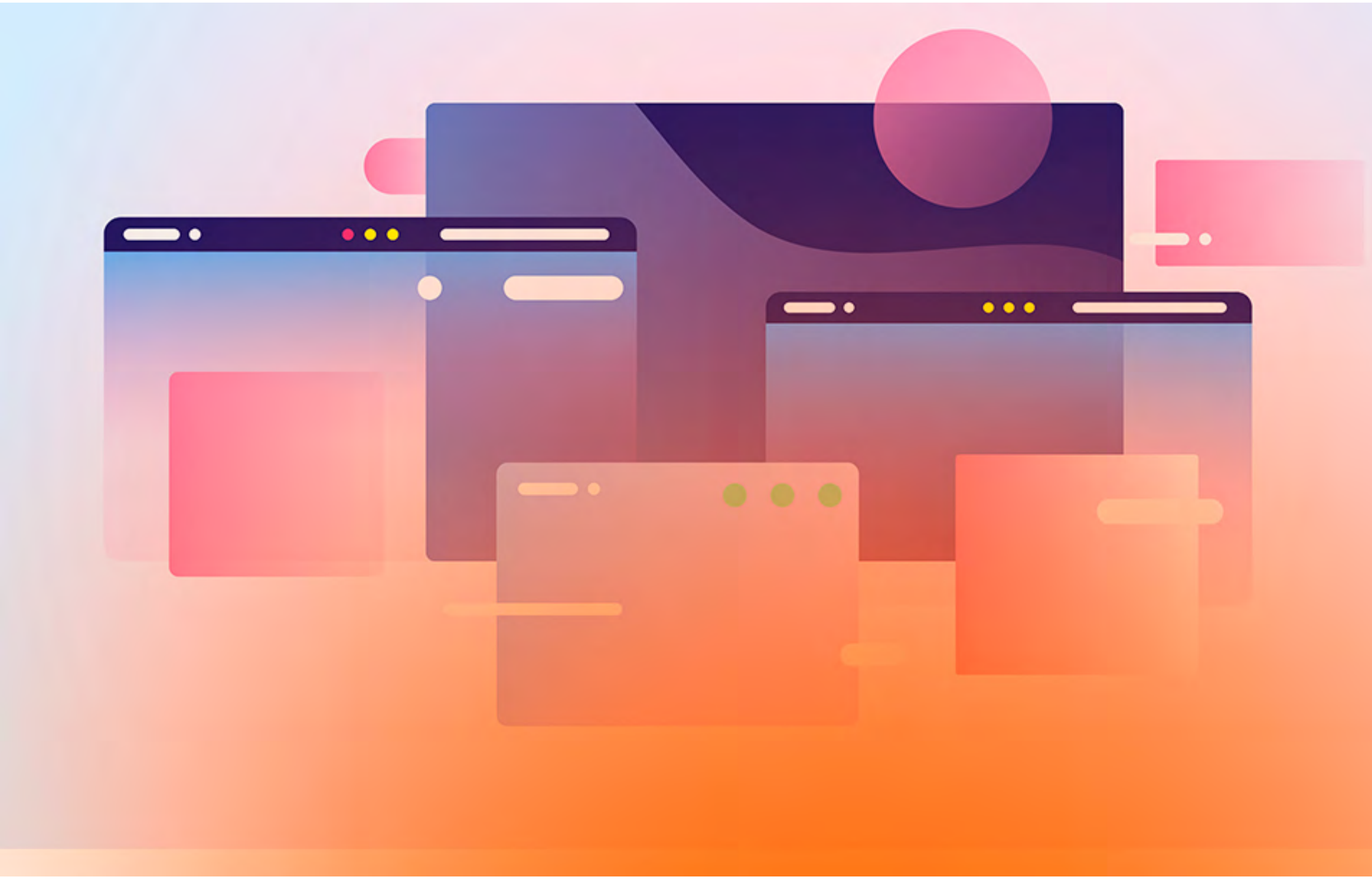
SALVANDO DADOS NO NAVEGADOR

Unidade 04



Sumário

Olá, **Fullturist,**

Seja bem-vindo(a)!

