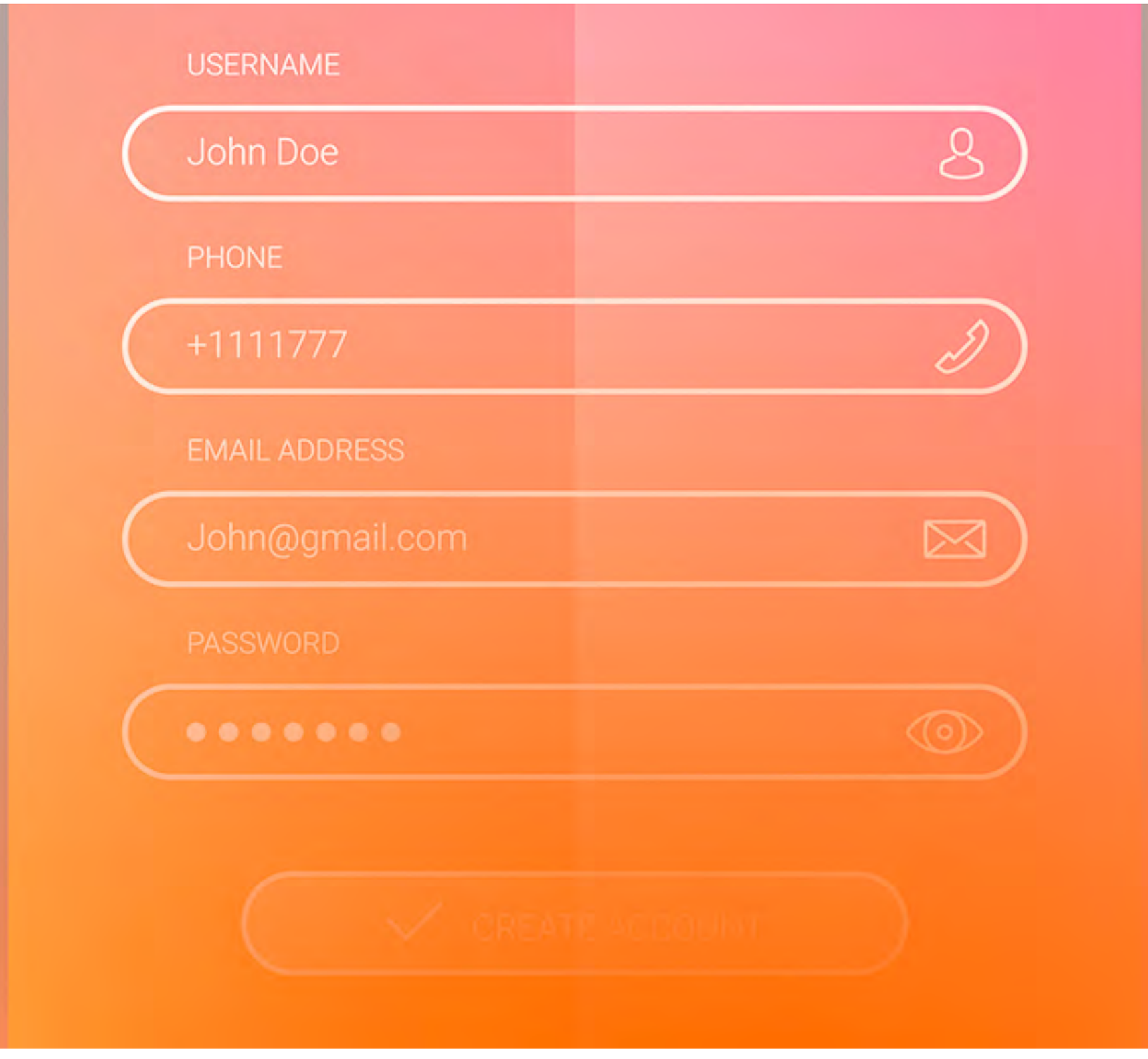
DOM E FORMULÁRIOS

Unidade 03



Sumário

Olá, **Fullturist,**

Seja bem-vindo(a)!

Olá, Fullturist! Bem-vindo(a) à trilha de Javascript!

Ao navegar pela internet, seja pelo seu computador ou utilizando seu smartphone, você já deve ter acessado sites como Facebook e Instagram e ficado admirado(a) com algumas funcionalidades fantásticas. Além disso, provavelmente, deparou-se com jogos divertidos e interativos que prendeu a sua atenção por um bom tempo. Nesse sentido, perguntou-se: como tudo isso é feito?

Para desenvolver algo parecido, é preciso dominar linguagens que vão além do HTML e CSS. Assim, nesta Trilha, vamos apresentar a linguagem de programação Javascript, que possibilitará o desenvolvimento de sites e aplicativos semelhantes ao Facebook, Instagram e Tik Tok.

A partir de agora, será possível entender e desenvolver funcionalidades mais complexas. Quando uma página web, como de redes sociais, faz mais do que apenas mostrar informações estáticas, ao invés disso, mostram em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, entre outros, tenha certeza de que o Javascript está por trás de tudo. Ficou empolgado(a)? Então, vamos lá!

Unidade **03**

DOM e Formulários

Olá, Fullturist!

Vamos recapitular o que você aprendeu até agora? Na Unidade 1, vimos os conceitos básicos da linguagem Javascript e testamos vários exemplos no VSCode e console. Já na Unidade 2, conhecemos os principais e mais utilizados eventos, os quais possibilitam que se desenvolva a interatividade nas páginas HTML.

Aposto que já começou a ficar mais empolgado(a). Agora, prepare-se, pois você irá aprender sobre o DOM (Document Object Model), uma ferramenta que permitirá o desenvolvimento de muitas coisas interessantes com Javascript, além de desenvolver formulários profissionais e validação de informações com Javascript. Ficou animado(a)? Então, vamos decolar!

