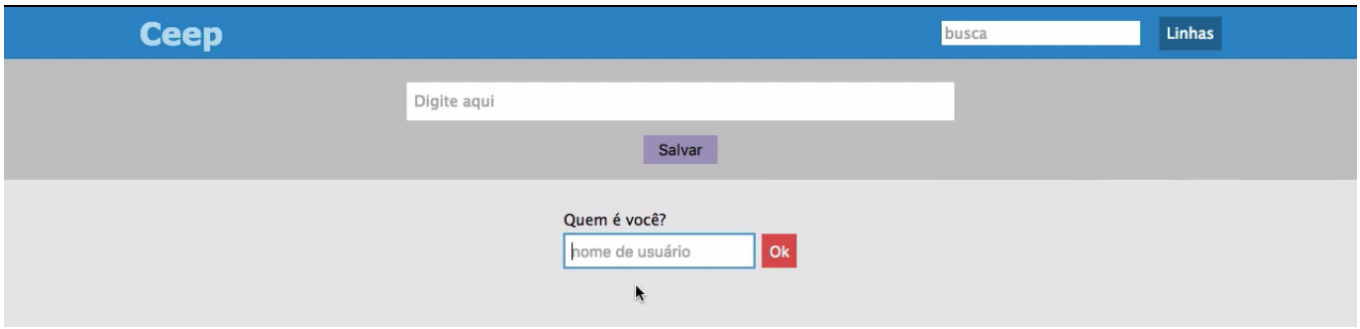
08

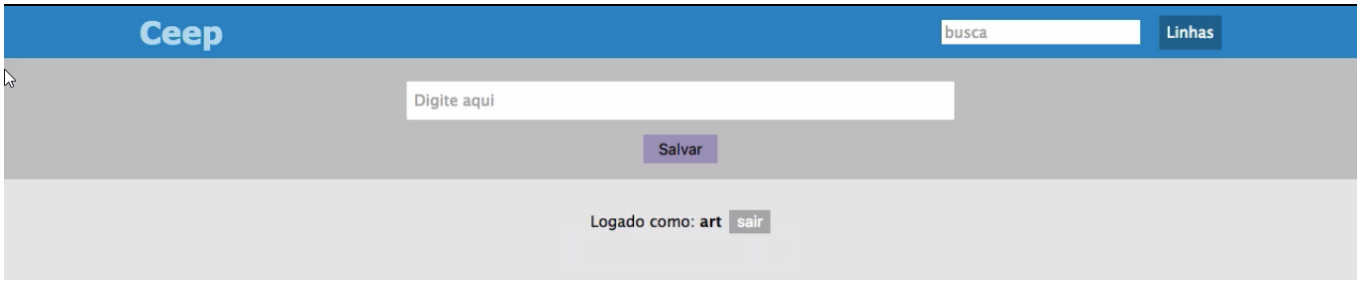
Separando os cartões por usuário

Transcrição

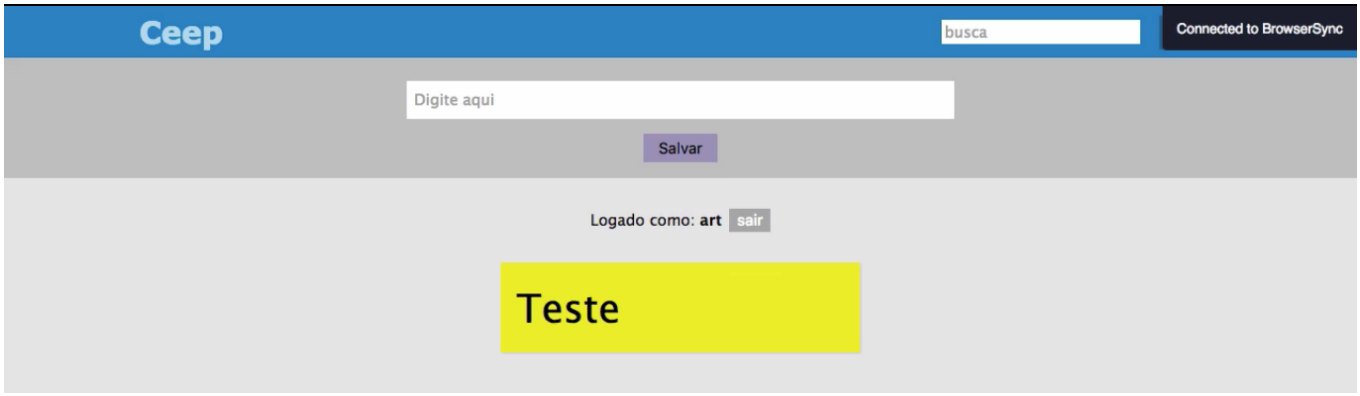
Vamos continuar nosso sistema! Os cartões já estão sendo salvos offline, portanto cumprimos um dos nossos objetivos.

