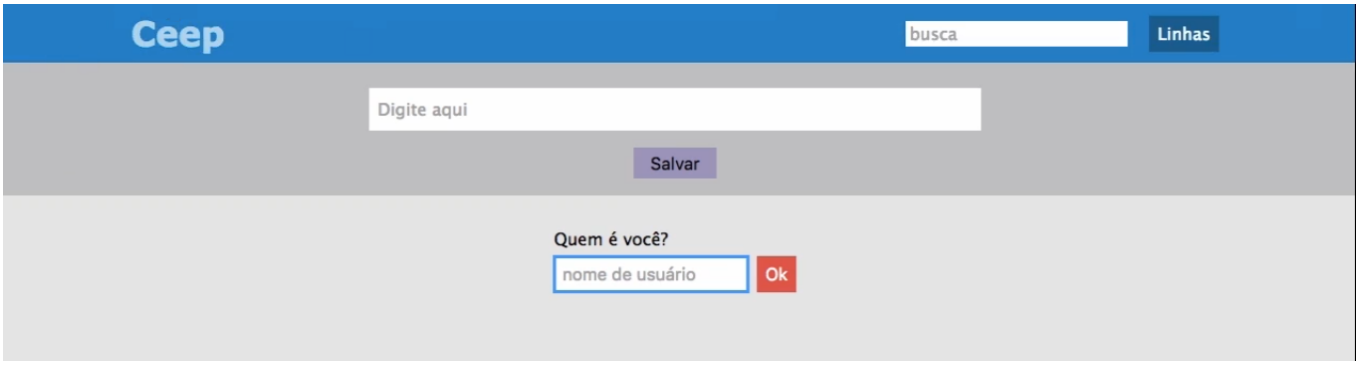
02

Logando o usuário com localStorage

Transcrição

Se atualizarmo a página, veremos o formulário de login novamente.



Se não fizermos nenhuma mudança, toda vez que o sistema for atualizado, perderemos o login anterior. Como isso é um pouco chato, podemos fazer uma melhoria, para que o sistema armazene o nome do usuário quando ele logar.

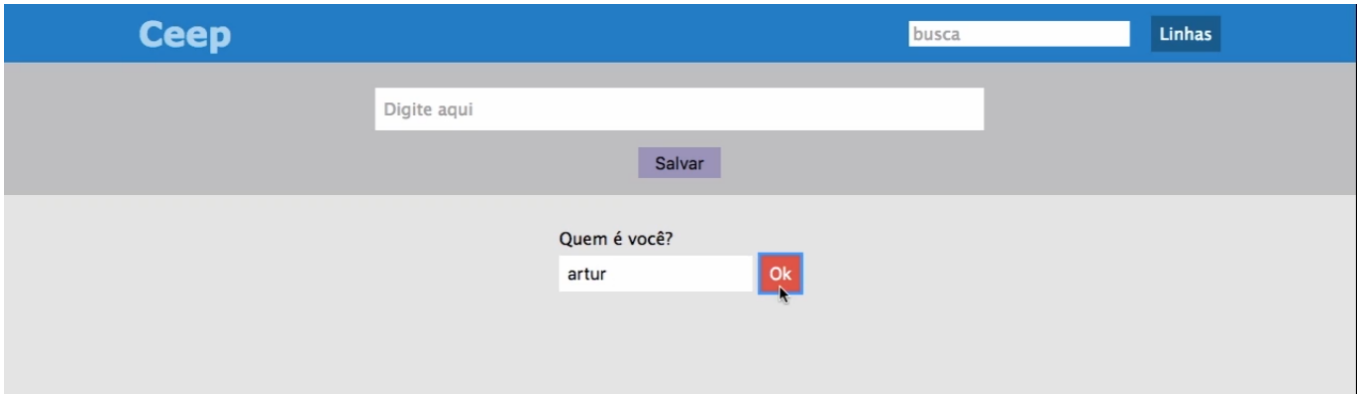
Ou seja, tendo `logado` como `true` ou `false`, precisamos de mais informação. Precisamos dar um jeito de armazenar offline o nome de quem se logou pela última vez. Para isso, usaremos o `localStorage()`, que será o responsável por armazenar informações no browser.

```
let logado = false

LoginUsuario_render({
  logado: false
  ,onLogin: () => {
    logado = true
    localStorage()
  }
  ,onLogout: () => logado = false

  })
```

Precisamos dar um nome para a coisa que queremos salvar, que é se o usuário está logado ou não. Bem parecido com a nossa variável, se ela for `true`. Assim:

```
let logado = false

LoginUsuario_render({
  logado: false
  ,onLogin: () => {
    logado = true
    localStorage("logado", true)
  }
  ,onLogout: () => logado = false

  })
```

É assim que salvaremos os dados. Se voltarmos agora ao sistema, e tentarmos fazer o login, teremos uma surpresa.