Objetivos de aprendizagem

Ao final do estudo desta unidade, você será capaz de:

- Entender como funciona o DOM, compreendendo que uma página HTML é um conjunto de nós que pode ser consultado e manipulado quando estiver sendo visualizado pelo usuário.
- Criar formulários e adicionar validações utilizando JavaScript puro e manipulação do DOM.
- Utilizar JavaScript para adicionar interação a elementos no lado do navegador, sem precisar consultar serviços de Backend.
- Manipular elementos com o JavaScript, explorando os formulários como elemento principal.
- Manipular os eventos através de um Event Listener, compreendendo de que se trata de uma forma profissional de programar eventos com Javascript.

3.1 DOM

Quando você está desenvolvendo um documento HTML, acaba utilizando várias tags para marcar os conteúdos que deverão ser apresentados no browser. No entanto, como o browser interpreta o documento HTML e exibe os conteúdos de forma correta? Você se lembra?

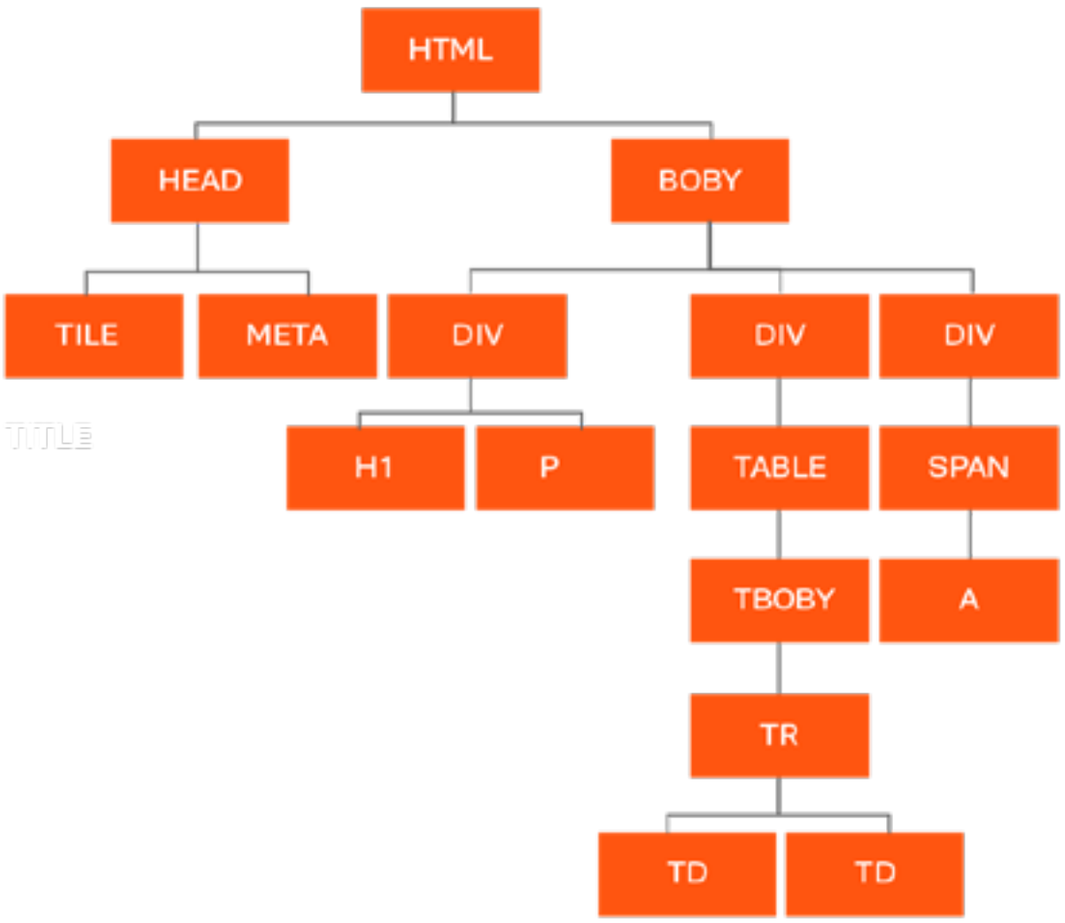
Vamos recordar: quando o browser abre um documento HTML, ele inicia a interpretação das tags de cima para baixo e da esquerda para a direita. Costumamos dizer que ele lê todas as tags, que informam para o browser qual o conteúdo ela representa e como deve apresentar esse conteúdo para o usuário. Além da exibição do conteúdo, o browser organiza todo o documento HTML na memória em uma estrutura tipo árvore. Essa estrutura é chamada de Document Object Model, mais conhecida pela sua sigla DOM.

Para saber mais sobre o DOM e conhecer todos os métodos disponíveis por esse objeto, clique aqui e visite o endereço na documentação MDN Web Docs.

No DOM, cada elemento HTML é um nó da estrutura árvore. Entretanto, o mais interessante vem agora: ele pode ser acessado através de um objeto, o document, que possui métodos (funções) que nos permitem acessar e alterar todos os elementos HTML utilizando Javascript. Desse modo, possibilita-nos desenvolver coisas fantásticas.

A imagem a seguir mostra como um documento HTML é estruturado no formato de árvore na memória, formando o DOM, que nos ajudará a entender ainda mais sobre esse poderoso objeto.

```html
<!DOCTYPE HTML>
<html lang="pt-br">
    <head>
        <meta charset="UTF-8">

        <title>HTML DOM</title>
    </head>

    <body>

        <div>
            <h1>Exemplo de árvore DOM</h1>
            <p>Esta página é um modelo para o estudo do DOM</p>
        </div>

        <div>
            <table>
                <tbody>
                    <tr>
                        <td>Primeira coluna da linha</td>
                        <td>Segunda coluna da linha</td>
                    </tr>
                </tbody>
            </table>
        </div>

        <div>
            <span>
                <a href="#">Link para outra página</a>
            </span>
        </div>

    </body>
</html>
```



Fonte: Elaborada pelo autor (2021).

