JSON e localStorage

Antes de partimos para as funcionalidades e conceitos do localStorage, precisamos entender o que é um JSON.

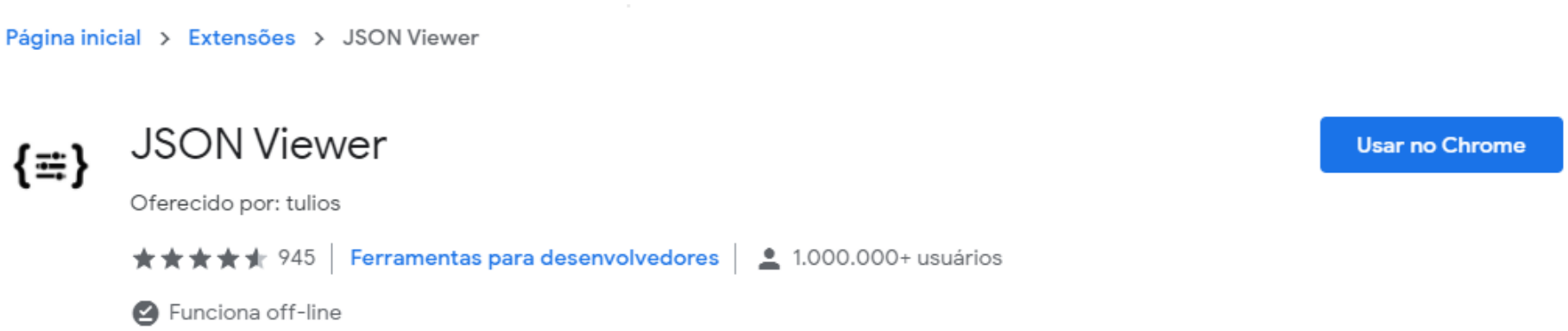
O que é um JSON? →

O JSON ou JavaScript Object Notation é um formato leve de troca de informações/dados entre sistemas, além de ser leve também é de simples leitura, trata-se de um formato compactado, de padrão aberto independente, criado por Douglas Crockford em 2000 de maneira a deixar trocas de dados entre sistemas muito mais rápidas. Muito utilizado em diversas aplicações que necessitam as trocas de dados constantes.

Antes de visualizarmos um JSON em si, temos uma dica de extensão do Google Chorme chamada **JSON Viewer**. Para adicionarmos em nosso navegador acesse:

- https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh?hl=pt-br;

Aparecerá a seguinte tela:



Clique em **Usar no Chrome → Adicionar extensão.**

Essa extensão nos ajuda a visualizar um JSON de modo geral.

Agora que já sabemos o que é um JSON, podemos visualizar um exemplo:

- https://api.github.com/users/faculdadeiv2/repos;

Qual a definição de JSON? →

Um JSON é muito parecido com o texto de um objeto ou array em JavaScript, sendo que a principal diferença é que esse pode conter apenas tipos primitivos (object, array, number, string, boolean). Funções, por exemplo, não podem ser anotadas em formato JSON.

Um JSON é nada mais do que uma representação textual de um objeto ou array, e portanto pode ser muito parecida com o que você verá ao digitar console.log() em um array ou objeto. Isso torna o formato JSON, uma excelente alternativa para transmissão de dados, uma vez que o tipo textual é compreendido quase que universalmente entre as linguagens de programação e a internet.

Qual a funcionalidade de um JSON?

Existem dois métodos principais quando falamos de JSON, o primeiro método irá transformar cadeias de dados com `Arrays` e `Objetos` em JSON, a segunda irá transformar os dados do formato JSON em um Array, contendo dados no formato de posições.

Por que fazemos isso? Normalmente quando falamos de APIs (Application Programming Interface, não se preocupe com isso agora) ou de consumo de Bancos de Dados de maneira geral, queremos encapsular nossos dados e mandá-los da maneira mais rápida e compacta possível, o JSON faz exatamente isto, encapsula nossos dados, fazendo com que tudo seja feito da maneira mais rápida possível.

Ficou com dúvidas ou não entendeu algo? Vamos resumir um pouco para você:

- **J**ava**S**cript **O**bject **N**otation;

- Formato para transmissão de dados;
- Sua sintaxe é basicamente um texto escrito no formato de um Objeto;
- Como possui apenas texto, pode facilmente ser enviado entre computadores;

JSON.stringify() - Objeto → JSON

Como já dito, conseguimos transformar Objetos em JSON, e o método usado para isso é o JSON.stringify(), ele converte os valores para uma String JSON.

Para um melhor entendimento, veja o exemplo a seguir:

```
// Javascript:

const carro = {
  nome: 'hilux',
  ano: 2010,
  ipvaOK: true
}
```

Criamos um objeto qualquer para uso, perceba suas propriedades e seus respectivos valores:

- nome: string;
- ano: number;
- ipvaOK: boolean;

Agora, usaremos o método em questão:

```
const carroJSON = JSON.stringify(carro);
```

1 → Criamos uma constante chamada 'carroJSON', responsável por guardar nosso JSON assim que gerado;

2 → Fazemos a chamada do objeto JSON, o JavaScript já reconhece que isso é um objeto automaticamente;

3 → Como já chamamos o objeto, várias opções são disponibilizadas e então escolhemos .stringify(objetoOuArray).

4 → Passamos como parâmetro a cadeia de dados (um objeto nesse caso) que queremos tornar um JSON, portanto, JSON.stringify(carro)

Como resultado no console temos:

```
console.log(carroJSON);

// {"nome":"hilux","ano":2010,"ipvaOK":true}
```

Agora, todas as propriedades são strings, porém perceba que os dados não foram alterados, ou seja, string continua sendo string, number continua como number e boolean continua boolean.

Comparado ao objeto em si temos o seguinte resultado:

```
const carro = {
  nome: 'hilux',
  ano: 2010,
  ipvaOK: true
}

const carroJSON = JSON.stringify(carro);

console.log('Objeto: ', carro);

// Objeto:  { nome: 'hilux', ano: 2010, ipvaOK: true }

console.log('JSON: ', carroJSON);

// JSON:  {"nome":"hilux","ano":2010,"ipvaOK":true}
```

JSON.stringify() - Array → JSON →

Basicamente funciona da mesma maneira que o exemplo acima, porém com uma pequena diferença que veremos. Da mesma maneira que encapsulamos o Objeto e o transformamos em JSON, iremos agora fazer a mesma coisa com um Array:

```
const carro = [
  {
    nome: 'hilux',
    ano: 2010,
    ipvaOK: true
  },
  {
    nome: 'sw4',
    ano: 2020,
    ipvaOK: false
  }
]
```

É um simples array com dois objetos que usaremos no método .stringify:

```
const carroJSON = JSON.stringify(carro);
```

Por enquanto continua igual ao processo com Objetos. Mas para relembrar:

1 → Criamos uma constante chamada 'carroJSON', responsável por guardar nosso JSON assim que gerado;

2 → Fazemos a chamada da classe JSON, o JavaScript já reconhece que isso é uma classe automaticamente;

3 → Como já chamamos a classe, várias opções são disponibilizadas e então escolhemos .stringify(objetoOuArray);

4 → Passamos como parâmetro a cadeia de dados (um array nesse caso) que queremos tornar um JSON, portanto, JSON.stringify(carro);

Como resultado no console temos:

```
console.log(carroJSON);

// [{"nome":"hilux","ano":2010,"ipvaOK":true},{"nome":"sw4","ano":2020,"ipvaOK":false}]
```

Perceba o detalhe diferente:

- Enquanto formamos um JSON de um Objeto com a instrução `{}` para delimitar início e fim do JSON em si, com Array utilizamos a instrução `[]`.