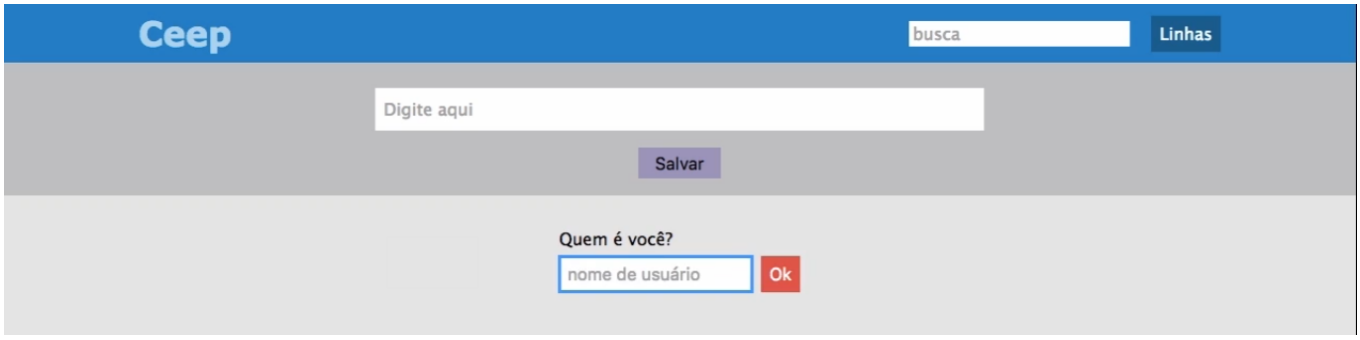
Ele não está funcionando! Isso acontece porque com o `localStorage` podemos pegar um dado ou inseri-lo. Precisamos especificar, e usaremos o `setItem` para isso.

```
let logado = false

LoginUsuario_render({
  logado: false
  ,onLogin: () => {
    logado = true
    localStorage.setItem("logado", true)
  }
  ,onLogout: () => logado = false

  })
```

Voltando à aplicação, tentaremos logar como `art` :

Logado como art (https://s3.amazonaws.com/caelum-online-public/Progressive+Web+Apps/Aula+2/Aula2.1_25_logado-como-art.png)

Funcionou! Será que quando recarregarmos a página estará funcionando?

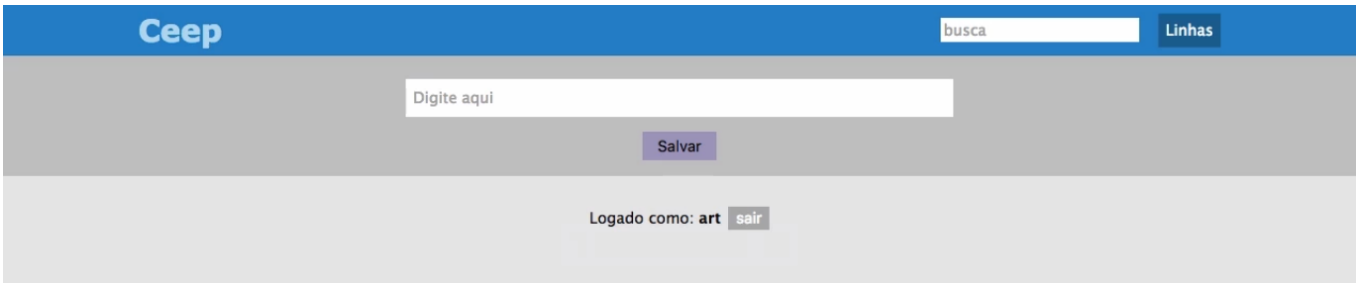


Só pedimos para os dados serem armazenados. Não avisamos que queremos que ele veja o último usuário logado quando atualizarmos a página. Para isso, será preciso alterar o valor da variável `logado` . E como podemos mexer em alguma coisa quando a página carrega?