3.2 Selecionando Elementos no DOM

O objeto DOM possui vários métodos e, aqui, iremos aprender os principais e aqueles que são mais utilizados pelos desenvolvedores profissionais para realizar o acesso e a manipulação dos elementos HTML. Vamos conhecer?

3.2.1 Método getElementById(<id do elemento>)

Esse método permite acessar e manipular o elemento através do identificador programado na propriedade id das tags HTML. Ele localiza e retorna o elemento que deve ser salvo em uma variável para que possa ser manipulado. Veja um exemplo simples a seguir.

```
let elemento = document.getElementsById('paragrafo1');
```

Fonte: Elaborada pelo autor (2021).

No exemplo, o acesso ao DOM está sendo feito através do objeto document. O método getElementById retorna e armazena um parágrafo identificado com o id ‘paragrafo1’. O identificador passado como parâmetro no método é o mesmo configurado na propriedade id das tag.

3.2.2 Método getElementsByClassName (<nome da classe>)

O método getElementsByClassName(<nome da classe>) retorna um uma lista de elementos HTML (HTMLCollection) com todos os elementos que fazem uso do nome da classe na propriedade class. O nome da classe é passado como argumento para o método.

Diante disso, volta para um objeto chamado de NodeList, que é um array com os elementos, e cada elemento pode ser acessado pelo seu índice. Se não existir nenhum elemento utilizando a classe passada como argumento, o método retornará null. Veja um exemplo simples:

```
let myContainer = document.getElementsByClassName(‘container’);
```

Fonte: Elaborada pelo autor (2021).

3.2.3 Método getElementsByTagName (<nome da tag>)

Da mesma forma do método anterior, o método getElementsByTagName(<nome da tag>) também retorna uma HTMLCollection, uma NodeList com todos os elementos com o nome da tag que foi passado como argumento. Caso não exista nenhum elemento com o nome da tag, o método retornará null. Segue um exemplo de utilização.

```
let buttons = document.getElementsByTagName('button');
```

Fonte: Elaborada pelo autor (2021).

3.2.4 Método querySelector(<seletor>)

O método querySelector(<seletor>) seleciona o elemento HTML, utilizando os seletores CSS, que irá retornar o primeiro elemento que se equipara ao seletor CSS, passado como argumento. Dessa forma, vale ressaltar que o seletor deve seguir a sintaxe CSS, ou seja, #id, .classe e o nome da tag. Caso não tenha nenhum seletor, o método retornará null. Veja o exemplo a seguir.

```
let resetButton = document.querySelector('#reset');
let paragrafo = document.querySelector('p');
let imagem = document.querySelector('.image');
```

Fonte: Elaborada pelo autor (2021).

No exemplo anterior, na linha 1 está sendo selecionado um botão cujo id é reset; já, na linha 2, um elemento pelo nome da tag, nesse caso, o parágrafo (p); e, na linha 3, o elemento está sendo escolhido pela classe image. Evidencia-se que o método irá retornar o primeiro elemento que corresponda ao seletor passado como argumento, ou seja, se existir mais de um parágrafo no documento HTML, o código da linha 2 retorna apenas o primeiro. Se for necessário selecionar mais elementos, é possível utilizar o método getSelectorAll().

3.2.5 Método querySelectorAll(<seletor>)

A forma de utilizar esse método é muito semelhante ao método querySelector(<seletor>), mas a grande diferença é que o método querySelectorAll retorna todos elementos de acordo com o seletor passado como argumento, que é um objeto NodeList, muito semelhante a um array. Se nenhum elemento for encontrado, ao invés de null, o método querySelectorAll retorna um NodeList vazio. Veja um exemplo simples, onde volta um NodeList com todos os elementos configurados com a classe ".menu-item".

```
let lista_de_elementos = document.querySelector('.menu-item');
```

Fonte: Elaborada pelo autor (2021).

Na Unidade 2, você já teve a oportunidade de acessar o DOM para selecionar elementos utilizando o método querySelector(<seletor>), com o objetivo de manipular os eventos. Além disso, é possível acessar todas as propriedades que esse elemento possui, por exemplo: alterar a imagem modificando o atributo src de um elemento img, mudar o estilo CSS alterando o atributo style, trocar o texto de um botão submit definindo o atributo value, e por aí vai. Você pode acessar tudo o que o elemento possui! O que achou?

Aposto que está ansioso(a) para ver um exemplo prático e colocar a mão na massa, certo? Então, vamos lá!

Para conhecer mais sobre os métodos abordados até aqui e outros métodos, clique aqui e poderá conhecer outros exemplos interessantes, vale a pena dar uma olhadinha!

No exemplo a seguir, criaremos uma página onde serão exibidos alguns cursos oferecidos pela Fullture. Um ponto importante é que nos modelos o foco é o JavaScript e não o design e estilo da página, portanto, não se incomode com a aparência e estética, ok?

