06

Salvando as imagens do cartão com Cache Storage

Transcrição

No último vídeo vimos o funcionamento do **Cache Storage**, e o que faremos agora é adicionar as imagens dos cartões do mural nesse cache. Cada um deles pode conter uma ou várias imagens, que também são textos:

Esta é a informação a que temos acesso por enquanto: o usuário coloca uma url de uma imagem, que precisa ser armazenada no nosso Cache Storage. Fazer isto imagem por imagem é um processo demorado, uma vez que é possível também mesclar texto e imagem nos cartões. É preciso, portanto, uma função que selecione essas urls.

Tal função existe dentro do arquivo `Cartao.js`. Trata-se da `Cartao.pegaImagens`, que recebe como parâmetro um cartão, a partir do qual nos retorna uma lista com todas as urls das imagens pertecentes a ele.

```
Cartao.pegaImagens = cartao => {
  return (cartao.conteudo.match(/\!\[(.+?)\]\((.+?)\)/g) || []).map(imagemMD => {
    return imagemMD.match(/\!\[.+?\]\((.+?)\)/)[1] || null
    })
}

  return Cartao
})(Cartao_render, EventEmitter2, TiposCartao)
```

De que maneira temos acesso aos cartões do mural? Já havíamos feito isso antes para salvar arquivos no `localStorage`. A questão é que agora quero salvá-los ali e também no Cache Storage. É o que faremos no `Mural.js`, em que temos várias funções com acesso aos cartões, sendo o `adiciona()` um deles. Esta função é chamada apenas quando o usuário adiciona cartões através da página principal. Quando esta é carregada pela primeira vez, os cartões vêm da função `pegaCartoesUsuario()`, o qual carrega o conteúdo para o `localStorage`.

Em algum lugar do `Mural.js` existe uma função que sempre tem acesso aos cartões... É a `preparaCartao`, por onde todas as funções relacionadas ao conteúdo do mural passam, é disso que precisamos. Acrescentaremos a linha `const urlsImagens = Cartao.pegaImagens(cartao)`, então, no arquivo `Mural.js`:

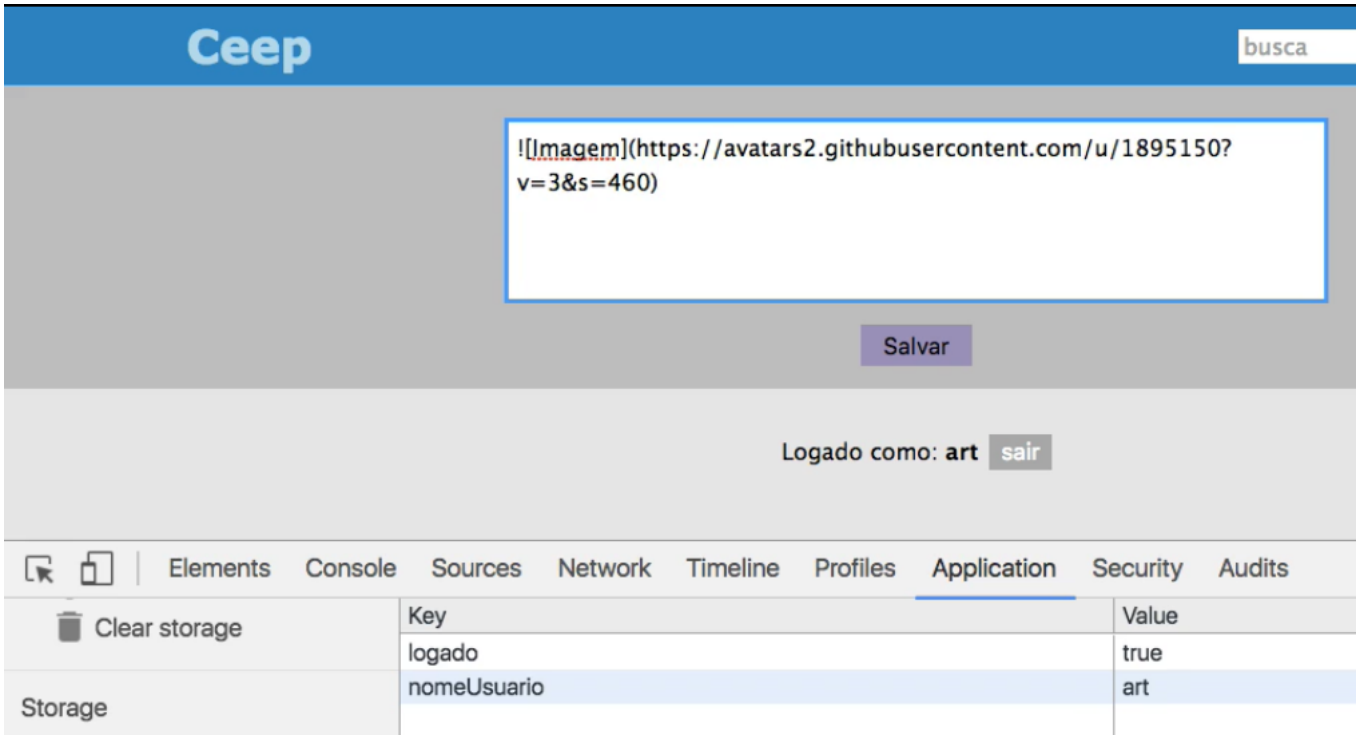
```
function preparaCartao(cartao){
  const urlsImagens = Cartao.pegaImagens(cartao)
}
```