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. Deve ter se deparado também com jogos divertidos e interativos que prenderam a sua atenção por um bom tempo. Você já se perguntou como tudo isso é feito? Para desenvolver algo parecido, você precisa dominar linguagens que vão além do HTML e CSS.

Nesta trilha, você irá aprender a linguagem JavaScript, que irá possibilitar que você crie sites e aplicativos semelhantes ao Facebook, ao Instagram e ao Tik Tok. JavaScript é uma linguagem de programação muito utilizada na Web.

A partir de agora, você será capaz de desenvolver funcionalidades mais complexas. Quando uma página Web, como Facebook e Instagram, faz mais do que apenas mostrar informações estáticas e, ao invés disso, mostram em tempo real conteúdos atualizados, mapas interativos, animações gráficas em 2D/3D, entre outros, você pode ter certeza de que o JavaScript está por traz de tudo. Ficou empolgado(a)? Então, vamos lá!

Unidade 04

Salvando Dados no Navegador

Olá, Fullturist!

Esta é a unidade 4 da trilha de aprendizagem JavaScript. Na unidade anterior, certamente você aprendeu muita coisa interessante sobre o DOM (Document Object Model) e ficou empolgado(a) com as possibilidades de desenvolver páginas e aplicativos dinâmicos. Além disso, você também aprendeu a desenvolver formulários profissionais e validação de informações com JavaScript. Mas temos mais assuntos para explorar nesta última unidade! Agora, iremos conhecer os principais recursos do HTML 5, que vão permitir que você desenvolva aplicativos web ainda mais interessantes. Esses recursos são chamados de APIs que podem ser acessadas pelo JavaScript. Animado(a) ou curioso(a)? Vamos botar a mão na massa e aproveitar toda essa empolgação!

Objetivos de aprendizagem

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

- conhecer as principais APIs disponíveis no HTML 5;
- entender que o HTML 5 não é apenas uma linguagem de marcação;
- utilizar a API Web Storage para armazenar dados local no dispositivo do usuário;
- salvar dados na sessão do navegador através do recurso sessionStorage;
- salvar e persistir os dados no disco local do dispositivo do usuário através do recurso localStorage;
- compreender como implementar um login e um cadastro de consumo de combustível.

4.1 HTML 5

O HTML5 não se trata apenas de uma linguagem de marcação, mas também de um conjunto de novas funcionalidades: as APIs, que podem ser acessadas via JavaScript. Nesta unidade vamos te apresentar uma série de APIs existentes no HTML5, mas vamos focar na API Web storage que, em conjunto com JavaScript, irá permitir que você desenvolva aplicativos e sites com armazenamento local, ou seja, armazenamento na máquina onde o navegador está sendo executado.

As APIs HTML5 beneficiam o desenvolvedor iniciante e o desenvolvedor experiente. Os desenvolvedores iniciantes sofrem para escrever códigos JavaScript devido à falta de experiência e conhecimento. E é neste ponto que as APIs HTML 5 vêm nos socorrer! As APIs HTML 5 são um conjunto de bibliotecas JS que podem ser usadas diretamente em arquivos HTML sem incorporar qualquer código JavaScript personalizado. Isso permite que o desenvolvedor iniciante desenvolva recursos superinteressantes, mesmo não tendo muito conhecimento com JavaScript. Para os desenvolvedores mais experientes, as APIs HTML 5 ajudam a ganhar tempo no desenvolvimento, uma vez que muitas funcionalidades já estão prontas para serem usadas. Vamos conhecer essas APIs? Então, vamos começar listando, na tabela a seguir, todas as APIs disponíveis hoje. Cabe destacar apenas que o HTML 5 está em constante evolução, assim, tenha certeza de que esta lista de funcionalidades vai crescer muito com o passar do tempo.