No VSCode, comece criando o seguinte arquivo index.html.

```
<!DOCTYPE html>
<html lang=”pt-br”>
<head>
  <title>Testando os métodos de acesso ao DOM</title>
  <meta charset=”utf-8” />
</head>
<body>
  <header>
    <div id=”logo”>
      <img src=”https://www.fullture.com/wp-content/uploads/2020/07/
Simbolo-Square-1.svg” />
    </div>
    <nav class=”primary-nav”>
      <ul>
        <li class=”menu-item”><a href=”#”>Home</a></li>
        <li class=”menu-item”><a href=”#”>FullturePro</a></li>
        <li class=”menu-item”><a href=”#”>Fullture Falst</a></li>
        <li class=”menu-item”><a href=”#”>Sobre nós</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <h1>Bem vindo à Fullture! Torne sua Carreira à Prova de Futuro!</h1>
    <div class=”container”>
      <section class=”section-a”>
```

```
            <h2>DEV FULL STACK</h2>
            <p>Já pensou que em menos de um ano você estará pronto para
construir um aplicativo do zero? Com nosso curso, você terá uma visão ampla
de todo projeto para criação de uma aplicação e poderá definir o seu caminho
profissional, apto para atuar no Front-end, Back-end e Mobile em uma das
maiores empresas do Brasil e do mundo.</p>
            <button id=”button1”>Leia Mais</button>
          </section>
          <section class=”section-b”>
            <h2>DATA SCIENCE</h2>
            <p>O FullturePro Data Science é um curso completo e de alto nível
para você construir uma carreira à prova de futuro. Abrangendo todos os tópicos
relevantes da área - desde a bases matemáticas e estatísticas ao aprendizado
da linguagem Python</p>
            <button id=”button2”>Leia Mais</button>
          </section>
          <section class=”section-c”>
            <h2>Product Manager</h2>
            <p>O Curso de Product Management da Fullture é um curso completo
de alto nível para aprender como atuar na carreira de Product
              Management em empresas modernas digitais e de tecnologia.</p>
            <button id=”button3”>Leia Mais</button>
          </section>
        </div>
    </main>
    <script src=”script.js”></script>
</body>
</html>
```

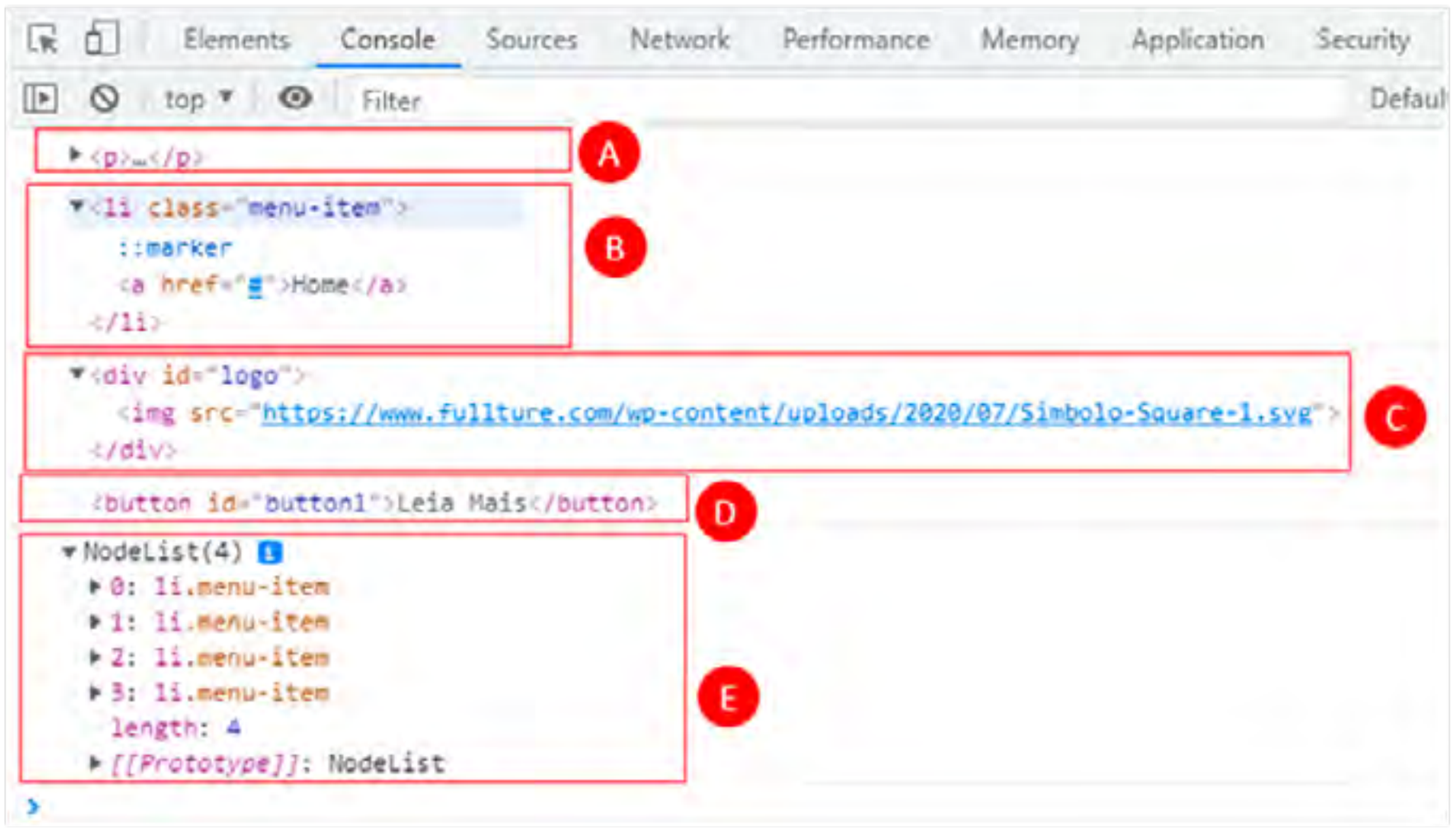
Fonte: Elaborada pelo autor (2021).

