Fundamentos do Bootcamp

Bootcamp: Programador de Software Iniciante

Guilherme Assis

2020

Fundamentos do Bootcamp

Bootcamp: Programador de Software Iniciante

Guilherme Assis

Sumário

Capítulo 1. Introdução

Introdução à programação

Um programa de computador é uma sequência de passos definidos por um programador para alcançar um objetivo. No nosso dia a dia, utilizamos diversos programas em nosso computador, como Word, Excel entre outros.

Esses programas também podem ser chamados de softwares. Software não é palpável, e sim instalável no computador, enquanto hardware é algo palpável, como por exemplo o notebook, mouse, teclado etc.

Os computadores só entendem bits (0 e 1), eles não entendem palavras formadas por letras, por exemplo. Para representar letras, números, símbolos etc., são utilizados um conjunto de bits, também chamado de byte. Por exemplo, para representar a letra “a”, o conjunto de bits seria 01100001. Seria inviável programar dessa forma, por isso são utilizadas as linguagens de programação.

Com as linguagens de programação, o programador pode então escrever os comandos em uma linguagem de alto nível, e depois uma outra ferramenta fica responsável por traduzir esse código para a linguagem que o computador entende. Existem várias linguagens de programação, como por exemplo JavaScript, Java e Python.

Um algoritmo é uma sequência de passos que devem ser executados para atingir determinado objetivo, como por exemplo uma receita de bolo. Um algoritmo não necessariamente é um programa de computador, ele pode ser executado por uma pessoa. Uma tarefa pode ser realizada por diferentes algoritmos, não existe somente uma forma de correta de realizar determinada ação.

Introdução ao JavaScript

O JavaScript é uma linguagem de programação criada em 1995, sendo hoje umas das linguagens mais populares. No próximo capítulo é mostrada uma pequena

história da sua evolução. Apesar de seu nome lembrar a linguagem de programação Java, uma não tem relação com a outra.

Inicialmente ela foi criada para ser utilizada dentro dos navegadores, porém, recentemente foram criadas formas de utilizá-la também fora do navegador, como com a utilização do Node.js, que interpreta o código escrito nessa linguagem fora do ambiente do navegador.

Uma página web é formada principalmente por três tecnologias: HTML, CSS e JavaScript. O HTML é o responsável por fornecer a estrutura da página, por exemplo colocando os títulos, imagens e botões na tela. Esses elementos inicialmente são colocados sem nenhuma estilização, somente com sua aparência nativa. Neste ponto entra o CSS, sendo o responsável por estilizar a página, customizando a sua aparência. Com ele, então, é possível definir questões relacionadas ao estilo, como cores, bordas, tamanho de fonte etc. Somente com o HTML e CSS uma página fica estática, sem a possibilidade de interagir com o usuário de acordo com suas ações. O JavaScript atua em uma página web justamente para isso, possibilitando que ela interaja com o usuário de acordo com suas ações.

Introdução ao NodeJS

O NodeJS foi criado por Ryan Dahl e mais 14 colaboradores em 2009, na tentativa de resolver o problema de arquiteturas bloqueantes. Plataformas como .NET, Java, PHP, Ruby ou Python, paralisam um processamento enquanto realizam um processo de I/O no servidor. Esta paralisação é o chamado modelo bloqueante (Blocking-Thread). Neste tipo de arquitetura, cada requisição que chega no servidor é enfileirada e processada uma a uma, não sendo possível múltiplos processamentos delas. Enquanto uma requisição é processada, as demais ficam ociosas em espera. Alguns exemplos de tarefas de entrada e saída de dados que bloqueiam as demais são enviar e-mail, fazer consulta no banco de dados e leitura em disco.

Para amenizar o problema gerado por essa espera, esses servidores criam várias threads para darem vazão à fila de espera. Quando aumenta-se muito o

número de acessos ao sistema, é preciso fazer upgrade nos hardwares, de forma a ser possível rodar mais threads, gerando um custo maior.

O NodeJS possui uma arquitetura não bloqueante (non-blocking thread) e apresenta uma boa performance em consumo de memória, utilizando ao máximo o poder de processamento dos servidores. Nele, as aplicações são single-thread, ou seja, cada aplicação possui um único processo. Assim, ele utiliza bastante a programação assíncrona, com o auxílio das funções de call-back do JavaScript. Quando uma requisição está fazendo uma ação de I/O, ele a deixa executando em background e vai processando outras requisições. Assim que o processamento do I/O termina, dispara-se um evento e o call-back correspondente à requisição, ou seja, a função que deve ser executada com resultado da resposta é colocada na fila para ser executada assim que possível. Em uma arquitetura bloqueante, o jeito de lidar com essa concorrência seria criar múltiplas threads para lidar com as diversas requisições. O problema é que, dentro de cada thread, o tempo gasto com espera de resposta não é aproveitado, deixando aquela parte da CPU ociosa.

O NodeJS foi criado utilizando o V8, que é um motor JavaScript de código aberto, criado pelo Google e utilizado no seu navegador, o Google Chrome. Ele compila e executa o código em JavaScript, além de manipular a alocação de memória, descartando o que não é mais preciso. Sendo assim, podemos dizer que o NodeJS é basicamente um interpretador de código JavaScript. A diferença do NodeJS para a codificação habitual do JavaScript é que o JavaScript sempre foi uma linguagem cliente-side, sendo executada somente no lado do usuário, em seu browser. Com o NodeJS, é possível executar o código JavaScript no servidor. Ele mantém um serviço rodando no servidor, que faz a interpretação e execução de códigos JavaScript. O V8 do Google é multiplataforma, sendo possível sua instalação em vários sistemas operacionais.

A criação do NodeJS está muito ligada à crescente utilização das SPAs (Single Page Applications), nas quais seu desenvolvimento é feito principalmente em JavaScript, pois a partir do momento em que é possível trabalhar com a mesma linguagem de programação tanto no back-end quanto no front-end, muitas empresas

se interessaram pela tecnologia, devido à possibilidade do reaproveitamento de conhecimento.