Vamos testar o fluxo inteiro da aplicação? Começamos pelo login:



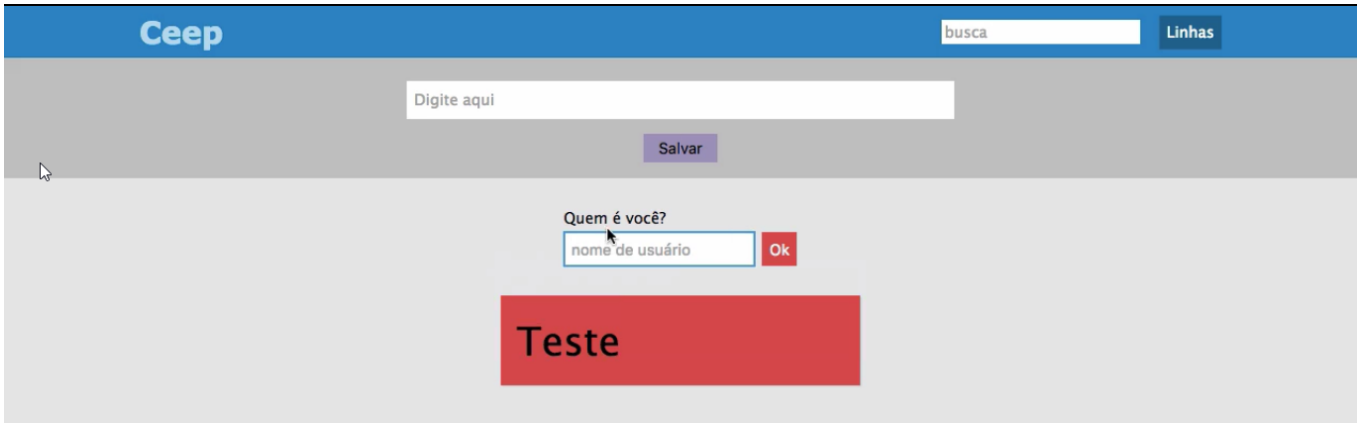


Conseguimos adicionar cartões depois do login.



Podemos mudar a cor do cartão, e ao atualizarmos, ele continua com a nova cor.

Mas ainda há algumas coisas estranhas acontecendo. Por exemplo, quando o usuário clica em `sair`, os cartões continuam na página, pois não os removemos lá no mural.

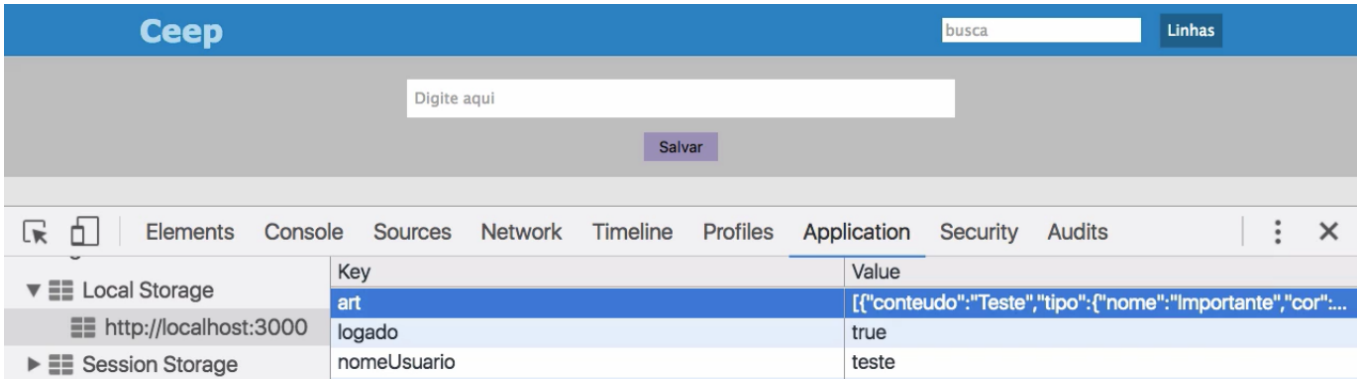


Não sabemos quando o usuário deslogou ou logou. Sabemos apenas que não podemos adicionar quando ele não está logado. Fazer alguma ação ou remover os cartões do mural quando o usuário deslogar, ou trazê-los quando ele logar, são coisas que não fizemos.