O código anterior é um documento html simples, sem formatação de css, que utilizaremos com o intuito de acessar seus elementos através do DOM e seus métodos. Alguns elementos como a DIV do logo e os bottons estão com o atributo id configurado e outros com classes iguais, como os itens do menu.
No código Javascript a seguir, vamos selecionar esses elementos de diferentes formas utilizando os métodos getElementById(id), querySelector(<seletor>) e querySelectorAll(<seletor>). No VSCode, crie um novo arquivo com o nome script.js e salve-o junto ao arquivo html.

```
// selecionando os elementos HTML acessando o DOM
const paragrafo = document.querySelector(“p”);
console.log(paragrafo);
const menu_item = document.querySelector(“.menu-item”);
console.log(menu_item);
const div_logo = document.querySelector(“#logo”);
console.log(div_logo);
const btn1  = document.getElementById(“button1”);
console.log(btn1);
const menu_itens = document.querySelectorAll(“.menu-item”);
console.log(menu_itens);
```

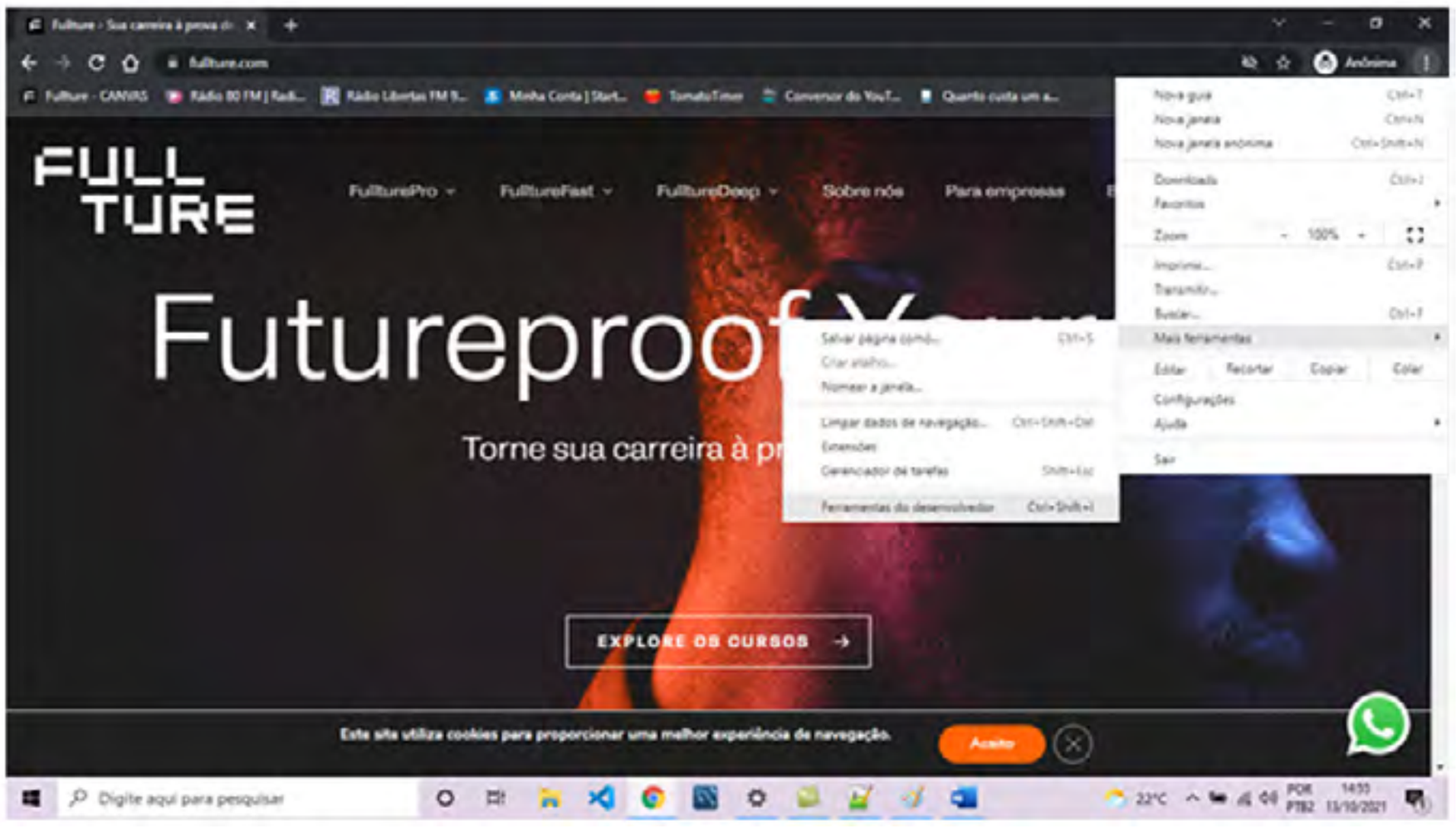
Fonte: Elaborada pelo autor (2021).

Para que consiga entender com mais facilidade o código anterior, veja na imagem que segue o resultado produzido depois de implementar e testar o JavaScript.



Fonte: Elaborada pelo autor (2021).

A partir disso, vamos entender o código JavaScript com base na análise da imagem com o resultado apresentado no navegador. Mas antes, se você está com dúvida em como abrir o console no navegador, siga os seguintes passos: se estiver utilizando o Google Chrome, acesse o menu -> mais ferramentas -> ferramentas do desenvolvedor, conforme a imagem a seguir.



Fonte: Elaborada pelo autor (2021).

Feitas essas colocações, vamos para a análise! Na figura 2, linha 3, do código JavaScript, estamos acessando, através do DOM, um elemento parágrafo, utilizando o método querySelector('p'). Depois de acessado, esse elemento parágrafo está sendo armazenado na constante paragrafo. Para selecionar esse parágrafo, foi passado como argumento o nome da própria tag 'p'. Observe que no HTML existem vários parágrafos e o método querySelector('p') retorna ao primeiro parágrafo que ele encontra no DOM.

Na linha 4, do código Javascript, estamos exibindo o elemento parágrafo, que é acessado pela constante paragrafo. Veja na Figura 2 (a) que a tag (elemento) parágrafo está sendo exibida. Note que é exatamente a tag parágrafo implementada no código HTML e isso permite que podemos manipular esse elemento dinamicamente pelo JavaScript.

