usando o `.map` .

Cada cartão que vem do `localStorage` , cujo nome será `cartaoLocal` , precisará chamar a função construtora `Cartao` , passando seu `conteudo` e seu `tipo` .

```
Const Mural = (function(_render, Filtro){
  "use strict"
let cartoes = JSON.parse(localStorage.getItem("cartoes") || [])
                  .map(cartaoLocal => new Cartao(cartaoLocal.conteudo, cartaoLocal.tipo))
  const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto});

  Filtro.on("filtrado", render)
  ...
}
```

Sendo um novo mapeamento, estamos criando uma lista nova baseada na lista salva no `localStorage` . Podemos salvar o arquivo e voltar para testar no navegador. Cpmo já pedimos o `render` e já temos o cartão `oi` armazenado, deveria bastar um refresh na página para que um cartão aparecesse.



O nosso cartão renderizado voltou. Isso significa que agora podemos adicionar quantos cartões quisermos, e mudar os seus tipos, que ele serão adequadamente armazenados. Criaremos o cartão `Teste` , que será vermelho.

Ao atualizar a página, veremos:

Ele ficou amarelo! O cartão foi armazenado, mas o seu tipo ainda não está funcionando. Veremos em breve como resolver isso. Até lá!