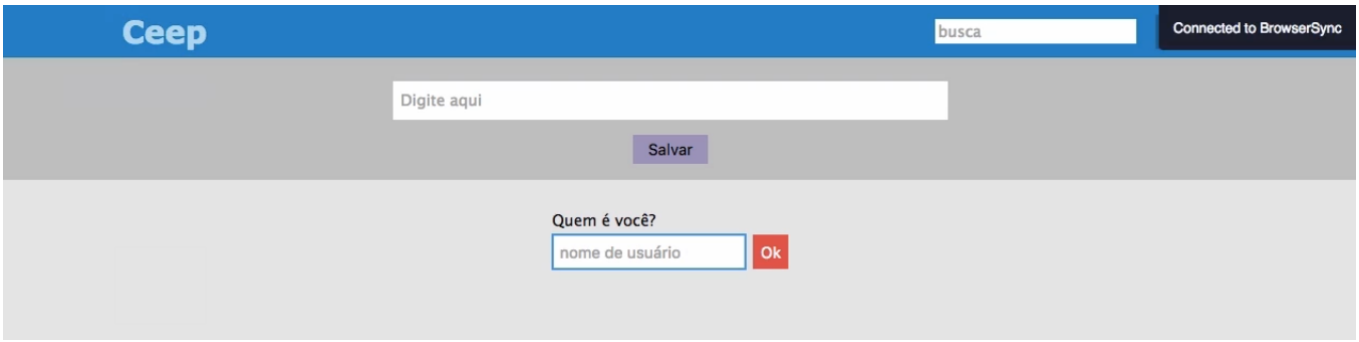
Ao recarregar, tudo o que está nesse arquivo será lido. Assim, a linha `let logado = false` é lida assim que a página carregada. Mas, agora não queremos que se leia que o usuário não está logado, mas que se verifique isso com o `localStorage` . Como precisamos pegar um valor, usaremos `getItem` .

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

LoginUsuario_render({
  logado: false
  ,onLogin: () => {
    logado = true
    localStorage.setItem("logado", true)
  }
  ,onLogout: () => logado = false

  })
```

Desse jeito, já estamos armazenando se o usuário estava logado, e pegando o seu valor quando a página carrega. Vamos testar!



Estamos logados como `artur`. Recarregando:



Deveríamos ter continuado logados, mas ele nos mostra o formulário de login novamente. O que será que está acontecendo no código? Nós alteramos o valor da variável `logado`, mas não renderizamos o seu valor. No `LoginUsuario`, o valor de `logado` é sempre `false`. Precisamos que `logado` tenha o valor da variável aí dentro.

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

LoginUsuario_render({
  logado: logado
  ,onLogin: () => {
    logado = true
    localStorage.setItem("logado", true)
  }
  ,onLogout: () => logado = false

  })
```

Se ele conseguiu armazenar o login no `setItem`, quando recarregarmos a página o sistema pegará o valor do `logado` e irá mostrar que ainda estamos logados.

O sistema nos mostra que estamos "Logados como", mas não mostra o meu nome! Se eu sair, e digitar meu nome novamente, veremos:

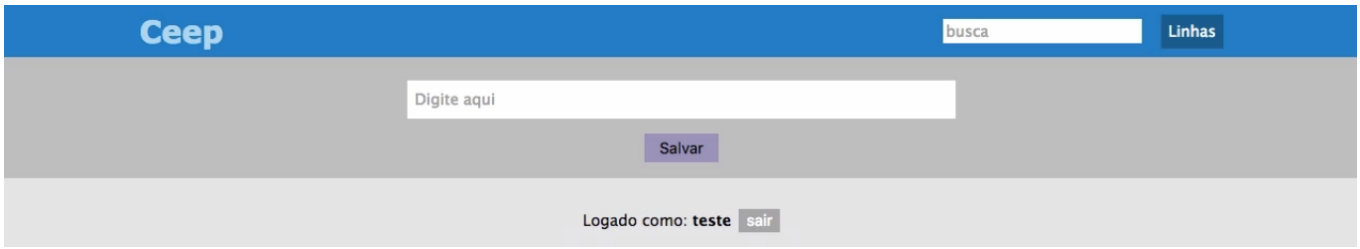
Ele nos mostra nosso nome de usuário. Mas, quando recarregamos, o nome desaparece.

Quando o login do usuário é renderizado, ele não sabe quem era o usuário que estava lá antes. Para passar qual é esse usuário, precisamos de mais uma propriedade, `usuario`.