Com o NodeJS, também é possível criar aplicações desktop, com o auxílio de ferramentas como o Electron. Ele é basicamente o acoplamento do NodeJS com o Chromium, que é uma parte open-source do browser Google Chrome. Com ele, uma aplicação desenvolvida em HTML5 e JavaScript pode ser empacotada em um arquivo executável e distribuído. Esta tecnologia é a base do editor de texto Atom, e foi desenvolvida pelo GitHub. Uma aplicação Electron pode utilizar as APIs do HTML5 e as APIs do NodeJS para prover uma boa experiência ao usuário. Esta tecnologia é muito útil para aplicações que já tem uma versão web consolidada, que foi desenvolvida em HTML5 e JavaScript, e que querem criar uma versão desktop para suprir uma demanda de seus usuários, sem que para isso seja despendido muito esforço e investimento.

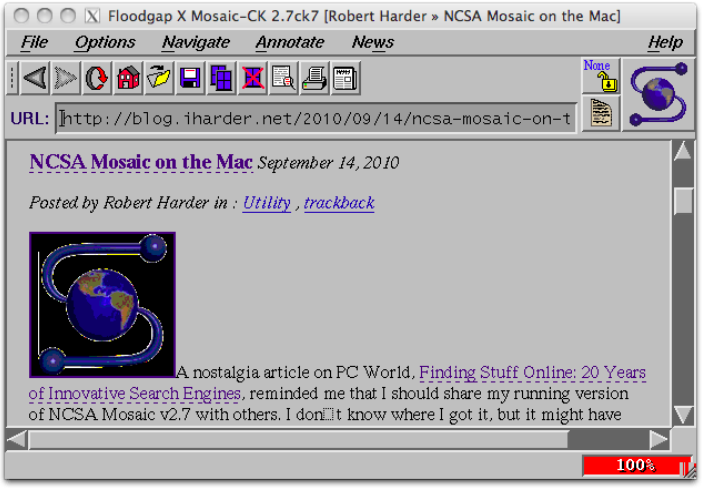
O NodeJS pode ser utilizado em aplicações Real-Time, como aplicações colaborativas como chats online, aplicativos de mensagens e jogos online. Ele é indicado para estes cenários por possuir pacotes compatíveis com o protocolo de WebSockets, que permitem trafegar dados por meio de uma conexão bidirecional, tratando as mensagens através de eventos no JavaScript.

Capítulo 2. JavaScript

Evolução do ECMAScript

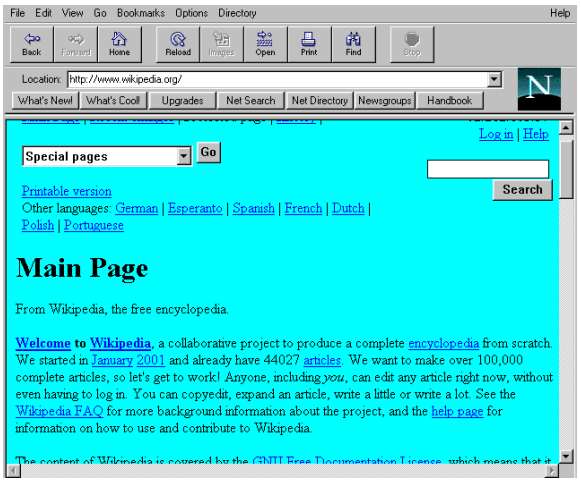
O JavaScript foi criado em 1995 por Brendan Eich, que era funcionário da Netscape. Nessa época os browsers ainda eram estáticos, e o mais famoso era o Mosaic, da NCSA. Em 1994, a Netscape foi fundada para explorar a web, criando-se então o Netscape Navigator, que em pouco tempo se tornaria o browser dominante da época. Como os browsers eram estáticos, a Netscape chegou à conclusão de que a web tinha que se tornar mais dinâmica, pois sempre precisava-se fazer uma requisição no servidor para obter uma resposta no navegador. Brendan Eich foi contratado para criar uma linguagem que resolvesse esse problema. As imagens abaixo ilustram o navegador Mosaic e Navigator da época.

Figura 1 – Navegador Mosaic.



Fonte: http://shipit.resultadosdigitais.com.br/blog/javascript-1-uma-breve-historia-da-linguagem/.

Figura 2 – Navegador Netscape Navigator.



Fonte: http://shipit.resultadosdigitais.com.br/blog/javascript-1-uma-breve-historia-da-linguagem/.

A ideia inicial era que fosse implementada a linguagem Scheme, uma linguagem funcional baseada no LISP no Navigator. Porém, a Netscape já tinha outros projetos em parceria com a Sun Microsystems, e elas desejavam colocar a promissora linguagem Java no Navigator. Houve uma discussão interna e optaram por ter uma linguagem ao invés das duas. A linguagem deveria parecer com o Java, pois como é uma linguagem funcional Scheme, tem um nível de complexidade maior, o que poderia fazer com que a linguagem não ficasse popular. Mesmo assim o JavaScript sofreu influência de linguagens funcionais, como Self, Perl e Python. Em maio de 1995, foi criado um protótipo chamado Mocha da linguagem em 10 dias. O nome, depois, foi alterado para LiveScript. Em novembro de 1995, saiu uma versão do Navigator com a versão da linguagem feita em 10 dias, sem muitas alterações. Em dezembro de 1995, devido ao Java estar em seu ápice, a linguagem foi renomeada para JavaScript.

Em agosto de 1996, a Microsoft criou uma linguagem idêntica ao JavaScript para ser utilizada no Internet Explorer 3. Devido a isso, a Netscape resolveu normatizar a linguagem através da organização ECMA International, que era uma companhia especializada em padrões e normativas. Em novembro de 1996, iniciou-se os trabalhos em cima da normativa ECMA-262. Como o nome JavaScript era patenteado pela Sun Microsystems (hoje Oracle), ele não poderia ser usado, e por

isso foi dado o nome ECMAScript. Porém, até hoje a linguagem é mais conhecida por JavaScript, e o nome ECMAScript é mais utilizado para se referir às versões da linguagem. O ECMA-262 é mantido pelo Comitê Técnico 39, também chamado de TC39, e é formado por especialistas de grandes empresas, como Microsoft, Mozilla e Google. Desde a adesão ao ECMA, o ECMAScript passou por várias versões. Segue abaixo uma breve explicação sobre elas:

- ECMAScript 1: lançada em 1997, representa a primeira versão da linguagem com padrões e normativas definidos após a adesão ao ECMA.
- ECMAScript 2: criada em agosto de 1997 para se adequar à ISSO/IEC 16262.
- ECMAScript 3: criada em dezembro de 1999, contém melhorias consideráveis, como laços de repetição do-while, tratamento de exceções, dentre outros recursos.
- ECMAScript 4: foi concluída em 2008, sendo baseada em ML, uma linguagem funcional utilizada em ambientes acadêmicos. Como a linguagem seria muito diferente do ECMAScript 3, optou-se por abandoná-la e dar continuidade à versão anterior.
- ECMAScript 5: foi lançada em 2012 com recursos importantes, como suporte nativo ao JSON, métodos avançados de manipulação de arrays, getters e setters, entre outros.
- ECMAScript 6: foi lançada em 2015 com recursos avançados, como reflection, collections, binary data, entre outros.
- ECMAScript 7: foi lançada em 2016, sendo também chamada de ECMAScript 2016. Foram adicionados mais recursos, como operadores exponenciais e outros.
- ECMAScript 8: foi lançada em junho de 2017, adicionando mais funcionalidades, como métodos para string padding e outros.

- ECMAScript 9: lançada em junho de 2018, adicionando mais recursos, como o método finally nas promises, a possibilidade de utilizar o iterator de forma assíncrona e os operadores rest e spread.

Podemos notar que, depois que o JavaScript passou a ser mais utilizado pela comunidade, principalmente após o lançamento do NodeJs, o pessoal está trabalhando ativamente em melhorias da linguagem, lançando todo ano uma nova versão com a adição de novas funcionalidades. Geralmente os navegadores modernos passam a dar suporte às novas versões rapidamente, porém alguns mais antigos podem não suportar recursos novos que foram adicionados. Nestes casos, existem alguns Polyfills que podem ser adicionados aos projetos para fazer com que a aplicação funcione em browsers antigos. O que um polyfill faz é simular o funcionamento daquele código de uma forma que o navegador antigo entenda, para que a aplicação possa funcionar sem que para isso se tenha que alterar seu código.

Uma ferramenta muito utilizada pelos desenvolvedores é o Babel. Ele basicamente é um compilador de JavaScript, que permite que os desenvolvedores escrevam código utilizando funcionalidades não suportadas pela versão mais recente do ECMAScript, porém que já estão previstas para as próximas versões. O Babel, então, pega esse código e o converte para uma forma que funcione de acordo com a especificação atual. Permite, assim, que os desenvolvedores trabalhem hoje com novidades que ainda estão por vir.

Variáveis

Uma variável é um local no qual pode ser armazenado um valor, como um número ou texto. Com as variáveis é possível realizar operações, por exemplo adição e subtração. Os valores das variáveis podem ser alterados de acordo com a lógica implementada.

Cada variável pode ter um tipo, que indica se ela tem um número ou texto, por exemplo. Segue abaixo alguns dos tipos de variáveis no JavaScript:

- **String**: representa um texto, e é representado entre aspas duplas ou aspas simples, exemplo: “texto de exemplo”.

- **Boolean**: uma variável do tipo boolean pode ter dois valores, verdadeiro ou falso, também chamados de true ou false.

- **Number**: uma variável numérica representa os números, podendo ser números inteiros ou decimais.

Uma variável é criada a partir do comando “var”, seguida de seu nome, exemplo: var a = 2. O operador “=” é utilizado para atribuir um valor a essa variável. Se ela for criada sem valor atribuído, seu valor é considerado como “undefined”, ou seja, não teve nada atribuído a ela ainda. É possível passar um valor nulo para a variável, a partir da palavra “null”. Esse caso é diferente do “undefined”, pois a variável tem um valor nulo, ao contrário do “undefined”, no qual ela não tem nenhum valor.

No JavaScript é possível criar uma variável numérica e depois atribuir um texto para ela, alterando assim o seu tipo. Não são todas as linguagens que permitem isso, algumas preferem restringir os valores que uma variável pode receber ao tipo que foi definido inicialmente. O tipo de uma variável no JavaScript pode ser consultado a partir da palavra-chave “typeof”, por exemplo: “typeof a” retornaria o tipo de variável “a”.

Operadores

No JavaScript, podemos utilizar operadores matemáticos para realizar operações, por exemplo soma e subtração. Segue abaixo os operadores básicos:

- **+**: utilizado para realizar adição, exemplo: 5 + 3, resultado 8.
- **-**: utilizado para realizar subtração, exemplo: 5 – 3, resultado 2.
- ***:** utilizado para realizar multiplicação, exemplo: 5 * 3, resultado 15.
- ***/***: utilizado para realizar a divisão, exemplo: 6 / 2, resultado 3.

- **%**: utilizado para obter o resto de uma divisão, exemplo: 7 % 3, resultado 1 (7 dividido por 3 é 2 e sobra 1).

No JavaScript, os operadores seguem a precedência das operações assim como na matemática, de forma que a divisão e multiplicação tem precedência maior que a adição e subtração. Dessa forma, caso queira que determinada operação seja executada primeiro sem olhar a precedência, é preciso colocá-la entre parênteses. Exemplo: (5 + 3) / 2. Dessa forma é realizada a soma e depois a divisão; se a operação tivesse sido colocada desta forma: 5 + 3 / 2, seria realizado a divisão e depois a soma.

No JavaScript também existe os operadores ++ e --, que realizam o incremento e decremento respectivamente de uma unidade. Exemplo: “a = a + 1” equivale a “a++”, assim como a = a – 1 equivale a--.

Também existem os operadores +=, -=, *= e /=, que é são forma de abreviar a expressão de a variável receber ela mesma mais alguma coisa. Exemplo: “a = a + 3” equivale a “a += 3”. Os demais operadores atuam de forma similar.

Arrays

No JavaScript, um array é utilizado para representar uma lista. Por exemplo: “var carros = [“Gol”, “Uno”, “Palio”];”. Os arrays são representados por colchetes e seus elementos são separados por vírgula. Um array pode ter elementos de qualquer tipo, como string e números.