Nesse meio as propriedades também se tornam strings e seus dados mantém seus valores sem alteração (string → string, number → number, boolean → boolean).

Comparado ao array em si temos o seguinte resultado:

```
const carro = [
  {
    nome: 'hilux',
    ano: 2010,
    ipvaOK: true
  },
  {
    nome: 'sw4',
    ano: 2020,
    ipvaOK: false
  }
]

const carroJSON = JSON.stringify(carro);

console.log('Array: ', carro);
/*
Array:  [
  { nome: 'hilux', ano: 2010, ipvaOK: true },
  { nome: 'sw4', ano: 2020, ipvaOK: false }
]
*/

console.log('JSON: ', carroJSON);
// JSON:  [{"nome":"hilux","ano":2010,"ipvaOK":true},{"nome":"sw4","ano":2020,"ipvaOK":false}]
```

JSON.parse() →

Como já vimos anteriormente, JSON é uma classe com seus próprios métodos dentro de JavaScript, o segundo método que iremos ver é o método que trás de volta um elemento ao seu formato original, por exemplo, temos um Array que foi transformado em JSON, esse JSON foi mandado para um servidor e nosso colega de trabalho tem que consumir esse dado lá da máquina dele, como ele faz isso?

Veremos então como ele conseguirá fazer isso utilizando o método .parse() presente na classe JSON:

```
// Javascript:

// 1:
const carro = [
  {
    nome: 'hilux',
    ano: 2010,
    ipvaOK: true
  },
  {
    nome: 'sw4',
    ano: 2020,
    ipvaOK: false
  }
];

// 2:
const carroJSON = JSON.stringify(carro);

// 3:
console.log('Array de objetos: ', carro)

// 4:
console.log('JSON: ', carroJSON)

// 5:

// 6:

// 7:
const carrosArray = JSON.parse(carroJSON)

// 8:
console.log('Array: ', carrosArray)
```

Passo a passo explicado:

- 1 → Array com os objetos que iremos utilizar;
- 2 → Constante que guardará nosso JSON assim que gerado chamada 'carroJSON'. Utilizamos a classe JSON e passamos por parâmetro a cadeia de dados que estamos utilizando (o array nesse caso);
- 3 → Mostra o Array no console:

```
/* Retorna:
Array de objetos:  [
  { nome: 'hilux', ano: 2010, ipvaOK: true },
  { nome: 'sw4', ano: 2020, ipvaOK: false }
]
*/
```

- 4 → Mostra o JSON gerado:

```
/* Retorna:
JSON:  [{"nome":"hilux","ano":2010,"ipvaOK":true},{"nome":"sw4","ano":2020,"ipvaOK":false}]
*/
```

- 5 → Vamos imaginar que nessa etapa fizemos algo para mandar esses dados no formato de JSON para o servidor, não se preocupe com como fizemos isso agora, apenas imagine.
- 6 → Nessa etapa nosso colega já consumiu os dados do nosso servidor, novamente, não se preocupe em como ele fez isso, só imagine.
  Agora ele precisa transformar esses dados (JSON) em um Array para trabalhar na aplicação.
- 7 → Constante que guardará nosso Array transformado a partir do JSON recebido pelo método .parse().

- 8 → Mostra o Array no console:

```
/* Retorna:
Array:  [
  { nome: 'hilux', ano: 2010, ipvaOK: true },
  { nome: 'sw4', ano: 2020, ipvaOK: false }
]
*/
```

Bom, agora que já sabemos o que é JSON e como usá-lo, veremos o tópico sobre LocalStorage.

LocalStorage →

> localStorage consiste em salvar, adicionar, recuperar ou excluir dados localmente em um navegador, esta informação é guardada na forma de pares de chave-valor e os valores podem ser apenas strings.

O que é localStorage? →

O localStorage ou armazenamento na Web, é uma maneira de oferecer métodos e protocolos para armazenar dados do lado do cliente aos aplicativos Web. Basicamente podemos dizer que é como os dados de uma aplicação são salvos no navegador do usuário final. Os dados salvos nesse local não expiram, diferentemente por exemplo do sessionStorage, que tem seus dados limpos assim que a sessão da página for expirada.

Quais as vantagens de utilizá-lo? →

Para a aplicação, a vantagem principal é o fato de salvar os dados na própria máquina do usuário, ou seja, não é necessário um banco de dados, tornando assim a manutenção da aplicação mais barata.

Já para o usuário, a maior vantagem é gerir seu próprio localStorage, por exemplo, apagar dados inseridos incorretamente ou dados salvos anteriormente, diretamente da sua máquina, sem que tenha que obrigatoriamente comunicar-se diretamente com o servidor da aplicação.

Estrutura base →

Ao falarmos de localStorage teremos dois parâmetros a serem analisados, o primeiro será a chave e o segundo parâmetro é o valor:

- Chave: é basicamente onde a informação é salva. Por exemplo: temos a chave chamada '`pessoas`', onde podemos salvar diversos dados, como a `pessoa` Clóvis, `pessoa` Lucas, posteriormente a `pessoa` Otávio e assim por diante. Perceba que são pessoas diferentes, por isso a chave `pessoas`.
- Valor: é o dado que contém as informações de algo. No exemplo acima, possuímos a chave `pessoas`, então podemos dizer que as pessoas inseridas nela são valores, como 'pessoa `Clóvis` → `Clóvis` é um valor'.

Principais métodos →

Podemos utilizar alguns métodos presentes no localStorage para salvar dados por exemplo. Seus 4 principais métodos são:

- setItem();
- getItem();
- removeItem();
- clear();

---

setItem() →

Responsável por salvar informações dentro do localStorage. Sua sintaxe é:

```
localStorage.setItem('chave', JSON.stringify(valor));
```

Vendo isso da primeira vez pode parecer um pouco difícil, portanto, vamos ver um passo a passo do que aconteceu:

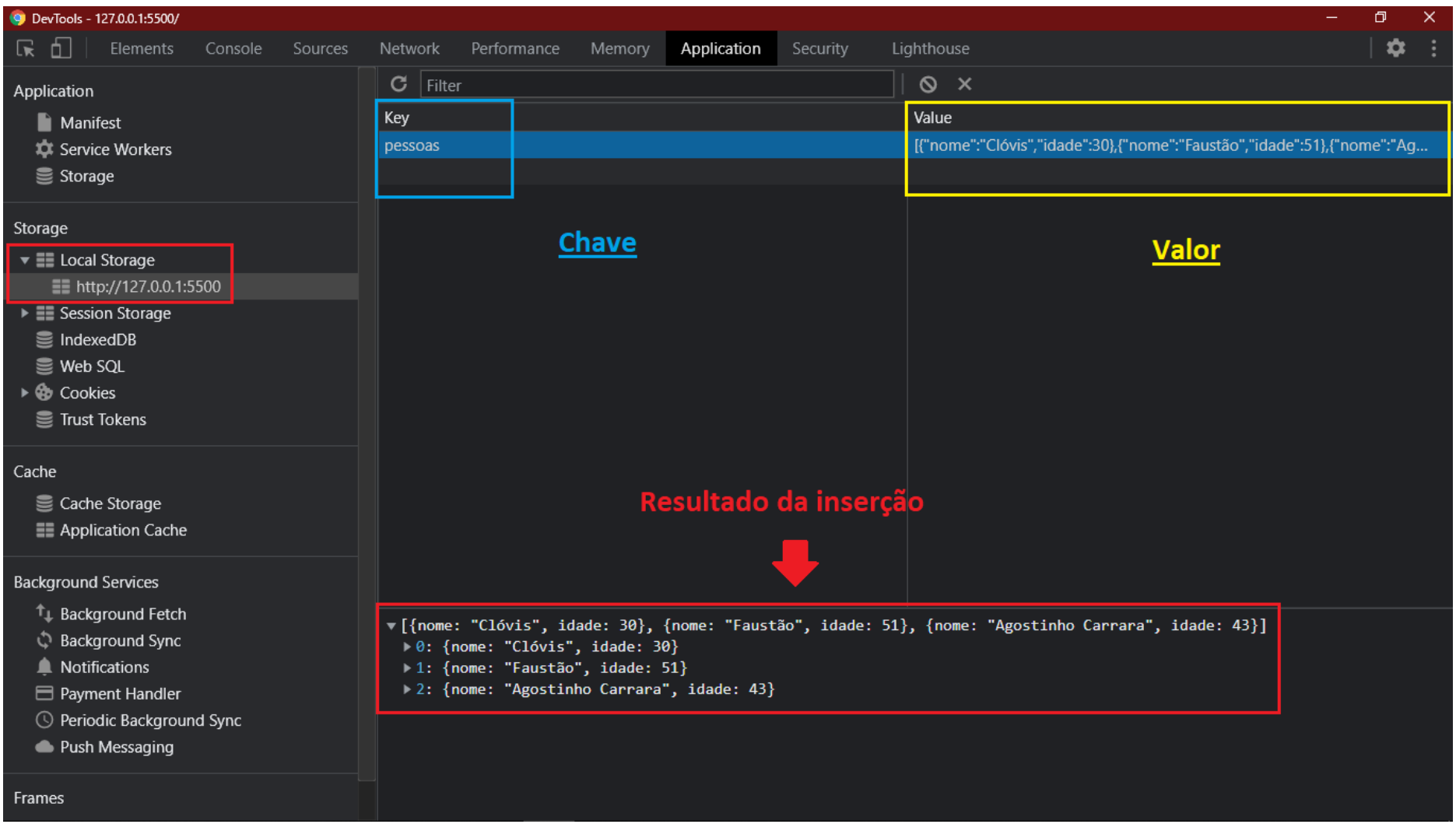
- 1 → Iniciamos o consumo do localStorage a partir da sintaxe '`localStorage`';
- 2 → Utilizamos o método setItem presente em localStorage para instruir que iremos fazer a função de salvar uma informação, esse método irá obrigatoriamente aguardar 2 `pararâmetros`: o primeiro será correspondente à chave e o segundo será referente ao valor;
- 3 → Passamos como primeiro `parâmetro` a chave, que referencia a qual local dentro do localStorage que iremos salvar as informações (utilizamos neste caso a palavra **chave** somente para ilustrar o que ocorre, poderia ser por exemplo: cidades, animais, pessoas, carros, etc...);
- 4 → Passamos como segundo `parâmetro` o valor, que referência qual o dado que queremos salvar dentro do localStorage, note que estamos fazendo uso de uma outra classe que já aprendemos, o JSON;
- 5 → JSON nesse caso faz com que as informações sejam encapsuladas, desta maneira, podemos mandar de maneira mais efetiva os dados para o localStorage, é sempre uma boa prática utilizar o JSON quando se quer enviar dados ao localStorage.

Um exemplo para facilitar sua compreensão:

```
const Pessoas = [
  {
    nome: "Clóvis",
    idade: 30,
  }, {
    nome: "Faustão",
    idade: 51
  }, {
    nome: "Agostinho Carrara",
    idade: 43
  }
]

function adicionarItem() {
  localStorage.setItem('pessoas', JSON.stringify(Pessoas));
}

adicionarItem();
```

- Primeiramente temos nosso array com objetos;
- Depois criamos uma função responsável por salvar as informações no locaStorage. Como chave usamos 'pessoas' e como valor usamos o Array Pessoas (perceba o uso do método stringify para transformarmos o mesmo em JSON);

O resultado será:



getItem() →

Esse método é responsável por consumir informações do localStorage. Sua estrutura é:

```
const algumaCoisa = JSON.parse(localStorage.getItem("chave"))
```

Vendo isso da primeira vez pode parecer um pouco difícil, portanto, vamos ver um passo a passo do que aconteceu:

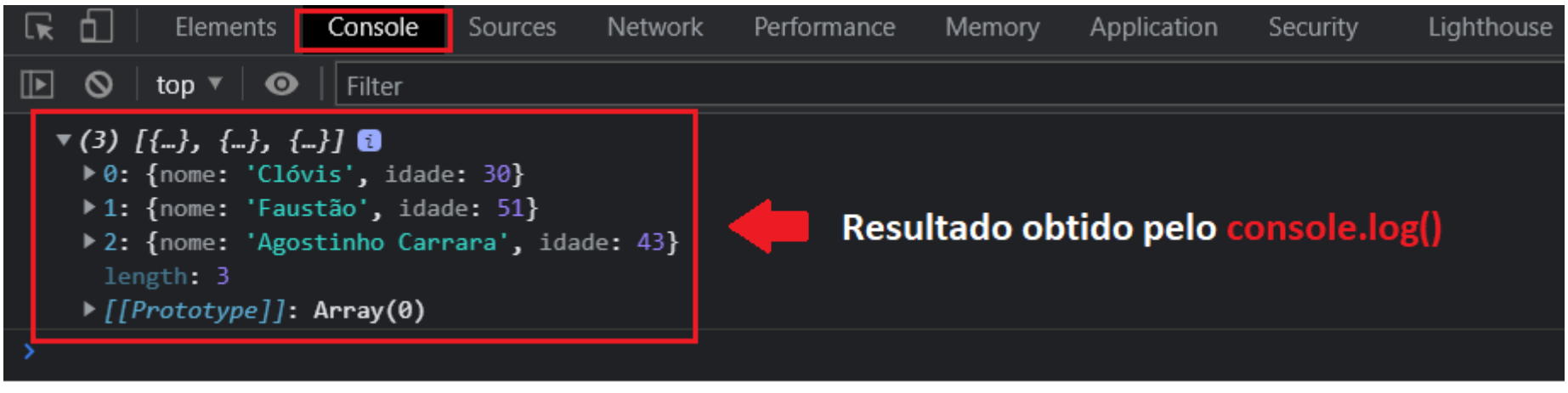
- 1 → Criamos a constante com o nome de algumaCoisa que é a variável onde iremos salvar nosso dado vindo do '`localStorage`';
- 2 → Chamamos a classe JSON com seu método .parse() que irá basicamente fazer a leitura e transformação de nosso dado advindo do localStorage, para ele se tornar um Objeto ou Array;
- 3 → Chamamos o localStorage e utilizamos o método .getItem() tendo como parâmetro a chave que salvamos nosso valor, uma vez feito isso temos salvo o valor desta chave na nossa variável algumaCoisa.

