Continuando nossa API a partir de onde paramos, a estratégia do ‘show’, já tínhamos visto. Ele deve retornar o conteúdo do JSON.

```
res.end(JSON.stringify(data))
```

No delete, a ideia é atualizar data.urls com o método filter. O método filter percorre os objetos JSON e, obedecendo a um filtro, devolve ou não o valor percorrido.

```
data.urls = data.urls.filter(item => item.url != url)
```

Ou seja, toda vez que encontrar alguém diferente de quem está passado na URL, retorna verdadeiro (devolve para data.urls). Quando encontrar quem foi passado na URL, retorna falso e não coloca o valor na nova versão de data.urls. Importante destacar que a alteração de data.urls não altera o documento JSON.

Depois que atualizarmos o data.urls, podemos fazer uma função para adicionar a alteração ao documento JSON.

O primeiro passo é importar os módulos fs e path.

```
const fs = require('fs')
const path = require('path')
```

O segundo passo é utilizar o método writeFile(). Para utilizá-lo, precisamos do caminho do arquivo, quais dados vão ser escritos nele e uma callback.

```
fs.writeFile(
    path.join(__dirname, 'urls.json'),
    JSON.stringify(data, null, 2),
    err => {
        if (err) throw err
        res.end('Operação realizada com sucesso!')
    }
)
```

No stringify, usamos 3 argumentos para configurar a quantidade de espaços da indentação no documento, a partir do valor do terceiro argumento. O callback que está sendo passado faz o tratamento caso aconteça algum erro ou, caso contrário, responde à requisição com o texto “Operação realizada com sucesso!”.

Ao testar, note que o que é resolvido na página é método end que gera o “create” como conteúdo exibido. Por que?

Como temos uma callback para a mensagem de que foi deletado com sucesso, ela entra para a fila de execução e não bloqueia a aplicação. Como a aplicação segue, o que é resolvido é o return que gera o texto “create” na tela. Como trata-se de um comando de retorno, a aplicação encerra o Event Loop.

Basta colocarmos a chamada do método writeFile como retorno:

```
return fs.writeFile( ...
```

Agora, index.js está com o código um pouco “sujo” dentro da verificação de se deve ser deletado um conteúdo. Vamos colocar esse trecho de código como uma função e fazer a chamada dele dentro do if que verifica o del. Ou seja, copiamos o conteúdo que estava sendo retornado (que era a função writeFile) para uma função antes das verificações.

```
function writeFile(cb) {
    fs.writeFile(
        path.join(__dirname, 'urls.json'),
        JSON.stringify(data, null, 2),
        err => {
            if (err) throw err
            cb('Operação realizada com sucesso!')
        }
    )
}
```

Notem que trabalhamos uma callback como parâmetro de writeFile, sendo ela quem resolve o conteúdo para a confirmação da deleção. Logo, na chamada do método writeFile dentro do if que trata a deleção vamos passar uma função como argumento:

```
return writeFile(message => res.end(message))
```

Pronto! A funcionalidade deletar está concluída.

Na parte de criar um novo link favorito via API, vamos alterar data.urls utilizando o método push (insere conteúdo em uma estrutura, no caso, um objeto JSON):

```
data.urls.push(name, url)
```

Em seguida, vamos fazer o retorno da função que reescreve o documento JSON com data, ao invés de retornar res.end(‘create’):

```
return writeFile(message => res.end(message))
```

Testar para conferir que todas as operações estão funcionando. O que falta, para a próxima aula, é fazer a ideia de consumir a API pelo frontend funcionar.