Continuando nossa análise, na linha 6, estamos acessando um elemento HTML através do método querySelector('.menu-item'), porém, nesse momento utilizamos o seletor de classe. Perceba na Figura 2 (b) que o primeiro elemento que está utilizando com a classe .menu-item é acessado, nesse caso, o primeiro elemento <li>. Se precisarmos acessar todos os elementos que usam a classe .menu-item, precisaríamos utilizar o método querySelectorAll('.menu-item'), como está sendo feito na linha 15 do código JavaScript. Esse método seleciona todos os elementos <li> através de um objeto NodeList, o qual pode ser visto na Figura 2 (e).

Para finalizar a análise desse exemplo, nas linhas 9 e 12, estamos acessando elementos HTML através dos seus ids. A diferença é que na linha 9 o elemento DIV com o id logo está sendo acessado através do método querySelector("#logo"), enquanto que, na linha 12, o elemento button é acessado pelo método getElementById("button1").

Diante disso, fique atento(a) e continue estudando o conteúdo, pois mais a frente você encontrará um comparativo entre os dois métodos.

Agora que você já implementou e testou o código JavaScript acessando os elementos através do DOM (com o objeto document), que tal alterar, dinamicamente, algumas propriedades desses elementos? Para isso, adicione os códigos abaixo no arquivo script.js a partir da linha 18. Salve o arquivo e teste novamente sua página no navegador.

```
paragrafo.style.color = ‘red’;
const titulos = document.querySelectorAll(“h2”);
titulos.forEach( function(element){
   element.style.color = ‘blue’;
});
const links = document.querySelectorAll(“.primary-nav .menu-item a”);
links.forEach( function(element){
    element.href=”https://www.fullture.com/”;
});
```

Fonte: Elaborada pelo autor (2021).

No novo trecho do código, começamos a alterar o elemento parágrafo na linha 18. Uma vez que se tem acesso a esse elemento, podemos alterar suas propriedades, inclusive, o texto, como veremos mais adiante. Nesse caso, apenas estamos mudando a cor da fonte, a partir da propriedade style e, em seguida, a propriedade css color.

Na linha 20, selecionamos todos os elementos H2, utilizando o método querySelectorAll(‘h2’). O resultado é um objeto NodeList com todos os elementos H2. Estando de posse de todos esses elementos armazenados na constante títulos, na linha 22 até a linha 24, estamos mudando a cor do texto desses títulos para azul. Para percorrer um objeto NodeList, usamos o método forEach(), que é do NodeList, o qual que executa algo semelhante à estrutura de repetição for e ao método map() de um array. A cada iteração do método forEach(), um elemento é acessado e passado como argumento para a função interna no atributo element.

O objeto NodeList retornado por alguns métodos do DOM, dentre eles, o querySelectorAll(<seletor>), possui outros métodos tão interessantes quanto o método forEach(). Para saber mais, clique aqui.

Entre as linhas 26 e 29, escolhemos os elementos de link ‘a’. Fique atento(a) para a forma em que esses elementos estão sendo selecionados pelo método querySelector(“.primary-nav .menu-item a”) na linha 26. Ali, está sendo utilizada uma regra de seletores CSS. Se fossemos traduzir o resultado poderia ser algo assim: selecione todos os elementos ‘a’ que estão contidos dentro do elemento que utiliza a classe “.menu-item” e no elemento que utiliza a classe “.primary-nav”. Na linha 27, o NodeList percorre através do método forEach() e, na linha 28, estamos mudando o endereço dos links através da propriedade href, um por um.

Os códigos que foram implementados até esse momento permitiram demonstrar que podemos alterar muita coisa em uma página HTML acessando os elementos na estrutura DOM. Agora, imagine se você precisasse adicionar um novo menu no HTML? Ou, ainda, se necessitasse adicionar um novo curso? Utilizando o Javascript podemos acessar a estrutura DOM e adicionar, de uma forma dinâmica, novos elementos e novos conteúdos. Vamos conferir como isso pode ser feito?

3.3 Manipulando e adicionando elementos no DOM

O DOM é uma estrutura de dados em memória que contém todos os elementos contidos dentro de um arquivo HTML. Os navegadores quando fazem a leitura de um arquivo HTML montam essa estrutura na memória para depois apresentar o conteúdo. Cada alteração realizada nele é percebida pelos navegadores que atualizam o conteúdo apresentado. Com o JavaScript podemos acessar a estrutura DOM, além de adicionar, alterar ou remover elementos. Vamos conhecer alguns desses métodos que possibilitarão com que você faça essas alterações no DOM.

createElement()	Cria um novo elemento na estrutura DOM.
appendChild()	Adiciona um nó (elemento) a uma lista de nós filhos de um nó (elemento) pai especificado. Por exemplo, em uma lista, o elemento é um nó pai, então, você pode adicionar um elemento como nó filho que usa esse método.

innerHTML	Adiciona um novo conteúdo a um elemento. Exemplo, se você quer alterar ou inserir um novo texto a um parágrafo, irá utilizar esse método.
append()	Adiciona um nó (elemento) após o último nó filho de um nó (elemento) pai. Exemplo, nó código HTML que estamos estudando nesta Unidade: se você precisasse inserir um novo link no menu, este método iria adicionar esse item no final do menu.
removeChild()	Remove elementos filhos de um nó.

Fonte: Adaptado de JavaScript TUTORIAL.

Feitas as devidas apresentações, vamos alterar o código Javascript utilizando esses métodos, para que você entenda como eles funcionam. No entanto, antes, altere o arquivo HTML adicionando um novo botão que irá disparar um evento de click com o código que fará a alteração do DOM. No arquivo index.html, adicione o código do botão na linha 23, logo abaixo do código: <h1 >Bem vindo à Fullture! Torne sua Carreira à Prova de Futuro!</h1 >.

```
<button id=”add-item-menu”>Adiciona novo Item no menu</button>
```

Fonte: Elaborada pelo autor (2021).