Por exemplo, neste teste que fizemos, caso eu recarregue a página enquanto logado como "teste", aqueles cartões que criei como "art" estão disponíveis para o usuário logado como "teste". Neste caso, precisamos de uma forma que deixe os cartões disponíveis para cada usuário, para termos essa divisão, que é o que queríamos a partir do login, ou seja, ter os cartões dos respectivos usuários separados.

Quando damos uma olhada no código que carrega os cartões, pegamos sempre a propriedade `cartões` do `localStorage`, o qual não depende do usuário, pois é sempre a mesma, através da função `salvaCartoes()`. O que acontece é que não conseguimos diferenciar um cartão do outro, e o que se pode fazer é ter uma lista de cartões para cada usuário. Se tenho o usuário "art", lá no `localStorage` terei as listas de cartões desse usuário. Conseguimos usar o `localStorage` a nosso favor.

Indo ao `localStorage`, salvamos sempre "cartoes", genericamente, em toda lista de cartões. Mas poderia também falar que os cartões são somente do usuário "art":



Daí, para cada usuário logado, pegamos a propriedade com o `nomeUsuario`, ou seja, não se trata mais de uma propriedade genérica. Essa será nossa solução: utilizar o comportamento de chave e valor do `localStorage` a nosso favor. Ainda logado como "teste", o programa não carregará nada. Caso alteremos nosso código, ele só pegará os cartões do usuário "art".

Então, quando deslogar e posteriormente logar como "art", recarregando a página, o que temos que fazer é com que ele pegue as propriedades dos cartões desse usuário. Essa é a funcionalidade que iremos implementar.

Primeiro precisamos saber quem é o usuário. Na hora de salvar os cartões, não podemos mais fazê-lo por uma propriedade fixa. Salvaremos usando o nome do usuário, dizendo que tenho um nome de usuário com uma variável (através de `nomeDoUsuario`):

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

Eu poderia criar uma variável, porém a lógica do login também deve estar junto, aqui. A lógica do login já foi feita. Para criar a variável `nomeDoUsuario`, posso ir até o arquivo do login (`LoginUsuario.js`). Ao abrirmos este arquivo, tenho acesso ao nome do usuário:

```
let logado = JSON.parse(localStorage.getItem("logado"))

LoginUsuario_render({
  logado: logado
  ,usuario: localStorage.getItem("nomeUsuario")
  ,onLogin: (nomeUsuario) => {
    logado = true
    localStorage.setItem("logado", true)
    localStorage.setItem("nomeUsuario", nomeUsuario)

  }
  ,onLogout: () -> {
    logado = false
    localStorage.setItem("logado", false)
    localStorage.removeItem("nomeUsuario")
  }
  })
```

Sempre que o usuário faz login, a função `nomeUsuario` é chamada, é ela que estamos usando para salvar no `localStorage` informações tais quais se ele estava logado ou não, seu nome...

Vamos criar uma variável `nomeUsuario`, a qual não terá nenhum valor quando a página for carregada por enquanto e, quando o usuário faz login, direi que `nomeUsuario` é igual ao informado anteriormente, usando "usuario" no lugar de "nomeUsuario" para evitar conflitos e simplificar:

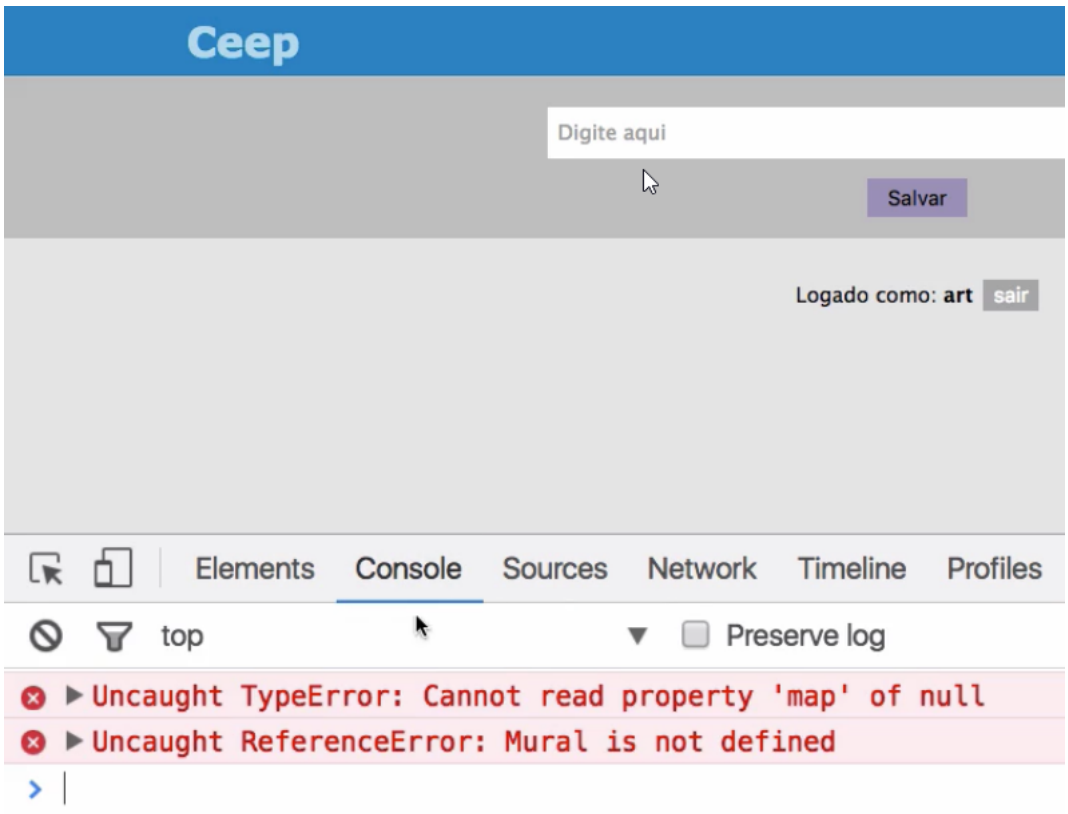
```
let logado = JSON.parse(localStorage.getItem("logado"))
let usuario

LoginUsuario_render({
  logado: logado
  ,usuario: localStorage.getItem("nomeUsuario")
  ,onLogin: (nomeUsuario) => {
    logado = true
    localStorage.setItem("logado", true)
    localStorage.setItem("nomeUsuario", nomeUsuario)
    usuario = nomeUsuario
  }
  ,onLogout: () -> {
    logado = false
    localStorage.setItem("logado", false)
    localStorage.removeItem("nomeUsuario")
  }
  })
```

A cada login, mudo a variável `nomeUsuario`. Lá no mural (`Mural.js`), portanto, preciso parar de chamá-la de "nomeUsuario", substituindo-a por `usuario`:

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

Se estivermos conseguindo logar no sistema, podemos ver os cartões sendo adicionados em locais diferentes. Vamos verificar?

Vou apagar todos os cartões que tinha antes, logado como "art" e adicionarei um cartão novo:



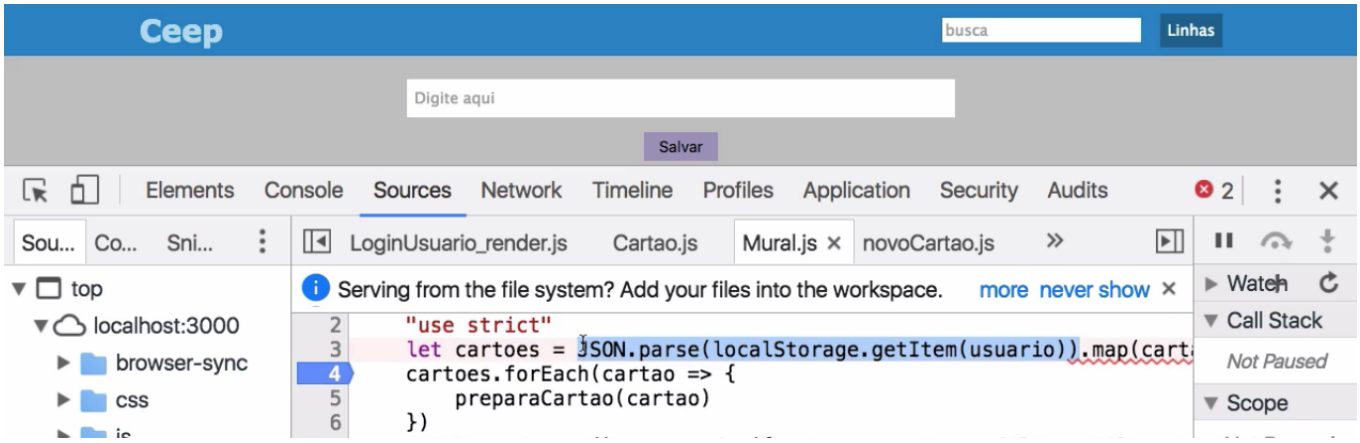
Deu um erro! O que ocorreu? Ele não conseguiu ler a propriedade "**map**" de null, porque ainda não fizemos alterações no `Mural.js`. No momento em que criamos a variável `cartoes`, pedimos a ele cartões sendo que precisamos antes disso pedir o nome do usuário (`usuario`) que estiver logado:

```
const Mural = function(_render, Filtro){
  "use strict"
  let cartoes = JSON.parse(localStorage.getItem(usuario)).map(cartaoLocal)
}
```