O parâmetro utilizado é `cartao`, o qual criará uma lista de urls. O que precisamos colocar no Cache Storage? A url e a imagem correspondete que retorna a partir dela. Precisamos portanto carregá-la. Para cada url que tiver, quero que haja carregamento, possível pela função `fetch`. Quando a imagem for carregada, terei uma resposta, e o `caches.open` acessa a imagem do `ceep-imagens`, tendo sua url disponibilizada pelo `cache.put`. O segundo parâmetro é o próprio arquivo, a resposta:

```
function preparaCartao(cartao){
  const urlsImagens = Cartao.pegaImagens(cartao)
  urlsImagens.forEach(url => {
    fetch(url).then(resposta) => {
      caches.open("ceep-imagens").then(cache => {
        cache.put(url, resposta)
        })
    })
})

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

Se recarregar a página e adicionar um cartão com imagem, podemos esperar que haja alguma informação no Cache Storage, que foi previamente limpo, similarmente ao que foi feito no Local Storage, já que estou começando do zero. Faço o login como usuário "art", adicionando ao cartão minha imagem de perfil usando a sintaxe necessária.

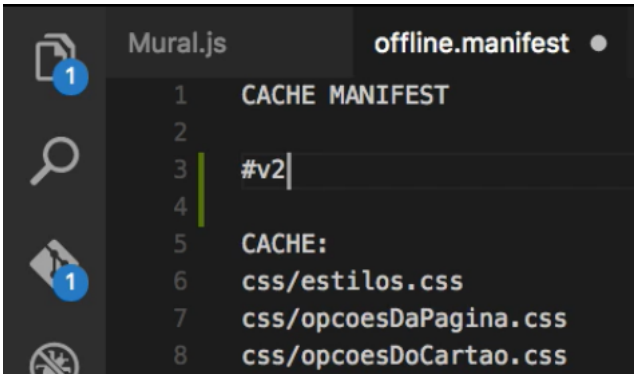


Vou adicionar o cartão, clicar em "Salvar", e o Local Storage estará funcionando perfeitamente. E o Cache Storage? Ele permanece vazio mesmo após a atualização da página. Para verificar se nosso código está funcionando de forma correta, vou à aba "Sources" e abro o arquivo `Mural.js` .

Vemos que a função `preparaCartao` está inalterada, embora tenhamos adicionado várias linhas de código anteriormente! Em exercícios prévios, começamos a utilizar o Application Cache, o qual salva em si o arquivo `Mural.js` , que permanece o mesmo independentemente das alterações feitas nele mesmo. Sempre que o HTML pede o `Mural.js` através da tag `script` , invariavelmente ele o pegará do Application Cache.