Feito isso, vamos ao JavaScript! No trecho de código a seguir, adicionaremos no menu da nossa página um novo item “contato” e remover o curso Product Manager. Nesse sentido, manipularemos o evento de click do novo botão adicionado anteriormente no HTML. Acrescente o trecho de código a partir da linha 31 do arquivo script.js.

```
const btnAdd = document.getElementById('add-item-menu');
btnAdd.addEventListener('click', function(){
   // criando um novo item no menu
   const li = document.createElement('li');
   li.className = 'menu-item';

   const a = document.createElement('a');
   a.href = 'https://www.fullture.com/';
   a.innerHTML = 'Contato';
   li.appendChild(a);
   const ul = document.querySelector('ul');
   ul.appendChild(li);
});
```

Fonte: Elaborada pelo autor (2021).

Para entender 100% da lógica do código JavaScript acima, com a ajuda da figura a seguir, vamos analisar a estrutura do código HTML referente ao menu.

```
<nav class="primary-nav">
    <ul>
        <li class="menu-item"><a href="#">Home</a></li>
        <li class="menu-item"><a href="#">FullturePro</a></li>
        <li class="menu-item"><a href="#">FulltureFalst</a></li>
        <li class="menu-item"><a href="#">Sobre nós</a></li>
    </ul>
</nav>
```

Fonte: Elaborada pelo autor (2021).

A estrutura de elementos HTML do menu está organizada da seguinte forma: o menu é uma lista não ordenada <ul> e cada item dessa lista é um elemento <li>. Os links de cada opção do menu, elemento <a>, estão dentro de cada item <li>, respectivamente. Então, os elementos <a> são nós filhos dos elementos <li>, e os elementos <li> são nós filhos do elemento <ul>. Então, para inserir um novo item “Contato” nesse menu:

- primeiro, precisaremos criar um elemento <li>;
- segundo, criar e configurar um elemento de link <a>;
- terceiro, adicionar o elemento <a> como filho do elemento <li> criado;
- quarto, selecionar o elemento <ul>;
- quinto, para finalizar, adicionar o novo elemento <li> como nó filho do elemento <ul>.

Agora que você já entendeu a organização da estrutura HTML do menu, vamos ver como funciona o código JavaScript.

O código foi adicionado no evento de click do botão ‘add-item-menu’. Então, quando clicar nesse botão, o código desse evento será executado. Na linha 35, está sendo criado um novo elemento <li> através do método document.createElement(‘li’). Quando um elemento é criado você tem acesso a todos os atributos que ele possui, dessa forma, veja que a classe ‘menu-item’ configura o elemento li criado.

Para configurar dinamicamente a classe CSS para o novo elemento, utiliza-se li.className. Não é possível empregar li.class, porque a palavra class é uma palavra reservada de JavaScript.

Na linha 38, criamos o elemento de link <a>. Feito isso, na linha 39, o endereço para onde o link aponta é configurado através da propriedade href e, na linha 40, o texto do link é configurado utilizando o atributo innerHTML do DOM (acessado através do objeto document), que irá inserir o texto “Contato” entre a abertura e fechamento da tag (nesse caso, entre <a> e). Nesse momento, estamos com o elemento link <a> criado e o elemento <li> também criado, então, na linha 42, adicionamos o elemento a como elemento filho do elemento li, utilizando o método appendChild().

Já temos o item de menu completamente criado, nesse instante, basta recuperar a lista, o elemento <ul>, como está sendo feito na linha 44 e, depois, adicionar o novo item criado dinamicamente. Teste o código e observe que, ao clicar no botão “Adiciona novo Item no menu”, o novo item de menu surgirá como mágica! Esse é o poder de manipular o DOM. Já imaginou?! Pense nas possibilidades que a partir de agora você será capaz de desenvolver!

Existem mais métodos interessantes e que um dia será muito útil para você. Então, não deixe de conhecê-los! Você pode saber mais sobre esses os métodos, acessando os sites JavaScript Tutorial e MDN Web Docs.

O DOM sempre será utilizado quando precisar atualizar uma página ou construir uma interface, uma página com interatividade avançada. Do mesmo modo, será aplicado quando estiver desenvolvendo com Ajax, mas esse é um assunto para outra discussão.

Através do objeto document, você manipulará o DOM e poderá mover itens dentro de uma página, adicionar novos itens e criar efeitos CSS dinamicamente sem precisar recarregar a página, manipular formulários e muitas outras coisas interessantes. Falando em formulários, eles são elementos especiais e podem ser trabalhados e manipulados de uma forma diferente do que vimos até agora, além de possuírem eventos bem específicos. Vamos dar uma olhada e aprender muita coisa interessante sobre esse assunto? Então, vamos em frente!

3.4 Formulários e o Objeto FORM

Nesse momento, o foco é para que você aprenda sobre o objeto Form contido na estrutura DOM e como manipular esses formulários de forma dinâmica.

Para relembrar, os formulários são uns dos principais recursos que utilizamos para prover a interatividade com os usuários dos nossos sites e aplicativos web. Todo sistema web possui uma série de formulários que permitem que os usuários informem dados, que, geralmente, são enviados a um servidor web para processamento e armazenamento ou são utilizados para atualizar imediatamente, de forma dinâmica, a página, como, por exemplo, adicionar outro item em uma lista.

Vamos começar a estudar o objeto form da melhor maneira, direto com um exemplo prático e colocando a mão na massa.