Os elementos de um array podem ser acessados a partir de seu índice. O primeiro elemento possui índice igual a 0. No exemplo acima, o elemento de índice 0 seria o Gol, de índice 1 o Uno e de índice 2 o Palio. Para acessá-los basta colocar o índice dentro de colchetes, exemplo: “carros[0]”.

Existem vários métodos para manipulação de arrays, por exemplo pop, push, shift, unshift, delete, splice, concat e slice. Eles são demonstrados nas aulas gravadas.

Estruturas de decisão

Estruturas de decisão são utilizadas para que o código consiga tomar decisões de acordo com as condições fornecidas. Elas servem para desviar o fluxo de execução de acordo com os critérios definidos.

Uma das estruturas mais utilizadas é o if / else. Ele pode ser visto como “se determinada condição for verdadeira, faça isso, senão, faça aquilo”. Vários if e else podem ser encadeados um no outro.

Para as comparações, são utilizados os operadores === e !==. O primeiro irá comparar se, por exemplo, uma variável é igual a outra, enquanto a segunda irá verificar se é diferente. Exemplo: if (a === b). Nesse caso, se a variável **a** fosse igual a **b**, a condição seria verdadeira e o código dentro do if seria executado.

No JavaScript, “===“ é utilizado para comparar se uma expressão é igual a outra, incluindo seu tipo, enquanto o operador “==” tenta realizar a conversão da variável para fazer a comparação. Por exemplo, a expressão (“1” === 1) retorna false, porque um número está em texto e o outro em number. Caso fosse utilizado (“1” == 1), a condição retornaria true, pois o JavaScript realizaria a conversão antes de fazer a comparação. Os operadores “!=” e “!==” atuam de forma similar, porém, verificando se as expressões são diferentes uma da outra.

Para concatenar expressões, são utilizados os operadores lógicos && e ||. O && é o “AND”, ele olha se todas as expressões são verdadeiras, enquanto o || vai olhar se pelo uma das comparações são verdadeiras.

Outra estrutura de decisão muita utilizada é o switch, que basicamente é uma outra forma de representar um agrupamento de vários if / else. Nas aulas gravadas todas estas estruturas são demonstradas a partir de exemplos.

Estruturas de repetição

As estruturas de repetição são utilizadas para executar uma ação várias vezes. Cabe à estrutura de repetição definir quando aquela repetição deve encerrar a execução. Geralmente, essas estruturas são muito utilizadas em conjunto com os arrays, para que possa realizar uma ação em todos os elementos do array.

No JavaScript temos várias estruturas, como por exemplo “for” e “while”. Durante a execução, é possível executar um “break” para encerrar todo o fluxo de repetição, ou um “continue” para que encerre somente a iteração corrente e vá para o próximo elemento. Nas aulas gravadas essas estruturas são demonstradas com mais detalhes a partir de exemplos.

Capítulo 3. HTML e CSS

HTML

Uma página web é constituída por três componentes básicos: HTML, CSS e JavaScript. O HTML é a linguagem de marcação responsável por mostrar o conteúdo. O CSS é a parte responsável por estilizar a página, por exemplo mudando o tamanho e cor da fonte. O JavaScript é o responsável por criar comportamentos na tela, como fazer com que uma informação desapareça da tela de acordo com alguma condição. O HTML (Hyper Text Markup Languag), além de exibir a informação, também dá significado à página, pois é através dos elementos contidos neles que buscadores de conteúdo leem a página e entendem o que os elementos significam. Ele marca as informações com tags, cada um com um significado. Por exemplo, ao escrever <h1>Título</h1>, você está falando que este conteúdo dentro da tag h1 é um título.

Tags são o conjunto de caracteres que formam um elemento. Existem tags que precisam de fechamento e outras que não. Por exemplo, a tag “p” representa um parágrafo precisa de fechamento, enquanto a tag
 que representa uma quebra de linha não precisa ser fechada. Um elemento é formado a partir de tags, e dentro das tags está o conteúdo do elemento. As tags servem para marcar onde começam e onde terminam os elementos. Atributos são informações que são passadas na tag para que ela se comporte da forma esperada, e possuem um nome e um valor. Por exemplo, o atributo “id” é o identificador único do elemento. Um exemplo de utilização de um atributo seria: Ir para o site do IGTI. Essa tag é uma das principais do HTML, ela cria um link para a URL informada no atributo href.

Um documento HTML possui uma estrutura básica, sendo dividido em três blocos principais: o doctype, o head e o body. O doctype é este código: <!DOCTYPE html>. Ele não é uma tag HTML; é uma instrução que sempre deve estar presente nos documentos, de forma a informar ao navegador e a outros programas que queiram ler a página que o código contido ali é HTML. Ele sempre deve estar na primeira linha do documento.

A tag head contém informações que não são mostradas para quem estiver lendo o documento. Eles são dados de uso e controle do documento, por exemplo vinculação de outros arquivos e aplicação de lógicas de programação através de scripts Javascript. Dentro da tag head pode estar incluída a tag meta, que não produz nenhum efeito aparente na página, mas é responsável por definir algumas informações que poderão ser utilizadas por exemplo por buscadores para pegar algumas informações básicas da página, como o título e uma pequena descrição.

A tag body contém o documento em si. É nesta tag que ficam as informações visíveis aos usuários, todo o texto e informações de mídia que serão apresentadas, como imagens. Os campos de formulários nos quais os usuários irão entrar com dados também estão incluídos no body. A figura abaixo ilustra a estrutura básica de um documento HTML.

Figura 3 – Estrutura básica HTML.

```
1.  <!DOCTYPE html>
2.  <html lang="pt-br">
3.    <head>
4.      <title>Título da página</title>
5.      <meta charset="utf-8">
6.    </head>
7.    <body>
8.      Aqui vai o código HTML que fará seu site aparecer.
9.    </body>
10. </html>
```

Fonte: http://blog.caelum.com.br/rest-principios-e-boas-praticas/.

Os formulários HTML são um dos principais pontos de interação entre um usuário e uma página web, que permitem aos usuários enviarem dados para o site. Na maioria das vezes os dados são enviados para o servidor web, porém a página web também pode interceptá-los para utilizá-los por conta própria. Um formulário HTML é constituído por um ou mais elementos. Esses elementos podem ser um input text, select box, buttons, checkboxes ou radio buttons. Na maioria das vezes, esses componentes estão ao lado de um label, que indicam seu propósito através de seu

nome. É muito importante que os labels estejam com conteúdo sugestivo da utilidade do elemento que estão sinalizando, pois esta informação também é utilizada por cegos que escutam a leitura da página através de programas de leituras específicos para cegos. A principal diferença entre um formulário HTML e um documento HTML normal é que, na maioria das vezes, os dados coletados pelo formulário são enviados para um servidor web para que sejam processados.