API	Funcionalidades
Ambient Light API	Fornece informações sobre os níveis de luz ambiente, conforme detectado por um sensor de luz do dispositivo.
Battery Status API	Fornece informações sobre o status da bateria do dispositivo.
Canvas 2D Context	Permite desenhar e manipular gráficos em um navegador.
Clipboard API	Fornece acesso às funcionalidades de copiar, recortar e colar do sistema operacional.
Contacts	Permite acesso ao repositório de contatos de um usuário no navegador da web.
Drag and Drop	Suporta arrastar e soltar itens dentro e entre as janelas do navegador.
File API	Fornece programas com acesso seguro ao sistema de arquivos do dispositivo.
Forms	Oferece aos programas acesso aos novos tipos de dados definidos em HTML5. Inclusive você já usou durante o curso!
Fullscreen API	Controla o uso da tela inteira do usuário para páginas da web, sem a interface de usuário do navegador.
Gamepad API	Suporta entrada de controladores de gamepad USB.
Geolocation	Fornece aplicativos da web com acesso a dados de localização geográfica sobre o dispositivo do usuário.
getUserMedia/Stream API	Fornece acesso a dados de dispositivos externos (como vídeo de webcam).
History API	Permite que os programas manipulem o histórico do navegador.
HTML Microdata	Fornece uma maneira de anotar o conteúdo com rótulos legíveis por computador.

Indexed database	Cria um sistema de banco de dados simples do lado do cliente no navegador da web.
Internationalization API	Fornece acesso à formatação sensível ao local e comparação de strings. Muito utilizado para preparar o aplicativo para trabalhar com múltiplos idiomas.
Offline apps	Permite que os programadores disponibilizem aplicativos da web no modo offline.
Proximity API	Fornece informações sobre a distância entre um dispositivo e um objeto.
Screen Orientation	Lê o estado de orientação da tela (retrato ou paisagem) e dá aos programadores a capacidade de saber quando ele muda e travá-lo no lugar.
Selection	Oferece suporte à seleção de elementos em JavaScript usando seletores de estilo CSS.
Server-sent events	Permite que o servidor envie dados para o navegador sem que o navegador precise solicitá-los.
User Timing API	Oferece aos programadores acesso a carimbos de data / hora de alta precisão para medir o desempenho dos aplicativos.
Vibration API	Permite acesso à funcionalidade de vibração do dispositivo.
Web Audio API	Permite acesso à funcionalidade de vibração do dispositivo.
Web Messaging	Permite que as janelas do navegador se comuniquem entre si de diferentes origens.
Web Speech API	Fornece entrada de voz e recursos de saída de texto para voz.
Web storage	Permite o armazenamento de pares de valores-chave no navegador. É o nosso foco de estudo nesta unidade!
Web sockets	Abre uma sessão de comunicação interativa entre o navegador e o servidor.
Web Workers	Permite que o JavaScript execute scripts em segundo plano.

XMLHttpRequest2	Utilizado para desenvolver com Ajax! Essa API aprimora XMLHttpRequest para eliminar a necessidade de contornar os erros de política de mesma origem e para fazer XMLHttpRequest funcionar com novos recursos de HTML5.

Fonte: Elaborada pelo autor (2021).

Para saber mais sobre algumas das APIs, acesse:

https://developer.mozilla.org/en-US/docs/Web/API/AmbientLightSensor

https://developer.mozilla.org/en-US/docs/Web/API

Que lista longa, não? Existem muitos outros recursos, vale a pena você acessar as fontes e conhecer ainda mais sobre as APIs. E não se preocupe, você não precisa saber programar e saber utilizar todas, o importante é saber que as APIs existem e onde você pode pesquisar e aprender ainda mais sobre elas.

Você vai observar, durante a sua vida como desenvolvedor, que você aprende e evolui seus conhecimentos conforme as necessidades forem surgindo!

Nesta unidade, então, vamos focar na API Web Storage, porque neste momento da sua carreira como desenvolvedor a necessidade é essa! Vamos então logo direto ao ponto e aprender sobre Web Storage? Vamos em frente!

4.2. API Web Storage

A API Web Storage é um conjunto de códigos JavaScripts prontos de armazenamento da Web. Essa API fornece formas que permite que você programe para que os navegadores armazenem dados localmente. Esses dados são armazenados no formato de pares de chave e valor, de uma forma muito mais intuitiva do que usando cookies.