No entanto, quando carregamos a página, nossa variável `usuario` não possui nenhum valor. Precisamos pegar o valor do usuário que está salvo, inclusive, lá no `localStorage`. Assim como salvamos como `nomeUsuario`, pode-se utilizar o `getItem` para conseguirmos esse nome:

```
let logado = JSON.parse(localStorage.getItem("logado"))
let usuario = localStorage.getItem("nomeUsuario")

LoginUsuario_render({
  logado: logado
  ,usuario: localStorage.getItem("nomeUsuario")
  ,onLogin: (nomeUsuario) => {
    logado = true
    localStorage.setItem("logado", true)
    localStorage.setItem("nomeUsuario", nomeUsuario)
    usuario = nomeUsuario
  }
  ,onLogout: () -> {
    logado = false
    localStorage.setItem("logado", false)
    localStorage.removeItem("nomeUsuario")
  }
  })
```

Estamos tentando pegar o item chamado `usuario`, que é o valor da variável `nomeUsuario` ("art"), porém, não existe no `localStorage` nenhuma propriedade com esse nome. Ele nos diz no console: "não consigo ler a propriedade `map` de null", ou seja, o que retornou do `JSON.parse` é null.



Só posso criar minha lista de cartões quando aquilo que vem do `localStorage` existe de fato. Não posso afirmar que essa propriedade sempre existe. A nossa lógica para popular os cartões vai ficar um pouco complexa agora. Não posso criar uma linha só com a variável `cartoes`. Por ora ela fica sem valor.

A primeira coisa que de precisamos é tentar pegar nossos cartões lá do `localStorage`, e chamar nossos cartões de `cartoesLocal`:

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

Filtro.on("filtrado", render)

}
```

Se conseguir pegar os valores, esse `cartoesLocal` será nossa lista. Mas se não conseguir pegar um valor para o `usuario`, quer dizer que esse `cartoesLocal` é null, e isso não possibilitaria o `map`. É o que preciso verificar. Posso utilizar o `cartoesLocal.map` apenas quando `cartoesLocal` for uma lista.

Acrescentarei um `if()`, aquilo que decidirá se é possível rodar o map. Caso contrário, se não tiver `cartoesLocal` na variável `cartoes`, quero uma lista vazia:

```
const Mural = function(_render, Filtro){
  "use strict"
  let cartoes

  let cartoesLocal = JSON.parse(localStorage.getItem(usuario))
```

```
  if(cartoesLocal){
    cartoes = cartoesLocal.map(cartaoLocal => new Cartao(cartaoLocal.conteudo, cartaoLocal.tipo))
    } else {
      cartoes = []
    }
    cartoes.forEach(cartao => {
      preparaCartao(cartao)
      })
  }
const render = () => _render({cartoes: cartoes, filtro: Filtro.tagsETexto}); render()

Filtro.on("filtrado", render)

}
```

Dessa maneira, a lógica deve funcionar. Porém, criarmos essa variável denominada `cartaoLocal` a mais, ela está à disposição de todos aqui no `Mural.js` , mas ela é uma variável que é usada somente em um ponto específico do código. Podemos melhorar isto, então, dizendo que tanto o `cartoesLocal.map` quanto a lista vazia `cartoes = []` são valores associáveis ao `cartoes` sem necessidade de atribuição de valor novo para a variável, que toda essa lógica ocorre no momento de criação de `cartoes` . Podemos isolar esta lógica dentro da criação da variável. Como fazemos isso? Colocando toda essa nossa lógica dentro de uma função **if**. Qualquer código que coloquemos dentro dela será executado no instante em que essa linha for lida, e tudo será protegido.

Essa variável `cartoesLocal` só existe dentro desta função, que executo neste instante. Ao invés de mudar o valor da variável `cartoes` aqui, retorno um valor:

```
const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = (function(){
    let cartoesLocal = JSON.parse(localStorage.getItem(usuario))

    if(cartoesLocal){
      return cartoesLocal.map(cartaoLocal => new Cartao(cartaoLocal.conteudo, cartaoLocal.tipo))
      } else {
        return = []
      }
  })(



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

  Filtro.on("filtrado", render)