Os dados do formulário são enviados para o servidor através de uma requisição HTTP. O formulário é definido no HTML pela tag form. Tudo que está dentro da tag de abertura e de fechamento do form faz parte do formulário. O form é quem define como os dados serão enviados. Os seus atributos servem para configurar a requisição que será enviada quando o usuário clicar no botão de submit. Os dois atributos mais importantes são o action e o method.

O atributo action define para onde o dado será enviado. Seu valor deve ser uma URL válida, e caso o atributo não seja informado, os dados serão enviados para a URL da página que contém o form. Antes do HTML5, este atributo era obrigatório, e quando desejava-se passar os dados para a URL da própria página, era preciso informar o valor “#” neste atributo. No HTML5, este atributo já é opcional.

O atributo method define como os dados serão enviados. A forma como os dados serão enviadas é definida através do método HTTP que for informado neste atributo. Os métodos mais comuns são o GET e o POST. O método GET geralmente é utilizado para buscar dados do servidor, e por isso o corpo da requisição é enviado vazio. Dessa forma, caso um formulário seja submetido com o GET, os dados serão passados na própria URL, pois o body da requisição vai vazio. A imagem abaixo mostra um formulário submetido com GET.

Figura 4 – Form Method GET.

```
<form action="http://foo.com" method="get">
  <div>
    <label for="say">What greeting do you want to say?</label>
    <input name="say" id="say" value="Hi">
  </div>
  <div>
    <label for="to">Who do you want to say it to?</label>
    <input name="to" value="Mom">
  </div>
  <div>
    <button>Send my greetings</button>
  </div>
</form>
```

Fonte: https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data.

Considerando o exemplo da imagem acima, a URL seria passada para o servidor da seguinte forma: www.foo.com/?say=Hi&to=Mom. Com o método POST, os dados serão enviados no body da requisição, pois esse método é utilizado para que o browser envie dados e obtenha uma resposta do servidor. A figura abaixo ilustra a resposta de uma requisição similar à da imagem acima, porém com o método POST no lugar do método GET. Pode-se observar na imagem os headers da requisição, e logo abaixo os dados sendo enviados no body.

Figura 5 – Method POST.

```
POST / HTTP/1.1
Host: foo.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 13

say=Hi&to=Mom
```

Fonte: https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Sending_and_retrieving_form_data.

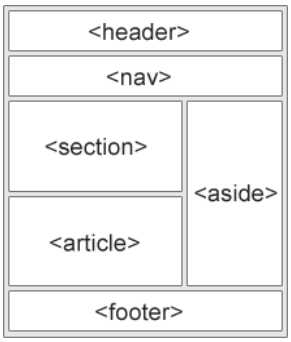
HTML5

O HTML foi criado por Tim Berners-Lee, na tentativa de criar uma linguagem que fosse entendida por meios de acesso diferentes. Na década de 90 ele ganhou força e passou a ser adotado por fabricantes de browsers, compartilhando as mesmas convenções. Entre 1993 e 1995 surgiram novas versões do HTML (HTML+, HTML 2.0 e HTML 3.0), nas quais foram propostas diversas mudanças para enriquecer a linguagem, porém ela ainda não era tratada como padrão. Em 1997 o W3C trabalhou na padronização da linguagem, gerando a versão 3.2. Em 2004 foi fundado o WHATWG (Web Hypertext Application Tecnnology Working Group) por empresas como Mozilla, Apple e Opera, onde se iniciaram os trabalhos de desenvolvimento da nova versão do HTML, hoje chamada de HTML5.

O HTML5 é a nova versão do HTML4, e um dos seus principais objetivos é facilitar a manipulação dos elementos. Ele fornece ferramentas para o CSS e o Javascript atuarem de forma mais fácil, deixando a página mais leve e funcional. Foram criadas novas tags, outras foram descontinuadas e outras modificadas. Todas as mudanças foram realizadas de forma a manter a retro compatibilidade, ou seja, páginas desenvolvidas em versões anteriores continuam funcionando.

Foram criados elementos semânticos para facilitar a organização das páginas, de forma a facilitar o entendimento do desenvolvedor e do browser. Por exemplo, no HTML4, para definir um footer para a página, era comum os desenvolvedores criarem uma div com o id chamado footer, e lá dentro colocarem o conteúdo do rodapé da página. No HTML5 foi criada a tag footer para subrir essa necessidade. Da mesma forma, foi criada uma tag para o header da página, uma tag para seções da página, chamada section, entre outros. A imagem abaixo exemplifica alguns elementos semânticos criados pelo HTML5 para definir diferentes partes da página.

Figura 6 – Elementos Semânticos HTML5.



Fonte: https://www.w3schools.com/html/html5_semantic_elements.asp**.**

No HTML foram criadas diversas tags de multimídia, como a tag video que permite reproduzir vídeos dentro da página. Antes do HTML5, isso só era possível com o auxílio de plug-ins, como por exemplo o Flash. É possível executar comandos sobre o vídeo através do Javascript, como iniciar ou parar o vídeo. Da mesma forma também foi criada a tag audio, que também permite reproduzir arquivos de áudios diretamente do browser. Antes do HTML5, isso também só era possível através de plug-ins. Outra tag criada foi o canvas, com a qual é possível desenhar gráficos com o auxílio do Javascript diretamente na página.

Uma importante API criada no HTML5, que facilita muito a vida dos desenvolvedores, é a Web Storage, com a qual é possível armazenar dados localmente no browser do usuário. Antes do HTML5, os dados de aplicação eram armazenados em cookies, que eram enviados em todas requisições ao servidor. O Web Storage, ao contrário dos cookies, não envia dados para o servidor; além de permitir armazenar quantidades maiores de informação sem afetar o desempenho da página. Cada origem possui seu próprio Web Storage, ou seja, todas as páginas de um mesmo domínio conseguem armazenar e acessar os mesmos dados, porém páginas de outros domínios não conseguem ver os dados de outros. Esta API provê dois tipos de objetos para armazenar dados no browser: o localStorage, que armazena os dados sem data de expiração, e o sessionStorage, que mantém os dados durante uma sessão, sendo removidos quando a aba é fechada.