Agora veremos a união desse método com o método anterior (setItem() e getItem()):

```
const Pessoas = [
  {
    nome: "Clóvis",
    idade: 30,
  }, {
    nome: "Faustão",
    idade: 51
  }, {
    nome: "Agostinho Carrara",
    idade: 43
  }
]

function adicionarItem() {
  localStorage.setItem('pessoas', JSON.stringify(Pessoas));
}

adicionarItem();

function buscarItem() {
  const objPessoas = JSON.parse(localStorage.getItem('pessoas'))
  console.log(objPessoas);
}

buscarItem();
```

- A lógica necessária para salvar dados no localStorage é exatamente igual ao exemplo do método setItem().
- Depois dessa parte responsável por salvar, iniciamos a busca do item a partir da função buscarItem(). Ela é responsável por atribuir o valor obtido à uma constante chamada objPessoas. Nós buscamos o item desejado passado apenas passando a chave requisitada ('pessoas').
- Perceba o uso do método parse() para transformarmos o JSON do localStorage em um Array.

O resultado será:



---

removeItem() →

Responsável por remover o item com a chave recebida do localStorage caso exista. Sua sintaxe é:

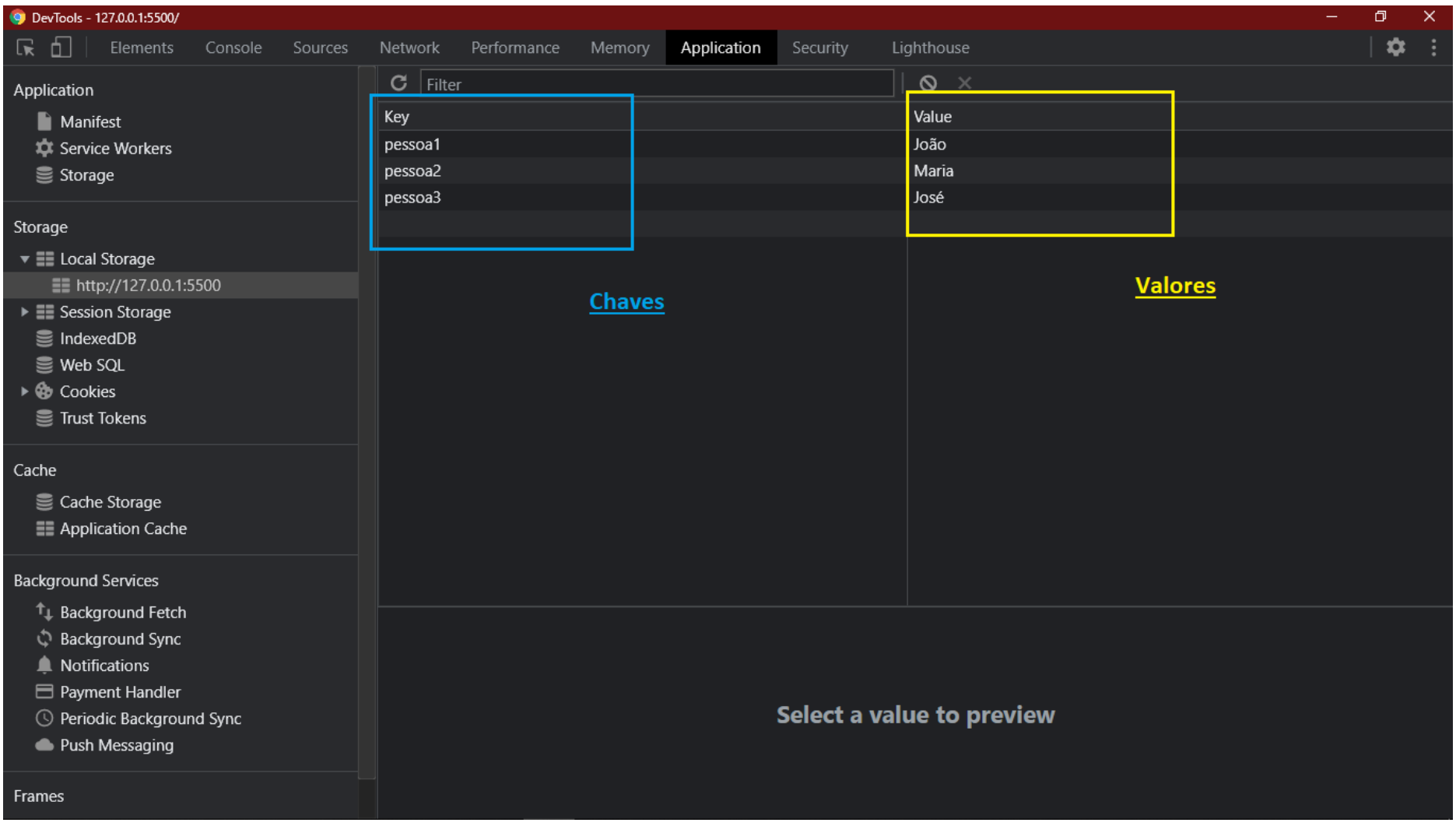
```
localStorage.removeItem("chave");
```

Para entendermos completamente iremos adicionar novos itens no localStorage e remover alguns para exemplo:

```
function adicionarItem() {
  localStorage.setItem('pessoa1', 'João');
  localStorage.setItem('pessoa2', 'Maria');
  localStorage.setItem('pessoa3', 'José');
  console.log(localStorage);
}

adicionarItem();

function removerItem() {
  localStorage.removeItem('pessoa2');
  console.log(localStorage);
}

removerItem()
```