  }
```

Estamos transformando os cartões em resultados desta função, para isolar a lógica, simplificando assim a leitura. Dessa maneira, temos nossos cartões sendo a lista vazia ou aquilo que estiver na lista do meu usuário.

Recarregarei a página e tenho a lista do "art" que provavelmente aparece no `localStorage`. Vou deslogar e carregar a página de novo, não tem mais nenhum cartão aqui. Se logar como "art", os cartões ainda não vieram, porém se recarregar a página, eles aparecem. Existem algumas coisas que precisamos arrumar ainda, só consigo ver os cartões após recarregar a página, e o mesmo ocorre quando saio, o cartão só será removido depois que recarregar a página.

Seria muito legal se os cartões fossem removidos assim que me deslogasse e, além disso, quando trocasse de usuário, ele trouxesse os cartões deste novo usuário, removendo os do antigo assim que houvesse o logout do mesmo.

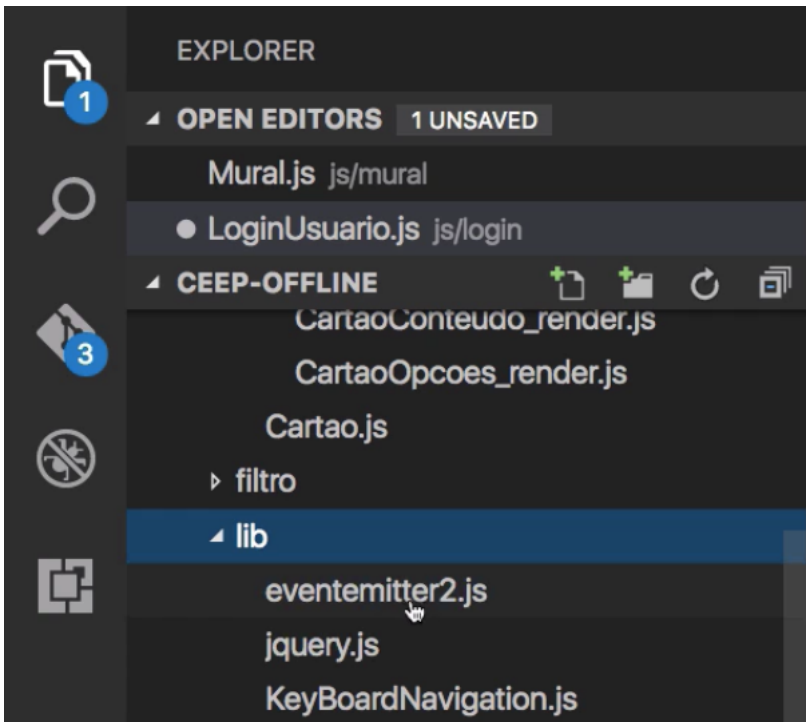
O máximo que nossa funcionalidade de login faz hoje em dia é nos dar uma variável `logado`, que é é usada no `Mural.js`, na função `adiciona()`. Não se sabe muito além da variável ou quando ela muda de valor. Para isso, precisaríamos ser avisados em nossa página, algo que nosso login não permite ainda. Isso é o que nossos cartões fazem, os quais foram implementados ao sistema de forma a nos avisar caso haja alguma mudança, o que possibilita que ouçamos esses eventos. De forma similar, conseguimos ouvir as mudanças quando removemos, trabalhamos com eventos aqui nos cartões.

Nosso login poderia trabalhar dessa forma, sem ter apenas uma variável `logado` e sim, além disso, quando alguém deslogar ou logar, termos acesso a essa informação, no mural.

Terei, então, alguém que possa ouvir os eventos de login e logout e realizar ações a partir disso.

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

login.on("login", ()=>{

  })

  login.on("logout", ()=>{

    })
```

Precisamos, então, ter uma variável `login` que consiga nos dizer quando houve login ou logout. Implementaremos isso dentro do arquivo `LoginUsuario.js`. Abro-o, vejo que já temos as variáveis `logado` e `usuario`, e agora teremos a `login`. Como faço algo que emita eventos? É complicado e, por isso, temos uma biblioteca chamada ***EventEmitter2***, na pasta lib do nosso projeto.