Outra funcionalidade do HTML5 é o WebRTC (Real Time Communication), que é uma tecnologia que permite o streaming de áudio, vídeo e compartilhamento de dados entre dois browsers. WebRTC é um conjunto de padrões que fornece ao navegador a capacidade de compartilhar dados do aplicativo e realizar conferências de ponto a ponto, sem a necessidade de instalar plug-ins ou software de terceiros. Os componentes WebRTC são acessados através APIs JavaScript.

Em relação a conectividade, outra tecnologia disponibilizada pelo HTML5 são os WebSockets. Através dele é possível abrir uma sessão de comunicação interativa entre o browser do usuário e o servidor. Com esta API, é possível enviar mensagens para o servidor e receber respostas orientadas a eventos, sem ter que consultar o servidor para obter uma resposta. Isso permite que o servidor possa enviar dados para o browser sem que ele tenha pedido. Esta tecnologia é muito útil para sistemas em tempo real, por exemplo dashboards de monitoramento.

Ao invés da página ter que ficar fazendo requisições de tempos em tempos ao servidor para verificar se há novos dados (técnica também chamada de pooling), assim que o servidor tiver alguma informação nova para disponibilizar ao browser, ele pode enviá-las por meio da conexão que está aberta através do WebSocket. Isso evita que seja feita uma sobrecarga desnecessária ao servidor através do pooling de vários browsers tentando verificar se há dados novos para recuperar.

Outra API interessante é a de geocalização, que permite à aplicação, mediante a concessão de permissão do usuário, buscar as posições geográficas do usuário. Esta API é muito útil para sites que utilizam a localização do usuário para lhe oferecer melhores ofertas e sugestões, por exemplo. Além dessa API, também há outras funcionalidades que auxiliam muito a utilização das páginas em dispositivos móveis, como a API de acesso a câmera e a capacidade de entender qual a rotação da tela, de forma a permitir adaptar o layout caso a tela esteja na vertical ou na horizontal, e até mesmo a capacidade de detectar eventos de touchscreen e permitir que a aplicação responda a eles através de ações com o Javascript.

O HTML5 trouxe diversas tecnologias que melhoram muito o desenvolvimento de páginas web. Neste capítulo foi dada uma visão geral de algumas

das novas funcionalidades. Caso deseje aprofundar melhor no conteúdo, consulte a documentação da Mozilla sobre o tema, que é bem completa e detalhada, no link abaixo:

- https://developer.mozilla.org/pt-BR/docs/Web/HTML/HTML5.

CSS

CSS (Cascading Style Sheets) é uma linguagem utilizada para definir a aparência de documentos que adotam para o seu desenvolvimento linguagens de marcação, como XML, HTML e XHTML. Ele define como serão exibidos os elementos contidos no código de um documento, e a sua maior vantagem é efetuar a separação entre o formato e o conteúdo de um documento.

O CSS foi criado porque à medida que a tecnologia foi evoluindo, passou a ser necessário melhorar a apresentação dos documentos, de forma a deixá-los mais atrativos aos usuários. Com isso, foram criadas no HTML novas tags e atributos de estilo, de forma que o HTML passou a exercer, além da função de estruturar o documento, a função de estilizá-lo. Porém, isso começou a gerar problemas para os desenvolvedores, pois não havia uma forma de definir um estilo padrão para alguns elementos em específico, por exemplo estilizar todos os cabeçalhos de um projeto da mesma forma, e assim era preciso alterar manualmente cada arquivo. Para solucionar esse tipo de problema é que nasceu o CSS, sendo desenvolvido primeiramente para habilitar a separação do conteúdo e formato de um documento de sua estilização. Dessa forma, as cores e formatos de fonte, por exemplo, poderiam ficar separadas da estrutura do documento, podendo assim serem reaproveitas por várias páginas. Esta separação aumentou esta flexibilidade, favorecendo o compartilhamento do formato e reduzindo a repetição de código na estrutura dos documentos.

A sintaxe do CSS é bem simples, e define a forma como o estilo será aplicado aos elementos HTML. Ela é composta por três partes principais: um seletor, uma propriedade e o valor que será aplicado. O seletor seria, por exemplo, um elemento HTML como uma div ou body, no qual a regra escrita será aplicada. A propriedade é

o atributo do elemento no qual a regra será aplicada, como por exemplo color, font ou position. O valor é a característica que será aplicada na propriedade do seletor informado, por exemplo uma determinada cor ou tamanho de fonte. A figura abaixo ilustra um exemplo de uma regra CSS, no qual é aplicada a cor preta no fundo do elemento body. O body seria o seletor, a propriedade o background-color e o valor seria #000.

Figura 7 – Exemplo de Regra CSS.

```
1. body{
2.   background-color: #000;
3. }
```

Fonte: http://tableless.github.io/iniciantes/manual/css/.

CSS pode ser aplicado de três formas: a primeira é chamada de inline, e consistem em aplicar o CSS a um elemento do HTML, utilizando seu atributo style. Dessa forma, o CSS é aplicado somente naquele elemento em específico. Outros elementos do mesmo tipo na página não receberão este estilo aplicado. A figura abaixo exemplifica esta forma.

Figura 8 – CSS Inline.

```
1. <p style="color: blue">Parágrafo com fonte azul.</p>
2. <p>Esse outro parágrafo não é azul, a não ser que
3. exista <span style="color: red">CSS em outro lugar</span>.</p>
```

Fonte: http://tableless.github.io/iniciantes/manual/css/.

A segunda maneira de aplicar CSS é chamada de CSS interno e consiste em utilizar a tag style dentro do head de uma página HTML. Desta forma, o CSS é válido para toda a página. Assim, o estilo aplicado em um seletor em específico valerá para toda a página. Por exemplo, caso seja aplicado uma determinada fonte no elemento label, todos os labels da página teriam esta regra aplicada. A figura abaixo ilustra o CSS interno.