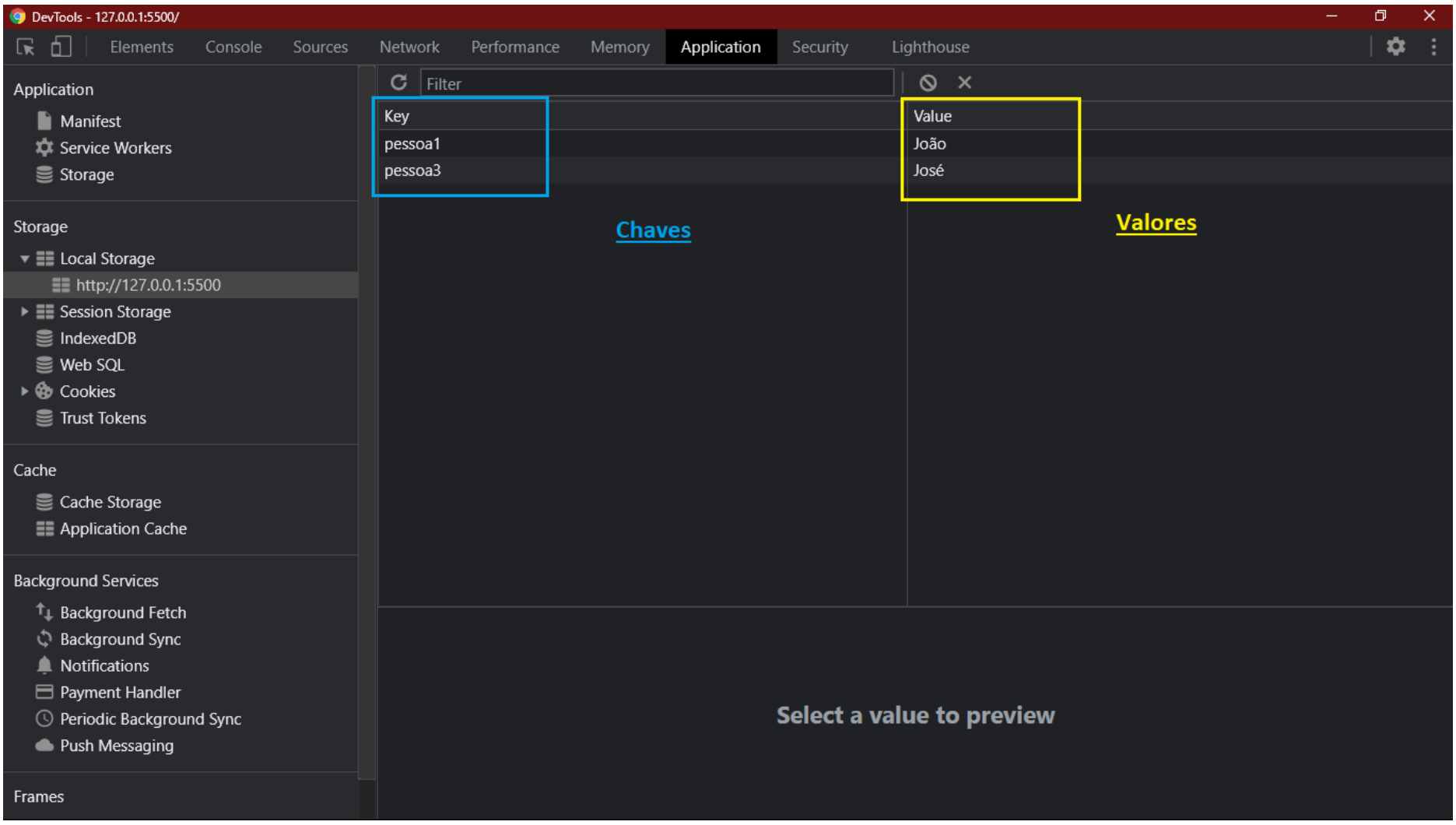
- Na função adicionarItem() adicionamos alguns itens de exemplo e retornamos o localStorage no console. O resultado trago será:

```
// Storage {pessoa1: 'João', pessoa2: 'Maria', pessoa3: 'José', length: 3}
```



- Já na função removerItem() excluímos o item com chave igual a 'pessoa2' e logo após retornamos o resultado:

```
// Storage {pessoa1: 'João', pessoa3: 'José', length: 2}
```



- Perceba que removemos o item do localStorage após passarmos a chave do item requisitado.

---

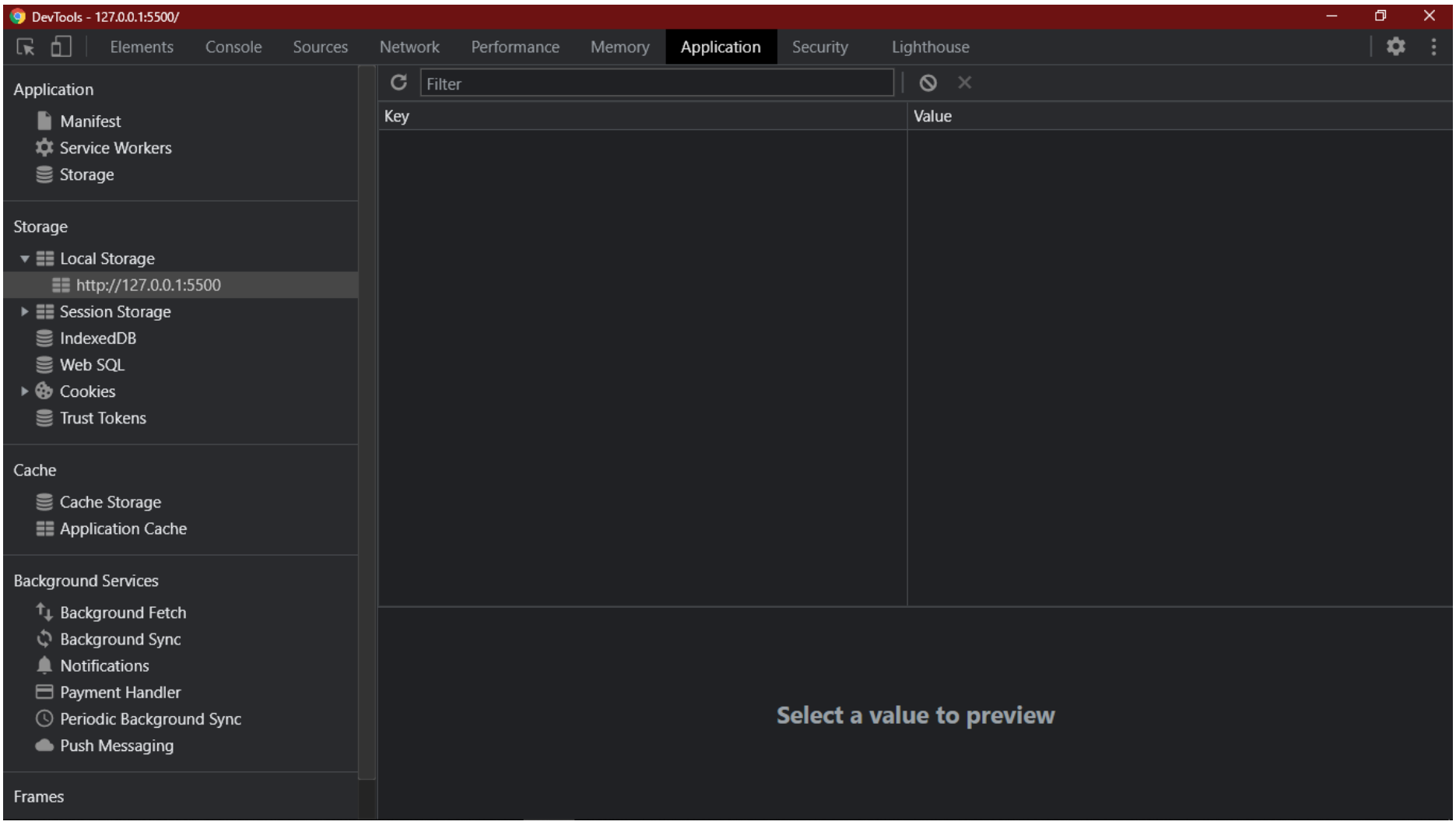
clear() →

É responsável por limpar todas as chaves armazenadas no localStorage.