O que acontece é que, caso se queira atualizar os arquivos da nossa app, mais especificamente os da Application Cache, precisamos definir isto explicitamente no arquivo `offline.manifest` . Por padrão, se este não for alterado, nenhum outro será carregado novamente, ou seja, se quisermos que as alterações presentes em `Mural.js` influenciem a página, precisamos que eles sejam recarregados pelo Application Cache. É bastante comum não termos arquivos novos para serem adicionados, e que só queiramos que os mesmos sejam recarregados.

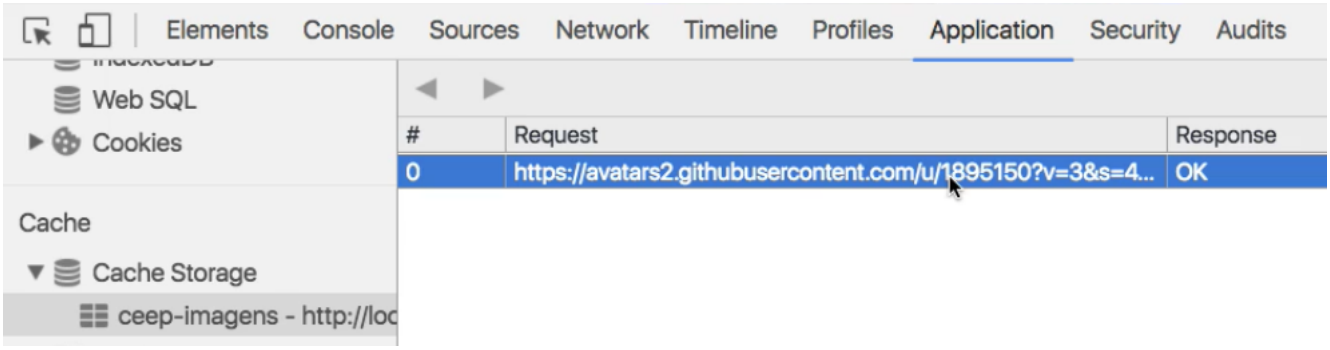
Para que isto seja feito, basta colocarmos um comentário no `offline.manifest` utilizando uma *hashtag* com uma palavra, "update" por exemplo, para que haja atualização sem o acréscimo de arquivos novos. Por estarmos trabalhando com uma Web App, existe aquela ideia de instalação de arquivos, e muitas pessoas costumam indicar no comentário uma atualização de versão (" `#v2` ").



Quando atualizo o `offline.manifest` , então, podemos verificar o que ocorre lá no Console, recarregando a página, o qual baixa todos os arquivos novamente. Porém, ao checarmos nosso `Mural.js` vemos que a função `preparaCartao` ainda não foi atualizada.

Precisamos lembrar sempre de que o Application Cache de fato atualizou os arquivos, mas enquanto o usuário estiver mexendo na página, ele não pode mudar todos os arquivos. Somente no próximo *reload* (atualização da página) é que os arquivos serão atualizados. Farei isto e verificarei no `Mural.js` da aba "Sources". Agora, sim, a função foi atualizada.

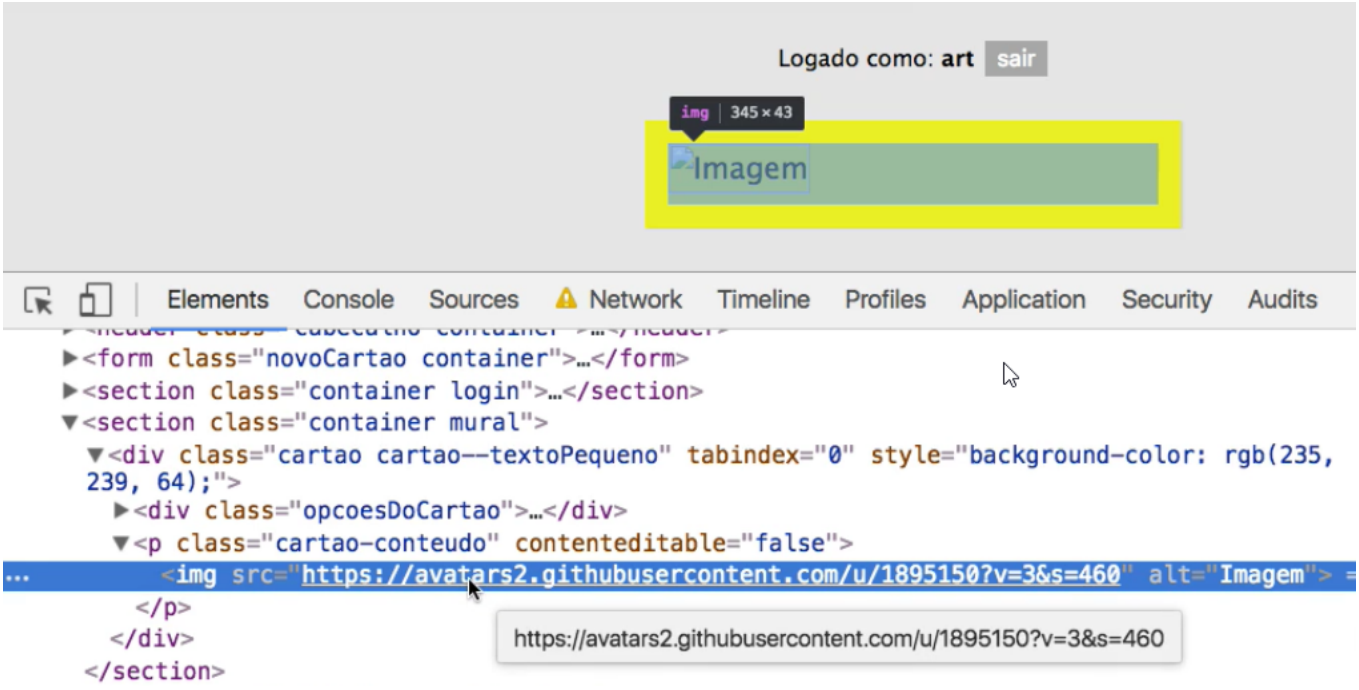
Isto quer dizer que a função foi rodada, o cartão foi adicionado à página e ele já foi alterado no Cache Storage. Carregando a página mais uma vez, vê-se que algo aconteceu ali. Confirmo clicando em `ceep-imagens` :