EXEMPLO

Como exemplo, será criado um formulário para coletar informações sobre o peso e a altura dos usuários. Além disso, vamos pedir para que o usuário também informe o seu nome e sobrenome. A ideia é que, quando o usuário informar essas informações no formulário, ao invés de enviar a um servidor, esses dados serão armazenados em um objeto literal Javascript (Opa! Olha coisa nova surgindo :D) e esse objeto armazenado em um array. Depois do objeto criado e adicionado ao array, vamos exibir este objeto em uma DIV resultado. Para isso, vamos começar a implementar o código HTML. Abra o VSCode e crie um novo arquivo chamado formulario.html e coloque o código abaixo. Lembre-se que aqui não estamos nos importando com a aparência do HTML.

```
<!DOCTYPE html>
<html lang=”pt-br”>
  <head>
    <meta charset=”UTF-8”>
    <title>Exemplo de Utilização do Objeto Form</title>
  </head>
  <body>
    <form method=”get” action=”#” id=”form”>
      <p>
  Nome: <input type=”text” id=”nome” />
</p>
<p>
  Sobrenome: <input type=”text” id=”sobrenome”/>
</p>
<p>
  Peso: <input type=”text” id=”peso”/>
</p>
<p>
  Altura: <input type=”text” id=”altura”/>
</p>
<p>
    <button>Cadastrar</button>
  </p>
</form>
<div class=’resultado’></div>
<script src=”formulario.js”></script>
  </body>
</html>
```

Fonte: Elaborada pelo autor (2021).

Esse é um bom exercício para você implementar um CSS e tornar a página do HTML anterior mais bonita e interessante. Feito isso, vamos ao código JavaScript, que vai manipular o objeto form e executar a criação de objetos com os dados e inserir esses objetos em um array. Assim, crie no VSCode um novo arquivo chamado formulario.js e implemente o código abaixo.

```
const pessoas = [];
const form = document.getElementById('form');
const resultado = document.getElementById('resultado');
function recebeEventoForm(evento){
    evento.preventDefault();
    const nome = form.getElementById('nome');
    const sobrenome = form.getElementById('sobrenome');
    const peso = form.getElementById('peso');
    const altura = form.getElementById('altura');
    pessoas.push( {
        nome: nome.value,
        sobrenome: sobrenome.value,
        peso: peso.value,
        altura: altura.value
    } );
    resultado.innerHTML = `<p>Nome: ${nome.value}</p>`+
    `<p>Sobrenome: ${sobrenome.value}</p>`+
    `<p>altura: ${altura.value}</p>`+
    `<p>Peso: ${peso.value}</p>`;
}
form.addEventListener('submit', recebeEventoForm);
```

Fonte: Elaborada pelo autor (2021).

Para começar, o nosso código está criando um array, que armazena na constante pessoas, como pode ser visto na linha 1. Será nesse array que iremos armazenar todos os objetos criados com os dados das pessoas coletados no formulário.

Seguindo o código, na linha 3, estamos selecionando o formulário no DOM através do método getElementById() e armazenando na constante form. Até esse ponto, nada de novidade. Porém, diferente dos outros elementos, o elemento form é um objeto especial e que possui todos os métodos para selecionar elementos, assim como o DOM, getElementById(), querySelector(), querySelectorAll() e outros. Esses métodos só podem ser utilizados para selecionar os elementos que compõem esse formulário, ou seja, os seus nós filhos.

Dessa forma, entre as linhas 11 e 14, os elementos inputs estão utilizando o objeto form e não o objeto object do DOM. Nesse ínterim, você deve estar se perguntando, qual a vantagem disso? Essa dúvida é fácil de responder: a performance!

Os nossos exemplos são simples, com poucos elementos HTML, mas em um sistema profissional, um layout de uma tela de um sistema ou site web profissional, é grande o número de elementos HTML, com isso, a estrutura DOM fica com um volume de dados considerável. Então, quando selecionamos os elementos de um formulário acessando o DOM diretamente pelo objeto document, o tempo para que esses elementos sejam encontrados aumentam. Quando utilizamos o objeto form a busca e manipulação dos elementos desse formulário é acessado rapidamente economizando tempo de execução.

Os estudos até o momento possibilitaram aprender muito sobre como manipular a estrutura DOM, que é uma estrutura que representa os documentos HTML, possibilita que você acesse a página através do JavaScript e, ainda, permite alterar a estrutura do documento, o estilo e o conteúdo.

A partir de agora, você é capaz de criar scripts que podem fazer coisas fantásticas e de forma dinâmica. Na próxima Unidade, vamos evoluir no aprendizado sobre a utilização de APIs HTML5, que permitirá armazenar dados nos computadores dos usuários. Até lá!

Síntese

Nesta Unidade, você conheceu o DOM e o Objeto Form e aprendeu a manipulá-lo com os seus principais métodos. Vamos a um resumo rápido dos pontos mais importantes desse estudo:

- Aprendeu que uma página HTML é um conjunto de nós que podem ser consultados e manipulados através da estrutura DOM.

- Compreendeu que, com o DOM, pode criar aplicações que atualizam os dados e elementos da página de forma dinâmica.

- Observou que, por meio do DOM, é possível adicionar, alterar e remover elementos na página, o que possibilita desenvolver uma interatividade fantástica ao usuário.

- Identificou que o objeto form é semelhante ao objeto document, compreendendo que isso possibilita manipular os elementos de um formulário de forma mais rápida e reduzindo o tempo de processamento.

Fullture Insights

- MALDONADO, Leonardo. Entendendo o DOM (Document Object Model). Tableless, [S.l.], 8 fev. 2018. Disponível em: https://tableless.com.br/entendendo-o-dom-document-object-model/. Acesso em: 6 set. 2021.

- MDN Web Docs. Documentação sobre Javascript. [S.l]: MDN Web Docs, 2021. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Building_blocks/Events. Acesso em: 25 ago. 2021.

- W3SCHOOLS. Javascript Events. [S.l]: W3 Scholls, 2021. Disponível em: https://www.w3schools.com/js/js_events.asp. Acesso em: 18 ago. 2021.

FU
LL
TU
RE

www.fullture.com