Suponhando que você realizou as etapas do método acima (removeItem()), podemos limpar as chaves restantes, 'pessoa1' e 'pessoa3':

```
function limparItens() {
  localStorage.clear();
}

limparItens();
```

Após isso, o localStorage estará totalmente vazio:



key() →

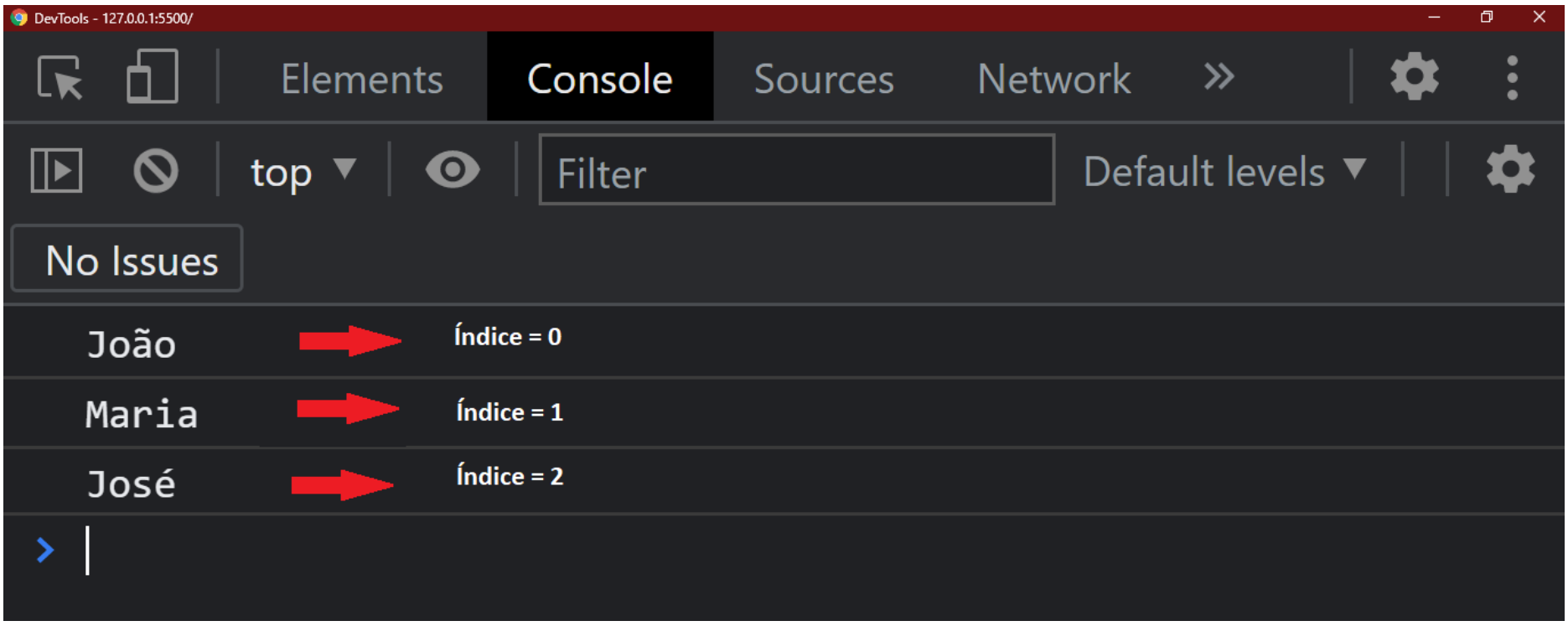
Caso possua um número inteiro especificado entre parênteses, retorna a chave daquela posição. Assim como um Array, sua primeira posição tem índice igual a 0. Sua sintaxe é:

```
localStorage.getItem(localStorage.key(indiceDesejado));
```

Podemos fazer uma combinação de estruturas e retornar todas as chaves através de um `for`. Por exemplo:

```
function adicionarItem() {
  localStorage.setItem('pessoa1', 'João');
  localStorage.setItem('pessoa2', 'Maria');
  localStorage.setItem('pessoa3', 'José');
}

adicionarItem();

function mostrarChaves() {
  for(var i =0; i < localStorage.length; i++){
    console.log(localStorage.getItem(localStorage.key(i)));
  }
}

mostrarChaves();
```

O resultado da execução de ambas funções será:



Para exemplificarmos mais ainda, faremos uma lógica a seguir. Sugerimos que você execute também no seu computador para fixar o conhecimento adquirido, vamos lá:

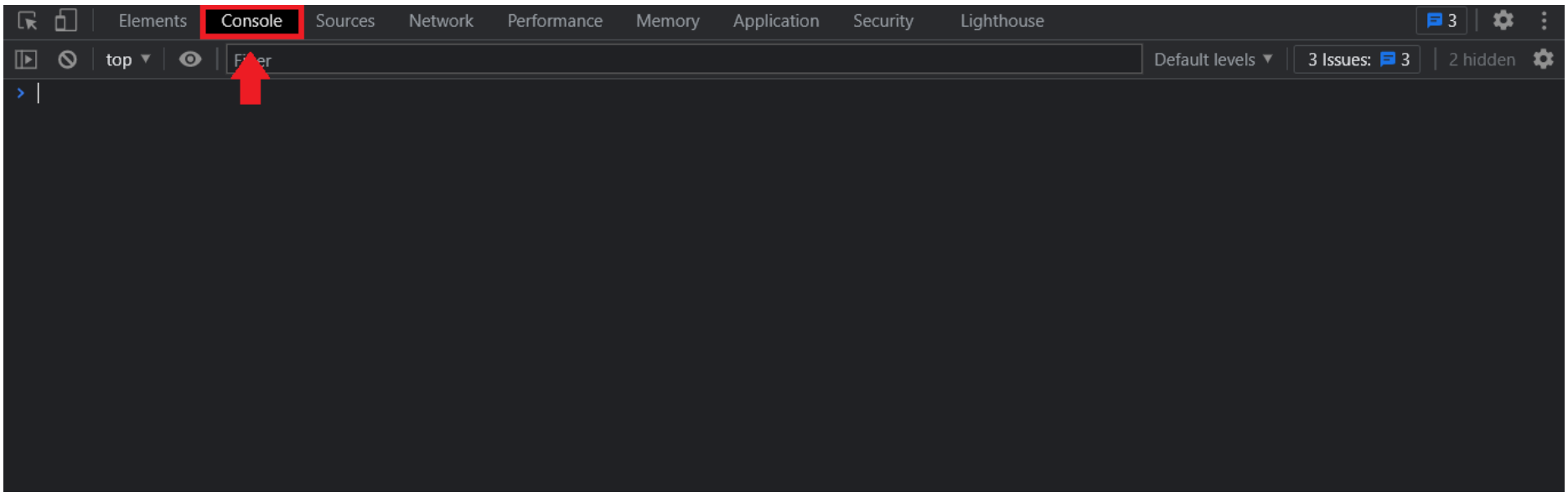
Requisitos:

- navegador, sugerimos que seja o Google Chrome, esse é um navegador muito bom e utilizamos ele aqui na Faculdade iv2.