No arquivo `LoginUsuario.js`, temos então:

```
let logado = JSON.parse(localStorage.getItem("logado"))
let usuario = localStorage.getItem("nomeUsuario")
let login = new EventEmitter2()

LoginUsuario_render({
  logado: logado
  ,usuario: localStorage.getItem("nomeUsuario")
  ,onLogin: (nomeUsuario) => {
    logado = true
    localStorage.setItem("logado", true)
    localStorage.setItem("nomeUsuario", nomeUsuario)
    usuario = nomeUsuario
  }
  ,onLogout: () => {
    logado = false
    localStorage.setItem("logado", false)
    localStorage.removeItem("nomeUsuario")
  }
})
```

O que ela nos disponibiliza é uma função construtora. Ao chamá-la, terei um emissor de eventos, que nos permite ouvir eventos, como já estou fazendo no mural (`login.on`), e também emiti-los (login e logout). Quando isto acontece? Voltando ao código, procuro a localização de quando há logout (no callback de logout), tomando cuidado para utilizar as mesmas nomenclaturas encontradas no `Mural.js` .

Para evitar erros de digitação, sugere-se copiar e colar o trecho do código desejado.

```
let logado = JSON.parse(localStorage.getItem("logado"))
let usuario = localStorage.getItem("nomeUsuario")
let login = new EventEmitter2()
```

```
LoginUsuario_render({
  logado: logado
  ,usuario: localStorage.getItem("nomeUsuario")
  ,onLogin: (nomeUsuario) => {
    logado = true
    localStorage.setItem("logado", true)
    localStorage.setItem("nomeUsuario", nomeUsuario)
    usuario = nomeUsuario
    localStorage.removeItem("nomeUsuario")
    login.emit("logout")
  }
  ,onLogout: () => {
    logado = false
    localStorage.setItem("logado", false)
    localStorage.removeItem("nomeUsuario")
    login.emit("logout")
  }
})
```

No callback de login, quando o usuário logar, quero emitir o evento de mesmo nome. Estamos criando no login do usuário três coisas importantes: a variável que indica se ele está logado ou não (`logado`), a do usuário (`usuario`) e por fim, a do emissor dos eventos (`login`).

Quando não tiver ninguém deslogando, quero que minha lista de cartões fique vazia, então o que podemos fazer é acrescentar `cartoes = []` no local apropriado em `Mural.js` , chamando também a função `render()` para que essas alterações apareçam na tela:

```
login.on ("login", ()=>{

  })

login.on("logout", ()=>{
  cartoes = []
  render()
  })
```

Esta função já aparece lá no começo do código, e ela não renderiza nenhum cartão, pois estou esvaziando minha lista. Vamos ver se está funcionando? Atualizo minha página, faço login como "art", adiciono novos cartões e, assim que fizer logout da página, nenhum cartão deverá ser exibido. Está funcionando, conseguimos remover os cartões do usuário que não estiver logado no momento.

Para que o usuário tenha acesso aos seus cartões que se encontram no `localStorage` ao logar, acrescento mais algumas linhas no código de `Mural.js` , utilizando também a variável `usuario` , disponível no login:

```
login.on ("login", ()=>{
  localStorage.getItem(usuario)
  })

login.on("logout", ()=>{
  cartoes = []
  render()
  })
```

