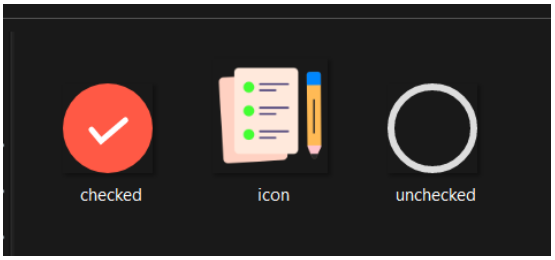
Dom – To DO List

Un "To-Do List" o lista de tareas es una herramienta que se utiliza para organizar y priorizar las tareas que se necesitan realizar. Es una lista de las tareas pendientes y las tareas completadas que te ayuda a llevar un seguimiento de lo que necesitas hacer y a asegurarte de que no se te olvide ninguna tarea importante.

Los " To-Do List " se utilizan en diversos ámbitos, tanto personales como profesionales. Por ejemplo, una persona puede utilizar una lista de tareas para organizarse en su día a día, mientras que una empresa puede utilizar una lista de tareas para asegurarse de que se están cumpliendo todos los objetivos y metas de la empresa.

Las imágenes que vamos a implementar en nuestro código serán 3, estas las buscaran y las colocaran en la carpeta **images/** preferiblemente en formato png, o si quieren utilizar iconos también lo pueden hacer.



@hdtoledo

Empecemos con el código HTML:

```html
<body>
    <div class="container">
        <div class="title">Lista de Pendientes<img src="images/icon.png" alt=""></div>
        <div class="todo-list-bar">
            <input type="text" id="task" placeholder="Ingrese una tarea para guardar ... " autocomplete="off">
            <button onclick="clicked()" class="addBtn">Agregar</button>
        </div>
        <ul class="lists">
        </ul>
    </div>
    <script src="js/script.js"></script>
</body>
```

1. ** **: Crea un contenedor para los elementos de la lista de pendientes.
2. ** Lista de Pendientes<img src="images/icon.png" alt=""> **: Crea un encabezado para la lista de pendientes, con un ícono al lado derecho. El texto "Lista de Pendientes" se muestra como título.
3. ** **: Crea un contenedor para el ingreso de tareas.
4. **<input type="text" id="task" placeholder="Ingrese una tarea para guardar..." autocomplete="off">**: Crea un campo de entrada de texto para que el usuario pueda escribir una nueva tarea. El atributo **placeholder** muestra un texto de ayuda en el campo antes de que se ingrese algún valor. El atributo **autocomplete** deshabilita la función de autocompletado.
5. **<button onclick="clicked()" class="addBtn">Agregar</button>**: Crea un botón "Agregar" que el usuario puede hacer clic para agregar la tarea que ingresó en el campo de entrada de texto.
6. **<ul class="lists">
**: Crea una lista no ordenada (ul) vacía para mostrar las tareas pendientes. La lista se llenará con las tareas agregadas por el usuario.

Vamos con el CSS:

```css
@import url("https://fonts.googleapis.com/css2?family=Poppins:wght@100;200;300;400;500;600;700;800;900&display=swap");

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
  font-family: "Poppins", sans-serif;
}

body {
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  background: linear-gradient(125deg, red, orange);
  flex-direction: column;
}

.container {
  width: 100%;
  max-width: 420px;
  background-color: white;
  padding: 20px;
  border-radius: 10px;
}
```

```css
.title {
  display: flex;
  align-items: center;
  justify-content: center;
  font-size: 30px;
  font-weight: bold;
  margin-bottom: 10px;
}

.title img {
  margin-left: 10px;
  width: 40px;
  height: 40px;
}

.todo-list-bar {
  width: 100%;
  height: 35px;
  background-color: gainsboro;
  border-radius: 20px;
  display: flex;
  justify-content: space-between;
}

input {
  flex: 1;
  padding: 10px 20px;
  border-radius: 10px;
  border: none;
  outline: none;
  background-color: transparent;
}
```

```css
.addBtn {
  padding: 5px 20px;
  font-size: 15px;
  outline: none;
  border: none;
  border-radius: 20px;
  background-color: #ff3939;
  cursor: pointer;
  color: white;
}

ul {
  padding: 10px;
}

ul li {
  list-style: none;
  font-size: 15px;
  padding: 5px 5px 5px 35px;
  position: relative;
  cursor: pointer;
}

ul li::before {
  content: "";
  position: absolute;
  height: 16px;
  width: 16px;
  border-radius: 50%;
  background-image: url("../images/unchecked.png");
  background-size: cover;
  background-position: center;
  top: 50%;
  left: 0px;
  transform: translate(50%, -50%);
}
```