Passo a passo:

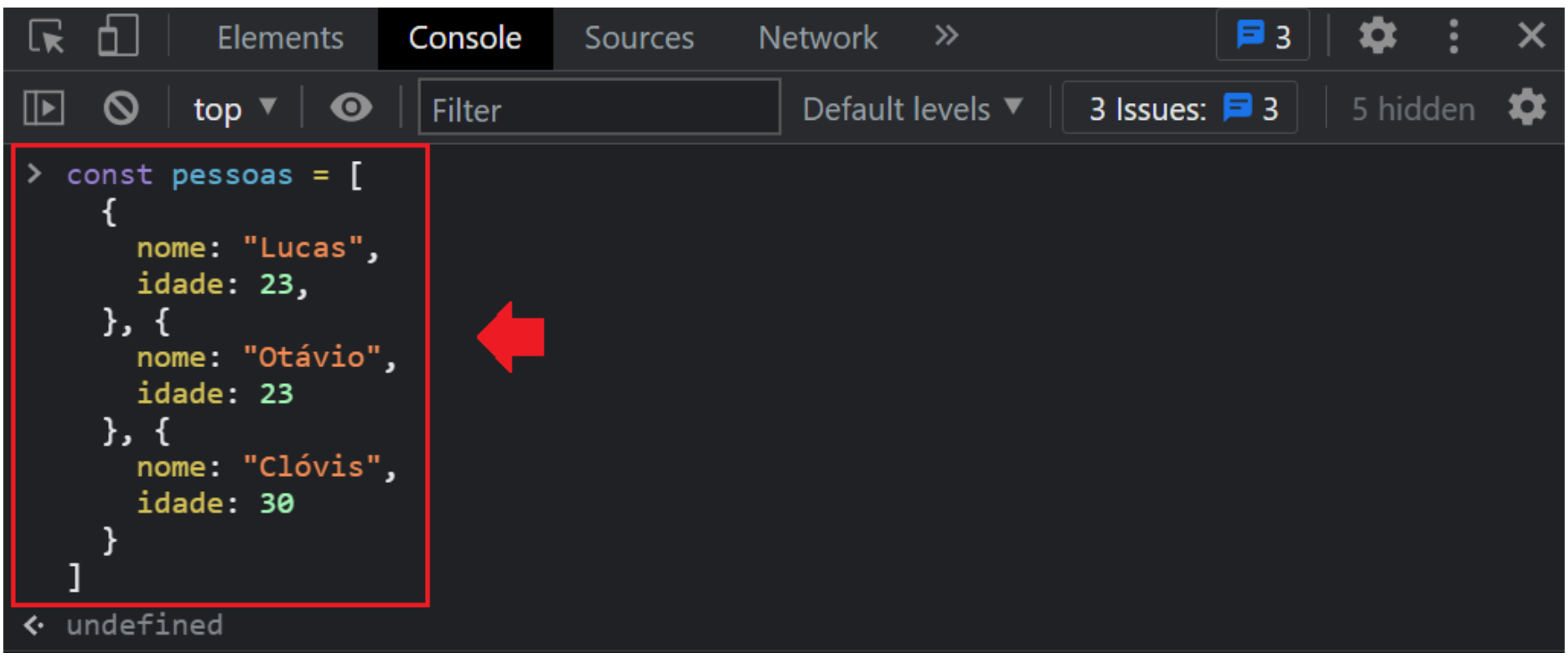
- Abra uma nova guia do navegador:

- Aperte `F12` ou clique com o botão direito do mouse sobre a tela e em seguida clique em `Inspecionar`, em seguida procure a barra de opções e clique na aba `Console`:



- Caso não tenha testado ainda, saiba que é nesse console que os resultados de `console.log()` aparecem.
  Nele conseguimos inserir comandos em geral, por isso recomendamos fortemente o uso do mesmo em testes de aplicações ou apenas para curiosidades.
  A seguir o usaremos para guardar determinado Array de Objetos no localStorage em formato JSON. Insira o seguinte trecho de código no console e aperte enter (pode utilizar o Ctrl+C / Ctrl+V):

```
const pessoas = [
  {
    nome: "Lucas",
    idade: 23,
  }, {
    nome: "Otávio",
    idade: 23
  }, {
    nome: "Clóvis",
    idade: 30
  }
]
```