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

LoginUsuario_render({
  logado: logado
  ,usuario: art
  ,onLogin: () => {
    logado = true
    localStorage.setItem("logado", true)
  }
  ,onLogout: () => logado = false

  })
```

Quando atualizarmos, ele nos mostrará que quem logou foi o `art`.

Vamos logar com um nome diferente agora, o `teste`. Depois do login, o nome aparece certinho.



Mas, quando atualizamos a página, o nome que aparece é `art`.

Precisamos armazenar mais informações do usuário, para que isso não aconteça. Não basta saber se ele está logado ou não, é preciso saber quem é. Vamos assumir que pegaremos (`getItem`) essa informação do `localStorage`

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

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

  })
```

Os nomes das propriedades não possuem nenhum padrão, podemos escolhê-los de acordo com o que for melhor para nós. Esperamos que o `nomeUsuario` esteja sendo salvo. E para isso acontecer, precisamos dar um `setItem`. Mas que valor devemos colocar?

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

LoginUsuario_render({
  logado: logado
```

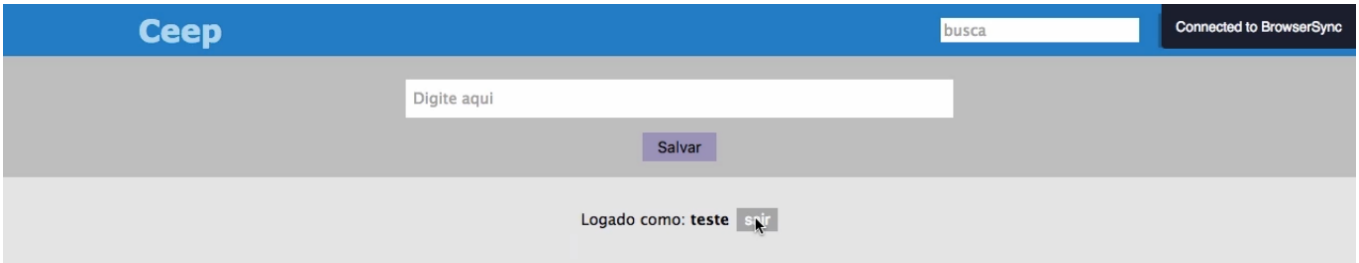
```
  ,usuario: localStorage.getItem("nomeUsuario")
  ,onLogin: () => {
    logado = true
    localStorage.setItem("logado", true)
    localStorage.setItem("nomeUsuario", true)
  }
  ,onLogout: () => logado = false

  })
```

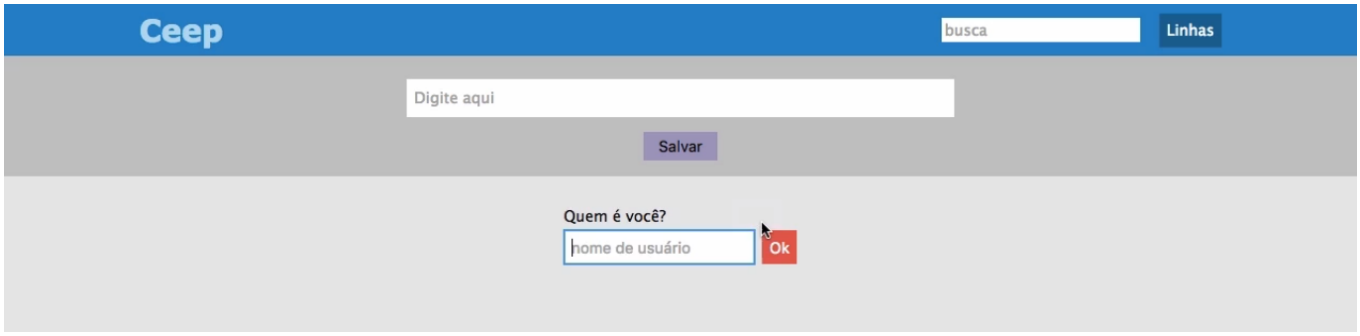
O nome do usuário aparece lá na interface. Podemos pedir esse nome no callback do login, como um parâmetro. E usaremos esse parâmetro como valor para o `localStorage`.