Agora sim, nossa imagem foi "cacheada". O que fizemos foi atualizar nosso arquivo para poder modificar o Cache Storage, assim como já tínhamos feito no vídeo anterior. Simplesmente alterar o arquivo em si não foi o suficiente. Utilizar o Cache Storage não necessariamente confirma o uso do Application Cache, importante para a funcionalidade offline. O que o Cache Storage faz, por sua vez, é armazenar as imagens, os arquivos dinâmicos que variam conforme a especificidade definida pelo usuário.

Vamos habilitar a opção de deixarmos a aplicação em uso offline indo à aba "Network", limpando-a e recarregando a página. A imagem não apareceu! Ao consultarmos o Cache Storage, verifica-se que está tudo OK. Por quê a imagem não aparece quando estou offline?

Por mais que tenhamos salvo a nossa imagem no Cache Storage, isso não garante que a imagem esteja sendo procurada ali. Quando o `src` tenta acessar a tag `img` na nossa página, no cartão criado por nós, ele fará o pedido dessa url:



Essa busca pela imagem no Cache Storage não é automática, sendo diferente do que acontece, por exemplo, com nossos `script`s no `index.html`. Cada tag dessas, e também `link`, `src`, são carregadas através do Application Cache. A partir desse comportamento, de transformar a página em uma aplicação com os arquivos necessários instalados, a ideia é de que ele o faça automaticamente. Dessa forma, tudo que está listado no `offline.manifest` e, consequentemente, no Application Cache, é carregado automaticamente, sem que precisemos fazer nada. O Cache Storage, por outro lado, não faz isto.

Se quisermos utilizar a imagem que colocamos nele, precisaremos criar mais códigos, o que não quer dizer que ele seja pior, apenas que é diferente, tratando-se de um "poder" inclusive, algo que veremos no próximo vídeo.