@hdtoledo

```css
ul li.checked {
  text-decoration: line-through;
}

ul li.checked::before {
  background-image: url("../images/checked.png");
  background-size: cover;
  background-position: center;
  top: 50%;
  left: 0px;
  transform: translate(50%, -50%);
}

span {
  position: absolute;
  font-size: 16px;
  width: 20px;
  height: 20px;
  top: 50%;
  right: 0;
  line-height: 21px;
  transform: translate(-50%, -50%);
  padding-left: 4.5px;
  border-radius: 50%;
  cursor: pointer;
}

span:hover {
  background: gainsboro;
}
```

Dentro del código nos econtraremos un, **ul li** se utiliza para aplicar un estilo CSS a todos los elementos **li** que se encuentran dentro de un elemento **ul**. Al utilizar **ul li**, se está especificando que se desea aplicar un estilo a todos los elementos de lista que se encuentran dentro de la lista no ordenada.

Usar **ul li** en lugar de simplemente **li** es una forma más específica de aplicar un estilo a los elementos de la lista, ya que se limita a los elementos de lista que se encuentran dentro de una lista no ordenada y no a otros elementos **li** que puedan existir en la página.

ul li::before es un selector de CSS que se utiliza para agregar contenido antes del contenido de cada elemento **li** que se encuentra dentro de un elemento **ul**. En otras palabras, se utiliza para agregar un elemento o texto adicional antes del texto de cada elemento de una lista no ordenada.

Ahora vamos con el js:

```
const tarea = document.getElementById("task");
const lista = document.querySelector(".lists");


function clicked(){
    if(tarea.value===""){
        alert("La tarea no debe estar vacia")
    }else{
        let li = document.createElement("li");
        li.innerHTML = tarea.value;
        lista.appendChild(li);
        tarea.value = "";
        let span = document.createElement("span");
        span.innerHTML = "\u00d7";
        li.appendChild(span);
    }
    guardar();
}

lista.addEventListener("click",(e)=>{
    if(e.target.tagName === "LI"){
        e.target.classList.toggle("checked");
    }else if(e.target.tagName === "SPAN"){
        e.target.parentElement.remove();
    }
    guardar();
})

function guardar(){
    localStorage.setItem("data",lista.innerHTML);
}

function getData(){
    lista.innerHTML = localStorage.getItem("data");
}

getData();
```

- **const tarea = document.getElementById("task");** busca el elemento del DOM con el ID "task" y lo asigna a la variable **tarea**.

- **const lista = document.querySelector(".lists");** busca el elemento del DOM con la clase "lists" y lo asigna a la variable **lista**.
- **function clicked(){...}** define una función que se ejecuta cuando se hace clic en el botón "Agregar".
- **if(tarea.value===""){...}** verifica si el valor del elemento de entrada (input) está vacío. Si lo está, muestra una alerta de que la tarea no debe estar vacía. Si no lo está, crea un elemento de lista (**li**) y lo agrega a la lista (**lista**). También agrega un elemento **span** con una "X" para eliminar la tarea, y al final limpia el elemento de entrada.
- **lista.addEventListener("click",(e)=>{...})** agrega un evento de clic a la lista (**lista**). Si el elemento clicado es una tarea (**LI**), se le agrega o quita la clase "checked" para indicar si ha sido completada o no. Si el elemento clicado es un **SPAN**, se elimina el elemento padre (**LI**).
- **function guardar(){...}** guarda la lista actual en el almacenamiento local del navegador utilizando **localStorage.setItem**.
- **function getData(){...}** obtiene la lista guardada del almacenamiento local del navegador utilizando **localStorage.getItem**.
- **getData();** carga la lista guardada en la página al cargar el script.

LocalStorage es una característica de JavaScript que permite a una página web almacenar datos en el navegador web de un usuario. Estos datos se guardan en el navegador y permanecen disponibles incluso después de que se cierra la página o se apaga el equipo.

En otras palabras, LocalStorage es una forma de almacenamiento en el lado del cliente que permite a las aplicaciones web guardar y recuperar datos localmente, lo que significa que los datos persisten incluso después de que se cierre el navegador.

LocalStorage utiliza una API de almacenamiento de clave-valor, lo que significa que los datos se almacenan como pares de clave-valor, donde la clave es una cadena que identifica al valor almacenado.

En nuestro código, el método **localStorage.setItem()** se utiliza para guardar el contenido de la lista de tareas en el LocalStorage del navegador. Y el método **localStorage.getItem()** se utiliza para recuperar los datos de la lista de tareas desde el LocalStorage y mostrarlos en la página web. De esta manera, cuando el usuario vuelve a visitar la página, la lista de tareas guardada en el LocalStorage se recupera y se muestra en la página web.