Um cookie são pequenas informações armazenadas no computador do usuário quando ele acessa um determinado site por meio de um navegador da web. Os cookies são usados para personalizar a experiência do usuário na web com um site. São muito utilizados pelos e-commerces e redes sociais como Facebook e Instagram. Porém, o usuário pode personalizar seu navegador para aceitar, rejeitar ou excluir esses cookies.

Quando necessitamos fazer uso da API Web Storage em nossos sites e aplicativos para armazenar informações no dispositivo do usuário, nós podemos armazenar essas informações na sessão e/ou no próprio disco local. Vamos ver com mais detalhes como isso funciona?

4.2.1 Conceitos e Formas de Uso da API Web Storage

Como dito anteriormente, você pode utilizar o armazenamento de informações por meio da API Web Storage armazenando dados na sessão ou no disco local do dispositivo do usuário. Cada uma das formas pode ser utilizada com objetivos bem específicos. Para o armazenamento, são utilizados os dois recursos abaixo:

> sessionStorage – permite armazenar dados apenas para uma sessão. Uma sessão é iniciada pelo navegador quando você acessa determinado site ou aplicativo. As sessões são temporárias, o que significa que quando você fechar o navegador, esta sessão será apagada e os dados perdidos. É muito utilizado quando fazemos login em um site ou aplicativo. Enquanto o navegador estiver aberto, você consegue acessar os formulários e dados, pois os dados de login estão armazenados

na sessão. Quando você fechar o navegador e abrir novamente, você terá que fazer login novamente. O navegador mantém uma sessão separada para cada site ou aplicativo e, mesmo que você recarregue o site ou aplicativo, os dados permanecem. Os dados armazenados em uma sessão nunca são transferidos para o servidor (enviados ao backend) e o limite de armazenamento é de 5 MB.

localStorage – possui o funcionamento bem semelhante ao sessionStorage, mas a grande diferença é que as informações persistem! Ou seja, elas permanecem salvas mesmo se você fechar o navegador e abrir novamente. O localStorage armazena dados sem data de expiração, isso quer dizer que eles vão permanecer sempre e somente serão apagados por meio do JavaScript ou se você executar a limpeza do cache do navegador e dos dados armazenados localmente.

Aposto que agora você está imaginando inúmeras possibilidades de uso da API Web Storage, não está? E aposto também que você está curioso e muito ansioso para ver como tudo isso funciona. Então, vamos aos exemplos.

4.3 Armazenando Dados no sessionStorage

Primeiramente, vamos a um exemplo de uso de armazenamentos de dados na sessão através do recurso sessionStorage. No nosso exemplo, vamos criar um formulário de login que irá coletar os dados de acesso a uma página de cadastro de consumo de combustível que iremos implementar como exemplo de uso do localStorage. Para começar, crie um arquivo HTML chamado login.html com o código 1 e um arquivo chamado login.js conforme o Código 2.

```
<!DOCTYPE html>
<html lang=”pt-br”>
  <head>
    <meta charset=”UTF-8”>
    <meta http-equiv=”X-UA-Compatible” content=”IE=edge”>
    <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
    <title>Registro de Consumo de Combustivel - Login</title>
    <style>
```

```
    h1{
      width:60%;
      margin:20px auto;
    }
    #container{
      border:1px solid blue;
      width:30%;
      padding:10px;
      margin:0 auto;
    }
  </style>
</head>
<body>
  <h1>App de Registro de Consumo de Combustivel V1.0</h1>
  <div id="container">
    <form id="form-login" method="GET" action="index.html">
      <p>
        Login<br>
        <input type="email" id="login" placeholder="Seu e-mail" />
      </p>
      <p>
        Senha<br>
        <input type="password" id="senha" />
      </p>
      <p>
        <input type="submit" id="entrar" value="Entrar >>" />
      </p>
    </form>
  </div>
  <script src="login.js"></script>
</body>
</html>
```

Fonte: Elaborada pelo autor (2021).