```
let logado = 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

  })
```

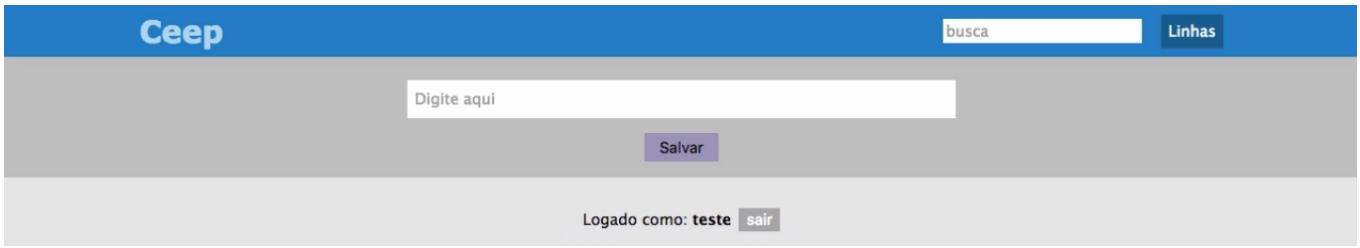
Quando a página recarregar, ainda estaremos logados, e o `usuario` vai pedir o valor para o `localStorage`. Vamos testar novamente, logando como `teste` e recarregando.

Continuamos logados como `teste`, então o login funcionou offline! O usuário poderá recarregar quantas vezes quiser. E se sairmos desse usuário?





Tentaremos recarregar essa página antes de logar. Veja só o que acontece:



Ele ainda está logado como `teste`. O que acontece é que definimos a lógica de só armazenar quando usuário fizer o login, mas não definimos nada para quando ele sair. É preciso avisar de alguma forma que não há mais usuário logado. As propriedades `nomeUsuario` e `logado` devem ser alteradas não só no `onLogin`, mas também no `onLogout`.

Para isso, podemos pegar as mesmas linhas do `onLogin`, alterando apenas os seus valores. Se o `logado` era `true`, agora será `false`. E para o `nomeUsuario`? Podemos colocar um texto vazio, mas fica ainda melhor com um `undefined`. Assim, é como se não existisse um nome de usuário.

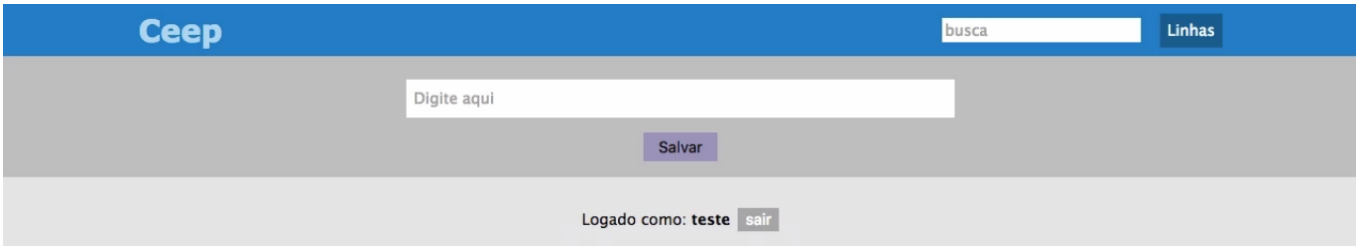
```
let logado = localStorage.getItem("logado")

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.setItem("nomeUsuario", undefined)
  })
```

Outra alternativa, é tirar o nome de usuário do `localStorage`, com o `removeItem`.