Figura 9 – CSS Interno.

```
<head>
  <style type="text/css">
    seletor { propriedade: valor; }
  </style>
</head>
```

Fonte: http://tableless.github.io/iniciantes/manual/css/.

A terceira forma de aplicar CSS é chamada de CSS externo. Esta forma consiste em criar um ou mais arquivos com extensão .css e incluí-los na estrutura head do documento HTML. A regra de aplicação é similar ao CSS interno, de forma que um estilo aplicado a um seletor específico valerá para toda a página. A diferença do externo para o interno é que no externo as regras são encapsuladas em um arquivo que pode ser reaproveitado por várias páginas, e no interno essas regras deveriam ser copiadas e coladas em cada uma das páginas que fossem utilizá-las, o que não é o ideal, pois no caso de uma mudança, ela deveria ser replicada manualmente em todas as páginas, ao contrário do externo, no qual a mudança seria feita somente em um arquivo. A imagem abaixo ilustra o CSS externo.

Figura 10 – CSS Externo.

```
<head>
  <link rel="stylesheet" type="text/css" href="reset.css">
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
```

Fonte: http://tableless.github.io/iniciantes/manual/css/.

Um seletor CSS, além de poder ser um elemento como um body ou div, também pode ser uma class ou um id. Uma class é uma forma de identificar um grupo de elementos. Através dela pode-se atribuir regras a vários elementos de uma vez. Ao escrever um CSS para uma class, todos os elementos que tiverem a mesma propriedade class serão afetados. A imagem abaixo ilustra um CSS para uma class, e sua utilização em várias divs.

Figura 11 – CSS em class.

```css
.classe1 {
  background: blue;
}
```

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title></title>
    <meta charset="utf-8">
  </head>
  <body>
    <div class="classe1">Div1</div>
    <div class="classe1">Div2</div>
    <div class="classe1">Div3</div>
    <div class="classe1">Div4</div>
    <div class="classe1">Div5</div>
  </body>
</html>
```

Fonte: http://tableless.github.io/iniciantes/manual/css/.

Id é uma forma de identificar um elemento, e ele deve ser único para cada elemento. Através dele pode-se atribuir um CSS para um elemento em específico, ao contrário da class. A imagem abaixo ilustra um CSS para alguns ids em específico, e sua utilização no HTML.

Figura 12 – CSS em id.

```css
#idUm {
  background: blue;
}
#idDois {
  background: yellow;
}
#idTres {
  background: lightblue;
}
#idQuatro {
  background: lightgreen;
}
```

```html
<!DOCTYPE html>
<html lang="pt-br">
  <head>
    <title></title>
    <meta charset="utf-8">
  </head>
  <body>
    <div id="idUm">Div1</div>
    <div id="idDois">Div2</div>
    <div id="idTres">Div3</div>
    <div id="idQuatro">Div4</div>
  </body>
</html>
```

Fonte: http://tableless.github.io/iniciantes/manual/css/.

O CSS3 é a mais recente evolução do CSS. Ele é uma extensão da versão anterior, a CSS2.1. Ele trouxe uma série de novidades muito úteis, como a possibilidade de utilizar cantos arredondados, sombras, gradientes, transições ou animações, além de novos layouts como multicolunas, caixas flexíveis e layouts de

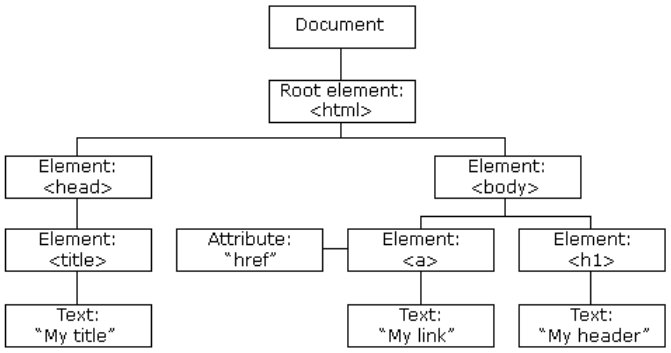
grid. Caso deseje aprofundar melhor no conteúdo, consulte o tutorial do w3schools, que é muito completo e exemplificado, no link abaixo:

- https://www.w3schools.com/css/.

DOM

O DOM é uma interface de programação para documentos HTML e XML. Ele provê uma representação de estrutura do documento e define um meio pelo qual a estrutura pode ser acessada por programas, permitindo assim que sua estrutura, estilo e conteúdo sejam alterados. Quando uma página web é carregada por um browser, é criado o seu DOM da página, representando a sua árvore de objetos. Com o DOM, o Javascript consegue criar páginas dinâmicas, sendo possível trocar elementos e atributos da página, alterar CSS, remover e incluir elementos e atributos do HTML e reagir ou criar eventos na página. O DOM é um padrão da W3C que define padrões para acesso e manipulação do conteúdo da página. As imagens abaixo ilustram como a árvore de objetos do DOM é estruturada.

Figura 13 – Árvore de Objetos do DOM.



Fonte: https://www.w3schools.com/js/js_htmldom.asp.

O DOM provê uma interface que pode ser acessada através de linguagens de programação como o Javascript. Apesar de geralmente ser manipulado através do Javascript, o DOM foi desenvolvido para ser independente de qualquer linguagem de

programação específica, de forma a possuir uma única API representando a estrutura do documento que possa ser acessada por qualquer linguagem de programação, como por exemplo Python.

No DOM, todos os elementos são definidos como objetos, e sua interface provê acesso às propriedades e métodos de cada objeto. Uma propriedade é um valor que pode ser buscado ou alterado, como um texto de um elemento HTML, e um método é uma ação que pode ser feita, como adicionar ou remover um elemento HTML. Por exemplo:

```
document.getElementById("exemplo").innerHTML = "Hello World!";
```

O código acima busca no DOM o elemento que possui o id “exemplo” através do método getElementById, e altera seu conteúdo utilizando a propriedade innerHTML.

O objeto document representa a página web. Para acessar qualquer elemento HTML da página, deve-se iniciar acessando o objeto document. Além do getElementById, também é possível encontrar um elemento pelo nome da tag utilizando o getElementsByTagName e através da sua classe com o método getElementsByClassName. Ao contrário do getElementById, que retorna um único objeto, estes dois métodos retornam um objeto do tipo HTMLCollection, que é uma lista dos objetos encontrados, pois é possível ter mais de um objeto na página com a mesma tag e classe.

Outra funcionalidade interessante do DOM é o Event, que permite ao Javascript reagir a eventos do HTML. Um código Javascript pode ser executado quando um evento acontece, por exemplo quando um usuário clica em um elemento HTML. Um evento pode ser executado em diversas ocasiões, por exemplo quando uma página é carregada, quando uma imagem é carregada, quando o mouse passa por cima de um elemento, quando o valor de um input é alterado, entre outros. Segue um código de exemplo de um event:

```
<h1 onclick="this.innerHTML = 'Clique realizado!'">Clique no texto</h1>.
```

Quando o usuário clica no texto, o código dentro do atributo onclick é executado, trocando o conteúdo do elemento. Há três formas de registrar um evento em um elemento do DOM. Uma seria diretamente no atributo, conforme o exemplo acima. Ela não é a forma mais indicada por deixar a tag maior e menos legível, além de misturar os interesses de estrutura e comportamento, fazendo com que um bug seja mais difícil de encontrar. Outra forma seria utilizar o método addEventListener no elemento do DOM através do Javascript, por exemplo:

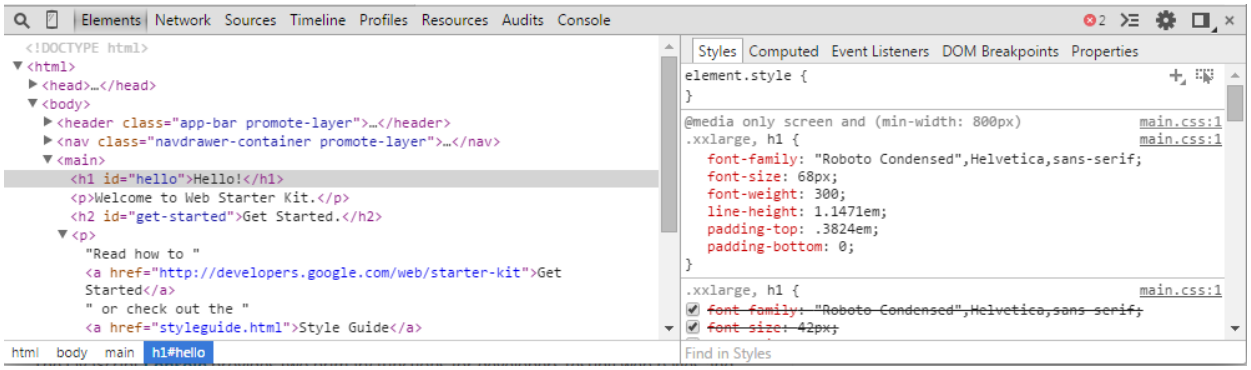
```
myButton.addEventListener('click', function(){alert('Hello world');}, false);
```

A terceira forma, seria possível passar uma função para o atributo correspondente ao evento desejado do elemento, através do Javascript, por exemplo:

```
myButton.onclick = function(event){alert('Hello world');};
```

Os navegadores modernos possuem uma parte dedicada aos desenvolvedores, permitindo a eles que naveguem pelos nós do DOM inspecionando suas propriedades ou até mesmo alterando valores para testar. Através dele, também é possível ver o tráfego de rede que a página está fazendo, ver e debuggar código Javascript, e através do console é possível ver os alertas que a aplicação está gerando, como mensagens de erro. A figura abaixo ilustra o Chrome Developer Tools, disponível no Google Chrome.

Figura 14 – Chrome Developer Tools.



Fonte: https://developer.chrome.com/devtools.

Referências

DEVMEDIA. *O que é HTML5*. Disponível em: <https://www.devmedia.com.br/o-que-e-o-html5/25820>. Acesso em: 12 de nov. 2020.

GETTING STARTED. *O que é CSS?*. 2020. Disponível em: <tableless.github.io/iniciantes/manual/css/>. Acesso em: 12 de nov. 2020.

GETTING STARTED. *O que é HTML? Linguagem base dos websites*. Disponível em: <tableless.github.io/iniciantes/manual/html/index.html>. Acesso em: 12 de nov. 2020.

LUIZ, Andrey. JavaScript #1 - Uma breve história da linguagem. Disponível em: <shipit.resultadosdigitais.com.br/blog/javascript-1-uma-breve-historia-da-linguagem/>. Acesso em: 12 de nov. 2020.

MDN WEB DOCS. *CSS3*. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Archive/CSS3>. Acesso em: 12 de nov. 2020.

MDN WEB DOCS. *Events and the DOM*. 2020. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Events>. Acesso em: 12 de nov. 2020.

MDN WEB DOCS. *HTML5*. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTML/HTML5>. Acesso em: 12 de nov. 2020.

MDN WEB DOCS. *JavaScript*. 2020. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Aprender/JavaScript>. Acesso em: 12 de nov. 2020.

MDN WEB DOCS. *Web forms — Working with user data*. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Forms>. Acesso em: 12 de nov. 2020.

OPENJS FOUNDATION. *Home*. 2020. Disponível em: <https://nodejs.org/en/>. Acesso em: 12 de nov. 2020.

TOOLS FOR WEB DEVELOPERS. *Chrome DevTools*. 2017. Disponível em: <https://developers.google.com/web/tools/chrome-

devtools?utm_source=dcc&utm_medium=redirect&utm_campaign=2018Q2>. Acesso em: 12 de nov. 2020.

W3SCHOOLS.COM. *CSS Tutorial*. 2020. Disponível em: <https://www.w3schools.com/css/>. Acesso em: 12 de nov. 2020.

W3SCHOOLS.COM. *Home.* 2020. Disponível em: <https://www.w3schools.com/html/default.asp>. Acesso em: 12 de nov. 2020.

W3SCHOOLS.COM. *JavaScript HTML DOM*. 2020. Disponível em: <https://www.w3schools.com/js/js_htmldom.asp>. Acesso em: 12 de nov. 2020.

W3SCHOOLS.COM. *JavaScript Tutorial*. 2020. Disponível em: <https://www.w3schools.com/js/>. Acesso em: 12 de nov. 2020.