Nossa página de login possui um design bem simples pois, como sempre, o foco é na funcionalidade e programação com Javascript. Mas fique à vontade para utilizar seus conhecimentos de HTML e CSS para deixar a página mais bonita e com mais estilo.

```
const login = 'fullture@fullture.com.br';
const senha = 'sucesso';
let paragrafoErro;
const div_campos = document.getElementById('campos');
const form = document.getElementById('form-login');
form.addEventListener('submit', function(event){
    event.preventDefault();
    const inputLogin = form.querySelector('#login');
    inputLogin.addEventListener('keyup', limpaMensagemErro);
    const inputSenha = form.querySelector('#senha');
    inputSenha.addEventListener('keyup', limpaMensagemErro);
    if ( validaLoginSenha(inputLogin, inputSenha) === false )
        return;
    myStorage = window.sessionStorage;
    myStorage.setItem('logado', true);
    window.location = 'index.html';
});

function exibeMensagem(mensagem){
    paragrafoErro = document.createElement('p');
    paragrafoErro.style.color = 'red';
    paragrafoErro.id = 'paragrafoErro';
    paragrafoErro.innerHTML = mensagem;
    div_campos.append(paragrafoErro);
}
function limpaMensagemErro(){
    if (paragrafoErro){
        div_campos.removeChild(paragrafoErro);
        paragrafoErro = null;
    }
}
```

```
function validaLoginSenha(inputLogin, inputSenha){
    if ( inputLogin.value === “” ){
        exibeMensagem(“É necessário informar o login”);
        return false;
    }

    if ( inputSenha.value === “” ){
        exibeMensagem(“É necessário informar a senha”);
        return false;
    }

    if ( inputLogin.value !== login ){
        exibeMensagem(“Login inválido!”);
        return false;
    }
    if ( inputSenha.value !== senha ){
        exibeMensagem(“Senha inválida”);
        return false;
    }
    return true;
}
```

Fonte: Elaborada pelo autor (2021).

O código 2 é um código Javascript que realiza algumas tarefas importantes, como validar os campos login e senha e exibir uma mensagem caso um deles esteja em branco ou inválido e armazenar a informação no sessionStorage. Vamos entender melhor o código JavaScript acima. Como não temos acesso a um banco de dados, foi definido o login e a senha para acesso nas constantes da linha 1 e linha 2. Na linha 3 foi declarado uma variável paragrafoErro. Essa variável irá armazenar um elemento parágrafo que será criado dinamicamente na função exibeMensagem() (linhas 28 a 36) e irá apresentar a mensagem adequada para cada campo informado. Esta variável foi criada como global e no início do código pois precisamos acessá-la nas funções exibeMensagem() e limpaMensagemErro() (linhas 38 a 43). A função limpaMensagemErro() será responsável por limpar a mensagem de erro quando qualquer tecla for pressionada nos campos login e senha. Por isso, ela está atrelada aos eventos de keyup dos campos conforme pode ser visto na linha 14 e linha 17. O

que acontece no evento submit do formulário? Primeiramente, na linha 11, a ação de submissão do formulário é anulada, dessa forma a ação do form definido pelo atributo action não ocorrerá. Na linha 19 a função validaLoginSenha() é invocada e os valores digitados nos campos são passados como argumento para esta função. Então, se as informações de login não forem especificadas ou forem inválidas, a função validaLoginSenha() retorna falso e o restante da execução após a linha 19 não acontecerá e a mensagem de erro será exibida. Agora, quero chamar sua atenção para o ponto principal desse código. Na linha 22 está sendo utilizado o sessionStorage. O recurso sessionStorage é acessado através do objeto window que representa a tela do navegador. Então, primeiramente a sessão está sendo acessada e armazenada na constante myStorage. Uma vez estando de posse do acesso ao recurso sessionStorage, na linha 23 está sendo armazenado uma informação indicando que o usuário está sendo logado. O formato de armazenamento é chave:valor, em que a chave é uma espécie de variável, que, neste caso, demos o nome de 'logado', e o valor é true, o que indica que o usuário está logado. Agora as informações estão salvas na sessão do navegador e permanecerão armazenadas que o navegador seja fechado. Depois de armazenar os dados no sessionStorage, o site é redirecionado para a página index (linha 25). Para finalizar o nosso exemplo, segue o código da página index.html e o código JavaScript do arquivo index.js.

