API – Consumindo

Detalha-se o código de consumo da API do viacep, com o fito de ampliar o entendimento acerca do tema:

```
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4    <meta charset="UTF-8">
5    <script type="module">
6        const pesquisarCep = async() => {
7            const cep = document.getElementById('cep').value;
8            const url = `https://viacep.com.br/ws/${cep}/json/`;
9                const dados = await fetch(url);
10                const endereco = await dados.json();
11               if(endereco.logradouro!=undefined){
12                    document.getElementById('endereco').value =
endereco.logradouro;
13                    document.getElementById('bairro').value =
endereco.bairro;
14                    document.getElementById('cidade').value =
endereco.localidade;
15                    document.getElementById('estado').value = endereco.uf;
16                }else{
17                     document.getElementById('endereco').value = "";
18                    document.getElementById('bairro').value = "";
19                    document.getElementById('cidade').value = "";
20                    document.getElementById('estado').value = "";
21                }
22       }
23 document.getElementById('cep').addEventListener('focusout',pesquisarCep);
24 </script>
```

Na linha 6 a função async() é utilizada pois a regra na linguagem JavaScript é a execução síncrona das linhas de código, isto é, execução na sequência de cima para baixo e da esquerda para direita. Como uma estrutura externa será consumida e não há como determinar o tempo ou o custo do processo, utiliza-se o recurso assíncrono para “impedir” o fluxo do código.

Na linha 7 é criada a referência/variável cep para armazenar o CEP digitado na tag <input> do documento HTML.

Na linha 8 é criada a referência/variável url para receber o endereço web da API viacep com o parâmetro de pesquisa (conteúdo da referência/variável cep).

Na linha 9 a referência/variável dados aguarda o fetch(busca) dos dados no endereço web contido na referência/variável url, de forma assíncrona, ou seja, não impedindo

a execução dos demais comandos.

Na linha 10 a referência/variável endereco recebe o json obtido ou quando obtido pela referência/variável dados.

A estrutura condicional da linha 11 verifica se a referência/variável endereco recebeu dados, para nas linhas 12, 13, 14 e 15 atribuir os dados definidos na referência/variável endereco as tags <input> com os ids mencionados no getElementById, se não houver definição de dados na referência/variável endereco, especificamente logradouro, as tags <input> com os ids mencionados nas linhas 17,18,19,20 terão os dados substituídos por "".

Na linha 23 o método/função addEventListener da tag <input id='cep'> é utilizado para "escutar/detectar" quando a tag <input id='cep'> - perde o foco ou cursor deixa de estar sobre ela - e neste momento a função pesquisarCep é "chamada/acionada".

```
25    <style>
26       label{
27         width:65px;
28         display:block;
29         float:left;
30         text-align:right;
31         padding-right:3px;
32       }
33    </style>
```

Nas linhas acima os rótulos são alinhados.

```
34    <title>Aula</title>
35 </head>
36 <body>
37        <h1>Aula</h1>
38        <div >
39            <div>
40                <label for="cep">CEP: </label>
41                <input type="text" id="cep"  maxlength="8" size="8" >
42            </div>
43            <div >
44                 <label for="endereco">Endereço: </label>
45                <input type="text" id="endereco">
46            </div>
47            <div >
48                <label for="numero">Número:</label>
49                <input type="text" id="numero" size="6">
50            </div>
51            <div >
52                 <label for="bairro">Bairro:</label>
53                <input type="text" id="bairro">
```

```
            </div>
            <div >
                <label for="cidade">Cidade:</label>
                <input type="text" id="cidade">
            </div>
            <div >
                <label for="estado">Estado:</label>
                <input type="text" id="estado" size="2">
            </div>
        </div>
</body>
</html>
```

Nas linhas acima as marcações HTML são realizadas.

No exemplo abordado os recursos async/await foram utilizados, mas vale lembrar que eles são resultado de outros mecanismos ou recursos da linguagem JavaScript, como a função callback e a função promise.

Uma função callback é uma função passada a outra função como argumento, que é então invocado dentro da função externa para completar algum tipo de rotina ou ação[1].

```
<!DOCTYPE html>
<html>
<body>

<p id="ver"></p>

<script>
function exibir(x) {
  document.getElementById("ver").innerHTML = x;
}

function calcular(v1, v2) {
  let soma = v1 + v2;
  return soma;
}

let resultado = calcular(450, 502);
exibir(resultado);
</script>

</body>
</html>
```

952

No exemplo acima a função exibir precisa esperar a variável resultado receber o retorno da função calcular para apresentar os dados no documento HTML.

[1] Mozilla Foundation.. © 2022. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Client-side_web_APIs/Introduction . Acesso em 20 de jul. de 2022.

```
<!DOCTYPE html>
<html>
<body>

<p id="ver"></p>

<script>
    function exibir(x) {
      document.getElementById("ver").innerHTML = x;
    }

    function calcular(v1, v2, testeCallback) {
      let soma = v1 + v2;
      testeCallback(soma);
    }

    calcular(450, 502, exibir);
</script>

</body>
</html>
```

952

No código acima a função calcular computa as ações ligando o resultado a função exibir, de sorte que os dados possam aparecer no documento HTML.

As promises no JavaScript são estruturas que aguardam um retorno, não interrompendo o fluxo natural do processamento do código.

Para utilizar o recurso a classe Promise é acionada, recebendo os argumentos resolve(solução) e reject(erro), permitindo o encadeamento de processos através do método then.

Para ampliar a compreensão acerca do tema, considera-se o seguinte cenário hipotético: Um código será recebido e comparado com um valor pré-definido, retornando uma mensagem de sucesso para a comparação verdadeira. Vale salientar que no exemplo abaixo receberemos o código de uma tag <input>, mas o código poderia ser oriundo de um processo automatizado.

```
<!DOCTYPE html>
<html>
<body>

<p id="ver"></p>
<!--tag para exibição do conteúdo que será rerefenciada pelo id-->

<script>
    function exibir(argumento) {
     /* função que aguarda o resultado da função promise, especificamente
      o método .then para exibir o resultado na tag de id ="ver" */
      document.getElementById("ver").innerHTML = argumento;
```

```
    }

    let minhaPromessa = new Promise(function(solucao, erro) {
    //início da instância Promise
      // os parâmetros resolve[solucao] e reject[erro] são definidos para
retornar a promessa/resultado
      var codigo = "M3UC0D1G0";
      //variável codigo é criada para simular a inserção de dado

      if (codigo === "M3UC0D1G0") {
      //veriável codigo é testada
        solucao("CÓDIGO CORRETO.");
       // será retornada a mensagem "CÓDIGO CORRETO",caso o resultado do teste
do if seja true
      } else {
        erro("Error");
      // será retornada a mensagem "Error",caso o resultado do teste do if
seja false
      }
    });
//fim da instância Promise
    /* as instruções da minhaPromessa só serão executadas quando o método then
for acionado.
    vale ressaltar que o método then recebe as funções com os dois argumentos
oriundos da minhaPromessa, escoando-os para a função exibir   */
    minhaPromessa.then(function(var_ok) {exibir(var_ok);}, function(var_error)
{exibir(var_error);});
</script>

</body>
</html>
```