```
let logado = 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")
    })
```

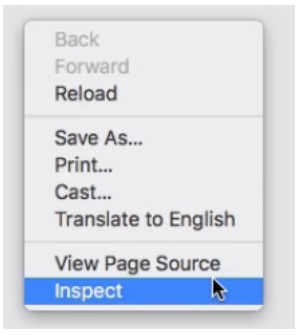
Vamos testar na interface. Ainda estamos logados como `teste`, e quando atualizamos, continuamos logados.



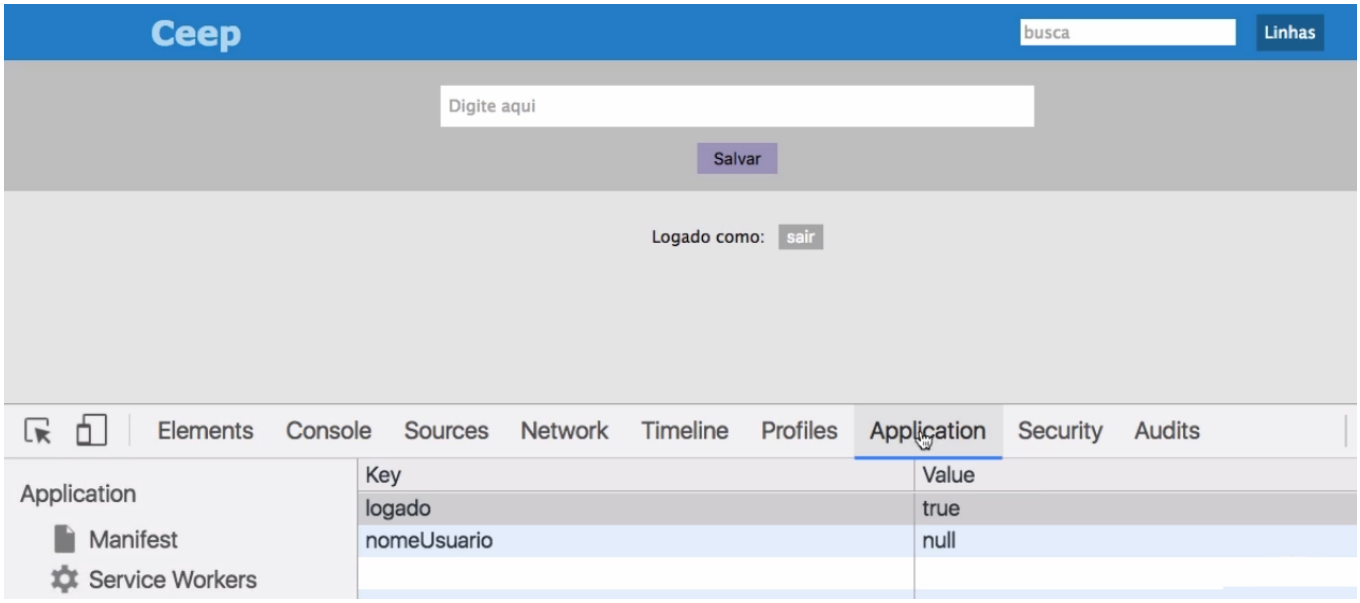
Quando clicamos em sair e atualizamos, vemos:

O nome do usuário saiu, mas a mensagem "Logado como" persiste. Isso nos mostra que o `true` e `false` pro `logado` não funcionou. Pode ser que o `setItem` esteja lendo algo errado no navegador, então teremos que dar uma olhada no que está sendo armazenado no `localStorage`.

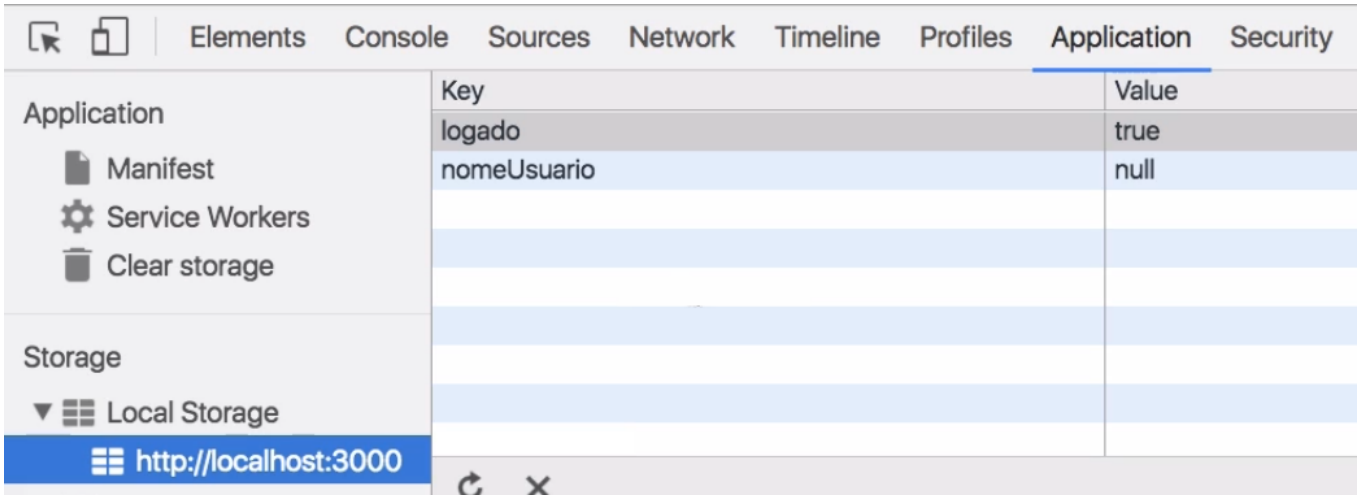
Para isso, abriremos as ferramentas do desenvolvedor do navegador, cujo atalho é `Command + Shift + I`. Também é possível clicar com o botão direito e em `Inspect`.