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registro de Consumo de Combustivel</title>
</head>
<body>
  <h1>Registro de Consumo de Combustivel</h1>
   <button id="logout">Sair</button>
 <script src='index.js'></script>
</body>
</html>
```

Fonte: Elaborada pelo autor (2021).

```
const myStorage = window.sessionStorage;
let logado = myStorage.getItem('logado');
if (logado){
   console.log("Usuario logado no sistema");
}else{
   window.location = 'login.html';
}
const logout = document.getElementById('logout');
logout.addEventListener('click', function(){
   myStorage.removeItem('logado');
   window.location = 'login.html';
});
```

Fonte: Elaborada pelo autor (2021).

No código Javascript, na linha 1 estamos recuperando o acesso ao recurso sessionStorage e na linha 2 está sendo acessada a informação armazenada na sessão do navegador através do método myStorage.getItem(). O nome da chave está sendo passado como argumento e o valor relacionado é acessado e armazenado na variável logado. No evento de click do botão logout, na linha 13, está sendo removida a informação armazenada na sessão através do método myStorage.removeItem(), no qual estamos passando como argumento a chave armazenada na sessão, em seguida, a página está sendo redirecionada para a página de login. Faça um teste, na barra de endereço do navegador, tente acessar novamente a página index.html. Você perceberá que não será possível! Isso porque, ao recuperar a informação da sessão na linha 2, um valor null é retornado, pois esta chave não existe mais, logo, o else será sempre executado e a página de login apresentada.

Para saber mais sobre o recurso sessionStorage, acesse o endereço https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage.

Até aqui você aprendeu como armazenar dados na sessão do navegador através de um exemplo de login. Se você fechar o navegador, as informações armazenadas serão perdidas. Agora, vamos estudar o recurso localStorage que utilizaremos para armazenar, persistir informações no disco local do dispositivo do usuário.

4.4 Armazenado Dados no localStorage

Vamos continuar evoluindo o nosso exemplo de um sistema de controle de consumo de combustível. Agora vamos implementar um formulário onde o usuário vai informar a quantidade de combustível e o valor pago. Altere o código do arquivo index.html conforme o código abaixo.

```
const myStorage = window.sessionStorage;
let logado = myStorage.getItem('logado');
if (logado){
   console.log("Usuario logado no sistema");
}else{
   window.location = 'login.html';
}
const logout = document.getElementById('logout');
logout.addEventListener('click', function(){
   myStorage.removeItem('logado');
   window.location = 'login.html';
});
```

Fonte: Elaborada pelo autor (2021).

No código acima estamos criando um formulário simples com dois campos, uma tabela sem dados onde serão exibidos de forma dinâmica os dados salvos no localStorage, que será realizado no evento de click do botão salvar.

Agora, altere o código Javascript do arquivo index.js de acordo com o código abaixo.

```
const myStorage = window.sessionStorage;
let logado = myStorage.getItem('logado');
if (!logado){
   window.location = 'login.html';
}
const dados = [];
function recuperaDados() {
   const registros = localStorage.getItem('consumo');
   const lista = JSON.parse(registros);
   for(let item of lista) {
      adicionaItemTable(item);
   }
} recuperaDados();
const logout = document.getElementById('logout');
logout.addEventListener('click', function(){
   console.log('sair');
   myStorage.removeItem('logado');
   window.location = 'login.html';
});
const form = document.getElementById('form');
form.addEventListener('submit', function(event){
   event.preventDefault();
   const litros = form.querySelector('#litros');
   const valor = form.querySelector('#valor');
   const obj = adicionaDados(litros, valor);
   adicionaItemTable(obj);
   litros.value = '';
   valor.value = '';
});
const btnSalvar = document.getElementById('salvar');
btnSalvar.addEventListener('click', function(){
   const dadosJSON = JSON.stringify(dados);
   localStorage.setItem('consumo', dadosJSON);
```