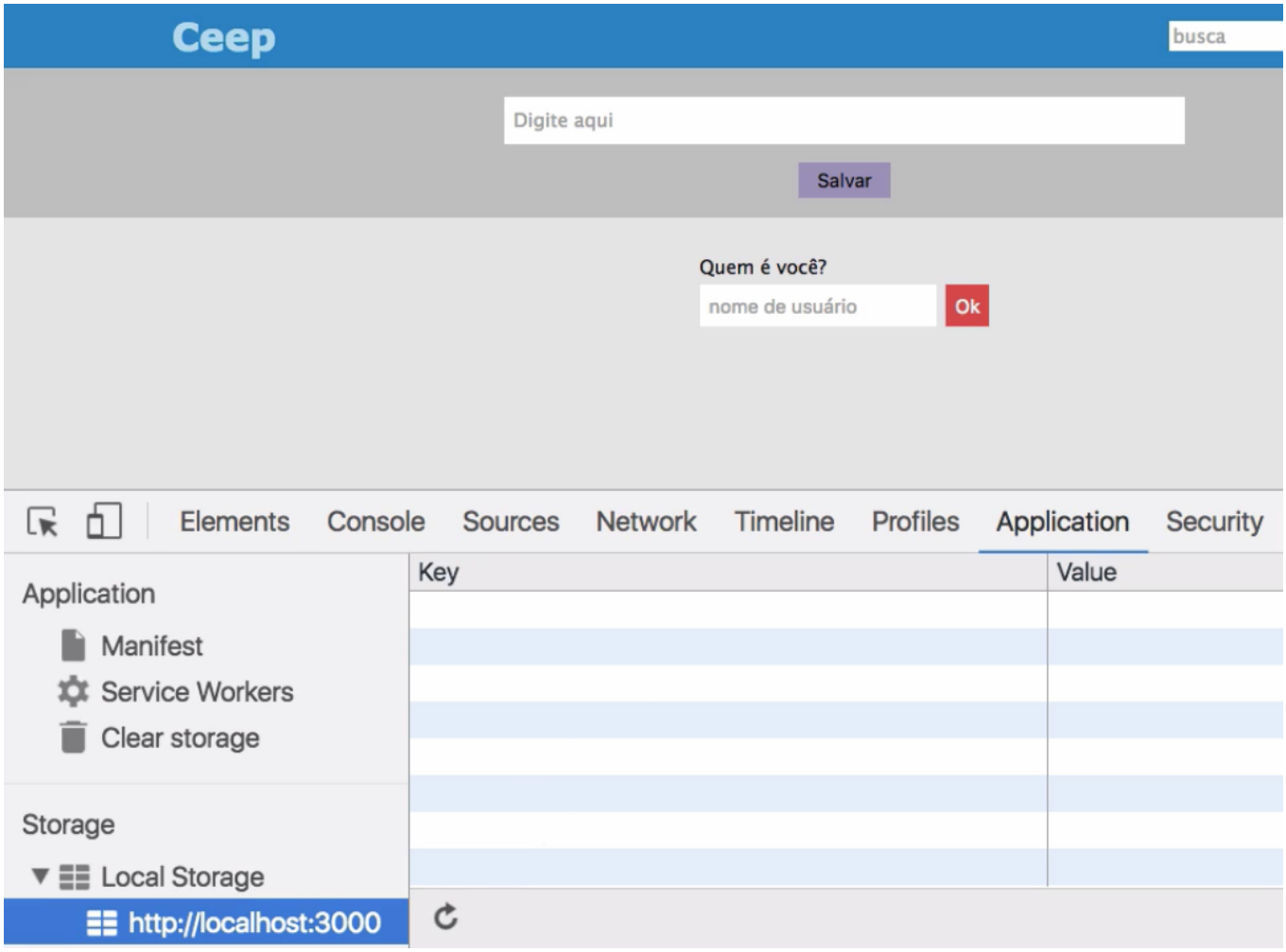
Precisamos usar o `JSON.parse` ainda; na verdade, ainda precisamos fazer tudo aquilo que estávamos fazendo dentro da nossa função **if** dos cartões. Quando a página carregava, pegávamos os cartões do `localStorage`, verificávamos se havia cartões, cuja resposta negativa nos retorna uma lista vazia. Caso contrário, fazíamos o mapa. Toda essa lógica precisa ser repetida no momento de login do usuário. Já que será a mesma lógica, posso dizer que isolarei tudo isso numa função (`pegaCartoesUsuario`):

```
const Mural = (function(_render, Filtro){
  "use strict"
  let cartoes = pegaCartoesUsuario()

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

  Filtro.on("filtrado", render)

  function pegaCartoesUsuario(){
    let cartaoLocal = JSON.parse(localStorage.getItem(usuario))
    if(cartaoLocal){
      return cartoesLocal.map(cartaoLocal => new Cartao(cartaoLocal.conteudo, cartaoLocal.tipo))
      } else {
        return = []
      }
    }
```

Esta pode ser usada tanto quando a página é carregada quanto quando o usuário estiver logado. Não precisamos mais refazer a lógica toda:

```
login.on ("login", ()=>{
  cartoes = pegaCartoesUsuario()
  })

login.on("logout", ()=>{
  cartoes = []
  render()
  })
```

Assim, minha variável `cartoes` receberá, no começo, os cartões do usuário logado, e quando ele logar novamente com outro nome, será executado de novo. Vamos dar uma olhada no browser, abrindo a aba *Application* e dando uma limpada em tudo, para fazermos o fluxo do zero:



Logarei como usuário "art" e cadastrarei alguns cartões. Quando sair (deslogar), estou removendo os cartões anteriormente criados. Quando entrar novamente como "art", espero que estes mesmos cartões reapareçam. Não foi o que aconteceu. O que houve? A propriedade "art" está correta, porém não solicitamos a renderização da página após essa última modificação. Corrigimos isso acrescentando uma linha com `render()`:

```
login.on ("login", ()=>{
  cartoes = pegaCartoesUsuario()
  render()
})

login.on("logout", ()=>{
  cartoes = []
  render()
})
```

Vamos recarregar a página, sair, limpar todo o `localStorage`, logar novamente como "art". Agora sim, ele está conseguindo trazer os cartões quando faço login, retirando-os quando faço logout. O mesmo acontece ao logarmos como outro usuário qualquer. Realizamos nosso objetivo!