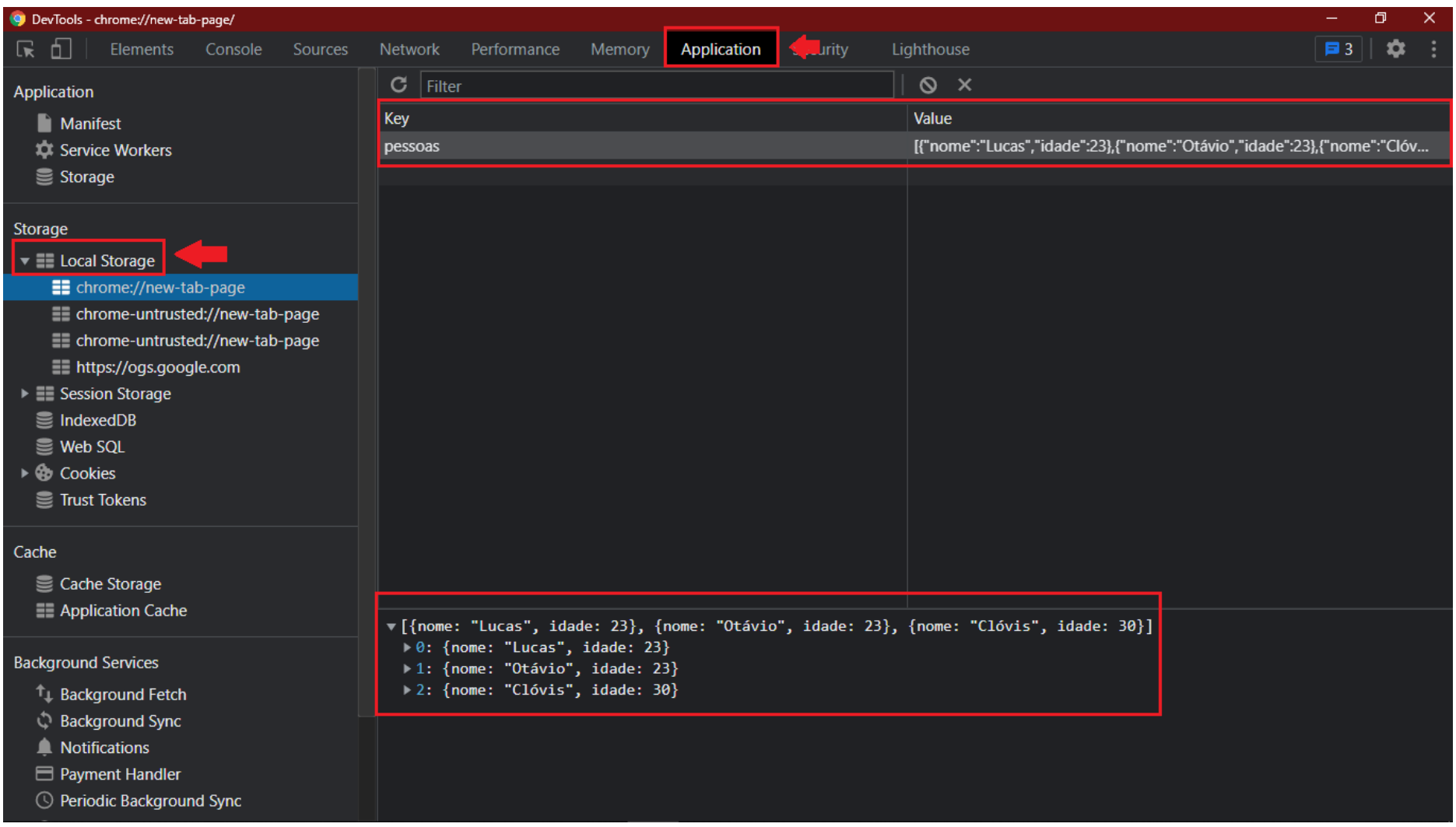


- Agora insira o seguinte código:

```
localStorage.setItem('pessoas', JSON.stringify(pessoas))
```

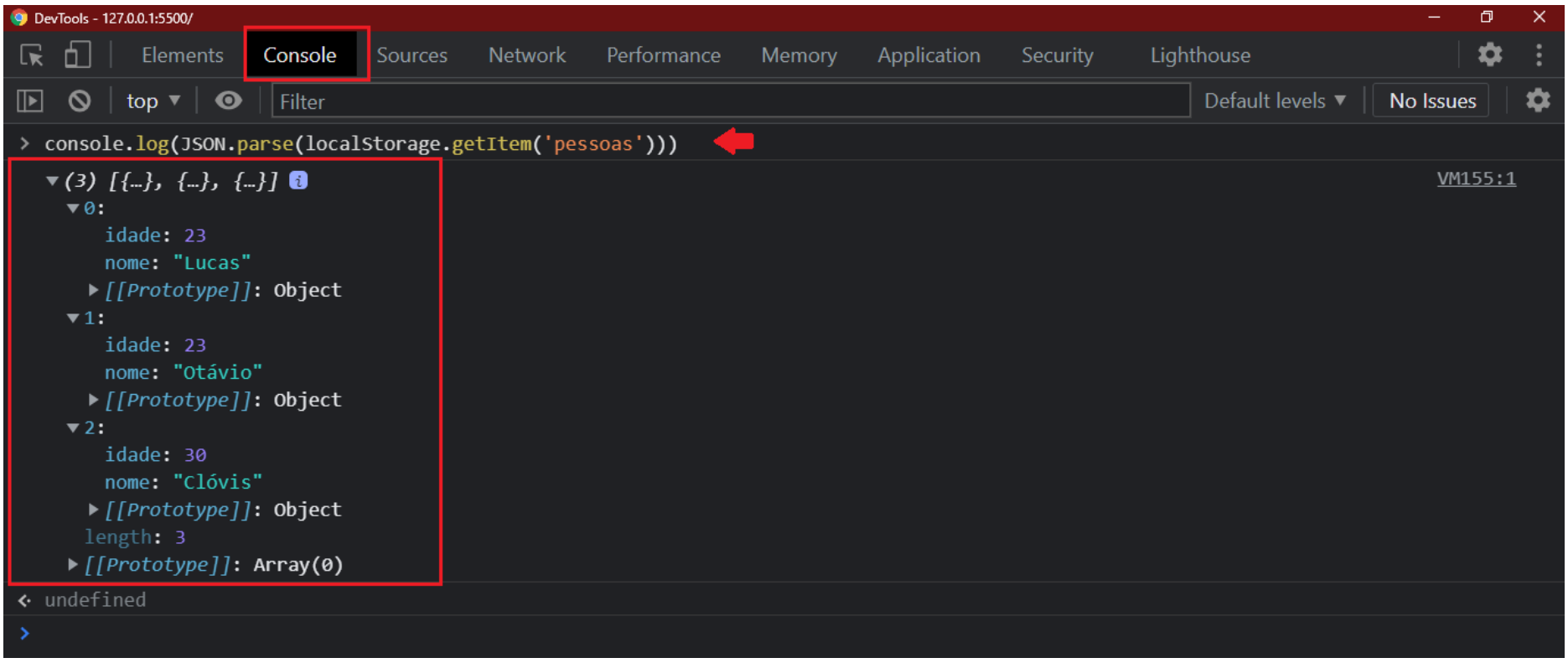
```
> const pessoas = [
    {
      nome: "Lucas",
      idade: 23,
    }, {
      nome: "Otávio",
      idade: 23
    }, {
      nome: "Clóvis",
      idade: 30
    }
  ]
<- undefined
> localStorage.setItem('pessoas', JSON.stringify(pessoas))
<- undefined
>
```

- Agora que inserimos nossas informações no localStorage, conseguimos vê-las na aba `Application`:



- Agora vamos visualizar esses dados de uma forma bem fácil, execute o seguinte:

```
console.log(JSON.parse(localStorage.getItem('pessoas')))
```



Explicando o resultado:

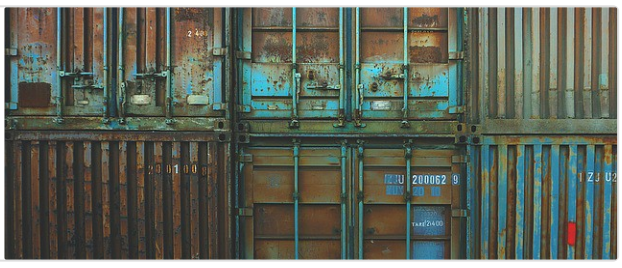
Primeiramente inserimos no Array `pessoas` três informações, a primeira era um Objeto com a propriedade nome="Lucas" e idade="23", na segunda informação temos outro Objeto, porém, desta vez com a propriedade nome="Otávio" e idade="23" e, por fim, outro Objeto com a propriedade nome="Clóvis" e idade="30". Dessa maneira recebemos como resposta de console.log(JSON.parse(localStorage.getItem('pessoas'))); uma informação que continha um Array de três Objetos, sendo a posição 0 contendo os dados de Lucas, na posição 1 contendo os dados de Otávio e na posição 2 os dados de Clóvis.

Além de todas nossas explicações, recomendamos fortemente a leitura do seguinte artigo:

Funcionalidades do Local Storage

Quando visitamos uma página web e temos que recorrer aos mesmos dados como, por exemplo: filtros de preferências, formulários , gerenciamento de sessão ou quaisquer outras informações que foram previamente salvas, recaímos em algumas escolhas: Fazer um login e rever as informações; Refazer o preenchimento dos

https://medium.com/jaguaribetech/dlskaddaldkslkdlskdlk-333dae8ef9b8



Bibliografias:

- https://www.json.org/json-en.html;
- https://www.w3schools.com/js/js_json_intro.asp;
- https://developer.mozilla.org/pt-BR/docs/Web/API/Window/localStorage;
- https://www.w3schools.com/jsref/prop_win_localstorage.asp;https://developer.mozilla.org/pt-BR/docs/Web/API/Storage/removeItem;