```
    alert('Dados salvo com sucesso!');
});
function adicionaDados(litros, valor){
    let obj = {
        litros:litros.value,
        valor:valor.value
    }
    dados.push(obj);
    return obj;
}
function adicionaItemTable(obj) {
    const tbody = document.querySelector('tbody');
    let td = document.createElement('td');
    td.innerHTML = obj.litros;
    let td2 = document.createElement('td');
    td2.innerHTML = obj.valor;
    let tr = document.createElement('tr');
    tr.appendChild(td);
    tr.appendChild(td2);
    tbody.appendChild(tr);
}
```

Fonte: Elaborada pelo autor (2021).

Neste momento, chamamos a sua atenção para as linhas 48 e 49 do evento de click do botão adicionar. Na linha 49, estamos acessando o recurso localStorage e executando o armazenamento local do array que estamos utilizando para guardar os objetos com os dados digitados pelo usuário. Mas antes, na linha 48, precisamos converter o array com os objetos no formato JSON utilizando o método JSON. stringify(dados). É preciso fazer esta conversão pois o localStorage armazena dados apenas no formato JSON. Teste a página adicionando informações, clique no botão salvar e depois feche o navegador e abra-o novamente. Perceba que os dados cadastrados anteriormente foram recuperados e exibidos na página. Como isso aconteceu?

Aconteceu porque, na função recuperaDados(), implementada entre as linhas 10 e 18, estamos recuperando o dados do localStorage e atualizando a tabela. Na linha 11 estão sendo recuperados os dados salvos através do método getItem() do localStorage. Esses dados são recuperados no formato JSON, e, agora, para que a tabela seja atualizada, é preciso executar o processo inverso. Na linha 12 está sendo convertido o formato JSON para o formato array e, assim, a tabela pode ser atualizada.

Para saber mais sobre o recurso localStorage, acesse o endereço https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

Depois de estudar e entender o que pode ser feito com a API Web Storage, imagina as funcionalidades que você poderá implementar nos seus sites de aplicativos daqui para frente? A forma de armazenamento sempre será a mesma, então, mão na massa e comece a criar e desenvolver!

Nesta trilha, você adquiriu conhecimentos sólidos sobre a linguagem JavaScript. Esses conhecimentos permitirão criar páginas e aplicativos web mais robustos e dinâmicos. Além disso, esses conhecimentos o(a) ajudarão a aprender muitas outras tecnologias e frameworks baseados em JavaScript utilizados no mercado de trabalho. O conhecimento evolui de forma constante, continue estudando, visite as referências indicadas nas unidades e dedique-se! Dessa forma, você será um(a) Dev de sucesso e muitas possibilidades surgirão para você.

Síntese

Nesta unidade, você conheceu as APIs HTML 5 disponíveis atualmente. Lembrando que o HTML 5 está em constante evolução, então vale ficar atento para novas atualizações que podem ser lançadas. Dentre estas APIs, você estudou, implementou e testou a API Web Storage, que permite o armazenamento de dados na sessão do navegador e no disco local do dispositivo do usuário. Vamos a um resumo rápido de tudo que você estudou nesta unidade:

- As aplicações precisam salvar dados e esses dados podem ser salvos no disco local do dispositivo do usuário por meio do recurso localStorage, sem a necessidade de integração com o backend.

- Salvar dados na sessão utilizando o recurso sessionStorage é muito útil para verificação de login, mas esses dados são perdidos quando o navegador é fechado.

- Esta unidade permitiu a você ter uma percepção que desde o momento inicial do curso você consegue criar aplicações com uma certa complexidade e salvar os dados inseridos pelo usuário do lado do navegador.

- O usuário não vai consultar os dados em outro computador porque não está disponibilizado na rede. Mas é possível criar aplicações que trabalham com o armazenamento do navegador, como fizemos na aplicação de exemplo.

Fullture Insights

- HTML Tutorial. W3School Web Docs, 2021. Disponível em: https://www.w3schools.com/html/default.asp. Acesso em: 22 set. 2021.

- WEB APIs. MDN Web Docs, 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API. Acesso em: 23 set. 2021.

- Window.localStorage. MDN Web Docs, 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage. Acesso em: 24 set. 2021.

- Window.sessionStorage. MDN Web Docs, 2021. Disponível em: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage. Acesso em: 24 set. 2021.

FU
LL
TU
RE

www.fullture.com