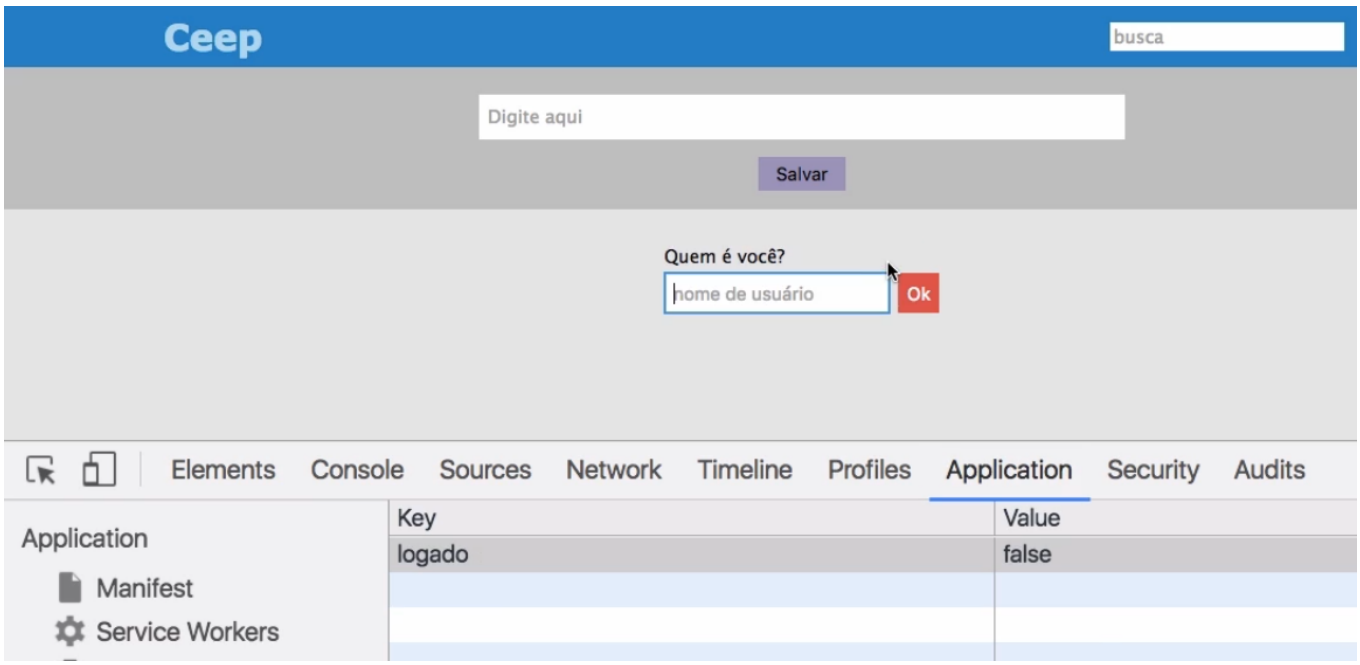
Aberta essa parte, podemos ir para a aba `Application`.



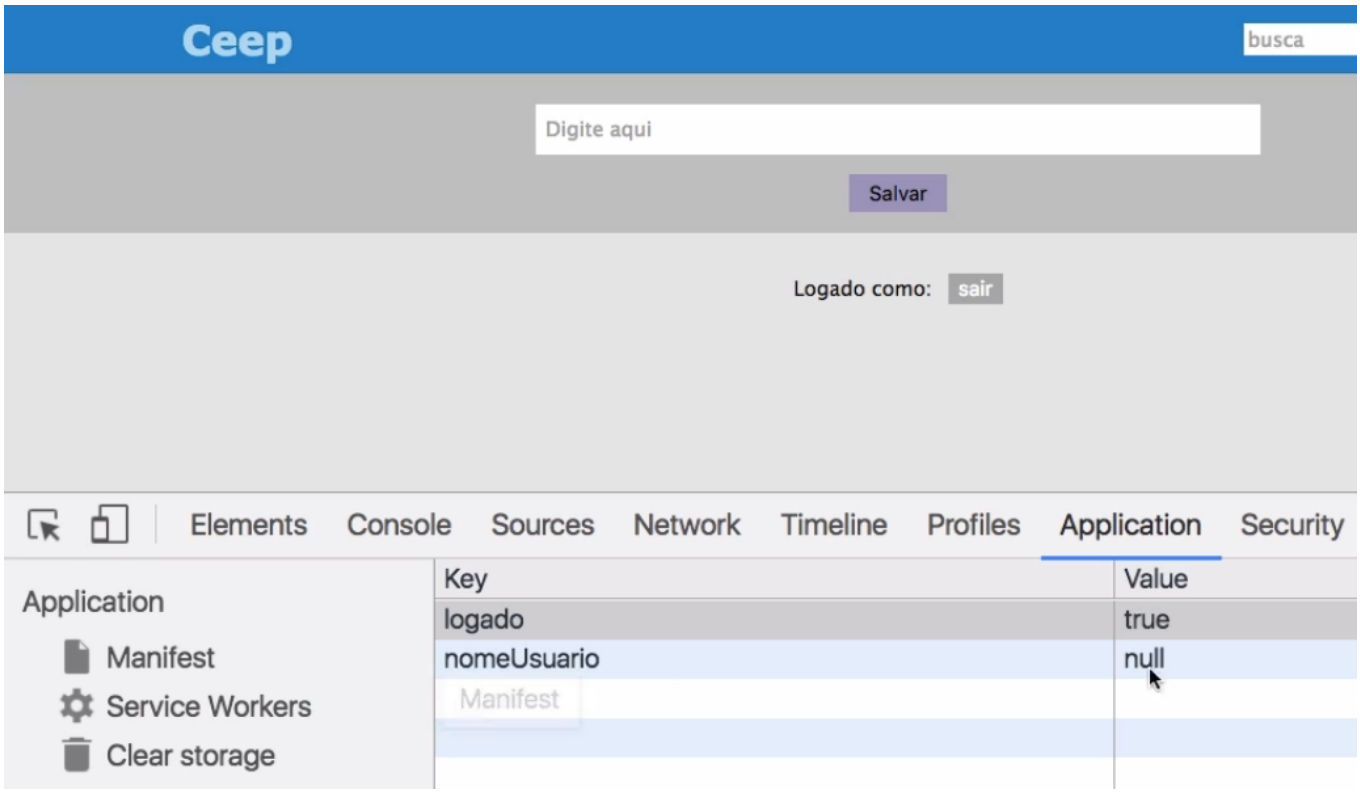
Nessa aba podemos ver tudo o que está armazenado offline para fazer a app funcionar. Iremos na parte de `Local Storage`, e entrar no nosso site. Lá, veremos as informações armazenadas, aqui chamadas de `keys`.



Essas chaves são `logado` e `nomeUsuario`, e os `values` (valores) são os que atribuímos a eles. Se nós sairmos, veja o que acontece:



O valor de `logado` passa a ser `false` e o nome do usuário já não está lá.



Ficamos logados novamente, mesmo com o valor do `logado` sendo `false`. O `localStorage` tem uma peculiaridade, que vamos ver agora, na aba `Console`. Perguntaremos a ele qual o valor de `logado`.

