Vamos trabalhar na criação de uma API.

Tudo está feito no frontend. Sempre que iniciamos a aplicação, perdemos tudo o que está na página. O próximo passo é preparar o backend para servir o frontend através de uma API (ou seja, o frontend vai, a partir de agora, receber os conteúdos do backend).

Criem uma nova pasta **api** dentro do projeto (project-favorites). Criem um documento **index.js** na pasta api. Vamos usar a mesma estratégia de criação do servidor feita anteriormente..

```
const http = require('http')

http.createServer((req, res) => {

}).listen(3000, () => console.log('API is running.'))
```

Nossa API deve fazer com que toda vez que entrar no / (ou seja, na index), seja enviado um JSON como resposta para a requisição, contendo os dados que vão servir à página.

Então, vamos criar o documento JSON nomeado de **urls.json** na pasta api.

```
{
    "urls": [
        {
            "name": "Web Academy",
            "url": "http://ufac.br/webacademy"
        }
    ]
}
```

Partimos, agora, para fazer a aplicação guardar e apagar no JSON. Para testar servir a API no frontend, vamos usar res.end passando o JSON. Antes, precisamos fazer a importação do documento JSON para index.js.

```
const data = require('./urls.json')
```

Em seguida, usem res.end dentro do createServer, da seguinte forma:

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

Quando entrarem no endereço do servidor local na porta 3000, vai ser enviado como resposta à requisição HTTP o conteúdo de urls.json.

Façam outro atalho no package.json para rodar com o nodemon o index.js da pasta api:

```
"api": "nodemon api/index.js"
```

Testem com **npm run api**.

Testem no navegador.

Agora, vamos fazer tratamento via endereço no navegador (URL), criando rotas na API. Quando passar name & url, adicionar. Quando passar name, url & del=1, deletar. Quando não passar nada, exibir o que estiver no JSON.

Ou seja, no padrão, vai servir o conteúdo na página. Com query strings, vai adicionar no JSON ou deletar de lá, dependendo da configuração. Query string é um modelo clássico de manutenção do estado da página. Elas são nada mais do que conjuntos de pares/valores anexados à URL.

Vamos criar as estratégias.

A primeira coisa é pegar as querys da URL.

Verifiquem o módulo URL na documentação do Node.js. O método que vamos usar é o parse. Ele tem três parâmetros, mas vamos usar somente dois: a URL e um valor verdadeiro. Passem a chamada dentro de um console.log para verificar o que retorna.

Antes disso, é necessário importar o módulo url:

```
const URL = require('url')
```

Dentro do create server, façam somente o console.log a seguir:

```
console.log(URL.parse(req.url, true))
```

Parte desse resultado é justamente o que queremos (o objeto JSON). Agora, adaptem o final para:

```
console.log(URL.parse(req.url, true).query)
```

Restou justamente o que queremos. Vamos extrair os dados de dentro do objeto com desestruturação.

```
const { name, url, del } = URL.parse(req.url, true).query
```

O dado del irá aparecer apenas na URL quando a intenção for de deletar. É uma informação que não será carregada para o JSON.

Finalmente, basta fazer as verificações necessárias com estruturas de seleção:

- Se não tiver name ou url, o conteúdo deve ser exibido na (servido à) página;
- Se tivermos name e url, queremos inserir no JSON;
- Se tivermos name, url e del, queremos apagar do JSON.

Apagar todos os testes no console e o res.end que estava fora dos ifs e fazer:

```
if(!name || !url)
    return res.end('show')

if(del)
    return res.end('delete')

return res.end('create')
```

Encerramos esse roteiro por aqui. Em seguida, continuaremos a estratégia para servir à página através da API.