```
localStorage.getItem("logado")
"false"
```

Ele prontamente nos respondeu `"false"`. Mas note que está dentro de aspas, o que significa que ele é um texto. No JavaScript, isso é uma string, que você pode usar texto como Boolean. Sempre que passamos a variável `logado` para o `LoginUsuario_render`, ele recebe uma string, que poderá ser `"true"` ou `"false"`. Dentro do console, podemos testar os seus valores boleanos.

```
localStorage.getItem("logado")
"false"
Boolean("true")
true
Boolean("false")
true
```

Para a nossa surpresa, o valor de `"false"` é `true`. Desde que não seja uma string vazia, o seu valor será `true`.

```
localStorage.getItem("logado")
"false"
Boolean("true")
true
Boolean("false")
true
Boolean("")
false
```

Então, precisamos transformar esse valor `"false"` em um valor boleano de verdade. Precisamos converter um trecho de texto em um JavaScript. A função `Boolean` nos mostrará sempre o valor verdadeiro do texto, que será sempre `true`. Então usaremos outra função, o `JSON`, cujo objetivo é transformar texto em código JavaScript. Faremos um `JSON.parse`, que é a função que permitirá que nosso valor seja efetivamente boleano e tenha o efeito desejado.

Vamos testar no console.

```
localStorage.getItem("logado")
"false"
Boolean("true")
true
Boolean("false")
true
Boolean("")
false
JSON.parse("false")
false
```

Voltando ao nosso código, teremos:

```
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")
  })
```