



CFP/UAX

Centro de Formación Profesional

Resumen de la Práctica Examen

DESARROLLO WEB EN ENTORNO CLIENTE

Vianca Tereza Franco Rodriguez

ENUNCIADO

1- Manipulación del DOM

- Buscar elementos
- Movernos entre los elementos del DOM Acceder a parente, hijo...
- Crear y borrar elementos.
- Para cada uno de los puntos no es necesario utilizar todos, con usar un par de ellos es suficiente.

2 - Manipulación del BOM

- Window Y control de tiempo.

3 - Validación

- ReportValidity () o CheckValiditz (se trata de validar con html y usas una de las dos).

4 - Eventos

- addEventListener.

5 - Ajax

- Json y PHP

6 - Jquery

- Efectos (hacer uno o dos efectos).

7 - Consumición de API

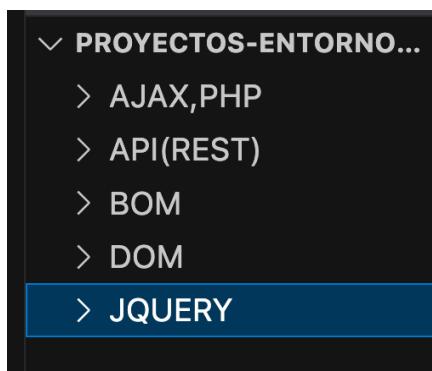
- (He usado Fetch para consumir la API) Get por lo menos (lo demás para subir nota).

Introducción

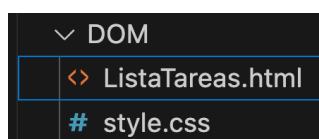
Este proyecto ha sido desarrollado de manera estructurada, planteando cada enunciado como un ejercicio individual. Para cada uno de estos enunciados, se ha creado y ejecutado un ejercicio específico, abordando todos los parámetros y puntos solicitados en detalle. No obstante, es importante destacar que algunos aspectos como la validación mediante **checkValidity()** o la implementación de eventos **addEventListener()** se han utilizado de manera anidada en otros ejercicios.

Resumen

Para iniciar, se presenta una captura que destaca la organización general del proyecto, donde cada carpeta contiene sus respectivos archivos.



1- Manipulación del DOM:



Para la ejecución del ejercicio, se ha desarrollado una aplicación que permite al usuario agregar tareas a una lista, así como quitarlas individualmente o eliminarlas todas de una vez. También, se agregó la funcionalidad de organizar las tareas alfabéticamente, ya sea de forma ascendente o descendente.

En primer lugar, para agregar tareas, se utiliza el botón Agregar Tarea, que invoca la función ***createTask()***. Esta función crea dinámicamente un nuevo elemento de lista **** con un campo de entrada de texto para ingresar la tarea y un botón para eliminarla.

En segundo lugar, para eliminar tareas, se emplea el botón Eliminar Tareas, el cual llama a la función ***clearTasks()***, la cual elimina todas las tareas presentes en la lista. También se utiliza la función ***removeTask(e)*** para eliminar una tarea específica cuando se hace clic en el botón Quitar Tarea.

En tercer lugar, para ordenar las tareas alfabéticamente, se proporcionan dos botones: Ordenar Tareas de Forma Ascendente o Ordenar Tareas de Forma Descendente. Al hacer clic en ellos, se activa la función ***sortByTitle(order)***, que ordena las tareas en la lista alfabéticamente.

```

<body>
    <!--Contenedor central-->
    <div class="container">
        <h1>Lista de Tareas</h1>
        <div>
            <button class="add-task" onclick="createTask()">Agregar Tarea</button>
            <button class="delete-task" onclick="clearTasks()">Eliminar Tareas</button>
        </div>

        <!-- Contenedor para mostrar la lista de tareas -->
        <div>
            <ul id="tasks"></ul>
        </div>

        <h2>Ordenar Tareas Por orden alfabetico</h2>
        <div class="child-container">
            <button class="upward" onclick="sortByTitle('asc')">Ordenar Tareas de Forma Ascendente</button>
            <button class="falling" onclick="sortByTitle()">Ordenar Tareas de Forma Descendente</button>
        </div>
    </div>

    <script>
        // Función para ordenar las tareas por orden alfabetico
        const sortByTitle = (order) => {
            // Obtiene la lista de tareas del HTML
            const tasks = document.getElementById("tasks");

            // Convierte la lista de tareas en un array
            // Luego ordena el array usando el metodo sort, comparando el valor de cada elemento
            const taskList = Array.from(tasks.children).sort((a, b) => {
                // Si el orden es ascendente, compara el valor de la primera etiqueta con el valor de la segunda
                if (order === "asc") {
                    return a.children[0].value.localeCompare(b.children[0].value);
                } else { // Si no, compara el valor de la segunda etiqueta con el valor de la primera
                    return b.children[0].value.localeCompare(a.children[0].value);
                }
            });
            // Limpia la lista de tareas
            tasks.innerHTML = "";
            // Agrega la lista de tareas ordenada al HTML
            taskList.forEach(task => tasks.appendChild(task));
        }

        // Función para borrar una tarea
        const removeTask = (e) => {
            // Obtiene la lista de tareas del HTML
            const tasks = document.getElementById("tasks");
            // Obtiene la tarea que se va a borrar
            const task = e.currentTarget.parentNode;
            // Elimina la tarea de la lista
            tasks.removeChild(task);
        }

        // Función para borrar todas las tareas
        const clearTasks = () => {
            // Obtiene la lista de tareas del HTML
            const tasks = document.getElementById("tasks");
            // Limpia la lista de tareas
            tasks.innerHTML = "";
        }
    </script>

```

```
// Función para agregar una nueva tarea
const createTask = () => {
    // Obtiene la lista de tareas del HTML
    const tasks = document.getElementById("tasks");

    // Crea una nueva tarea
    const task = document.createElement("li");
    task.classList.add("todo-item");

    // Crea un input para la nueva tarea
    const input = document.createElement("input");
    input.classList.add("todo-input");
    input.type = "text";
    task.appendChild(input);

    // Crea un botón para borrar la tarea
    const deleteButton = document.createElement("button");
    deleteButton.textContent = "Quitar Tarea";
    deleteButton.classList.add("delete-task");
    deleteButton.onclick = removeTask;
    task.appendChild(deleteButton);

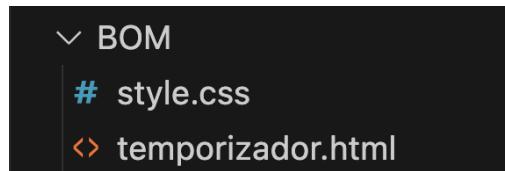
    // Agrega la nueva tarea a la lista
    tasks.appendChild(task);
}

</script>
</body>
```

Capture de la Aplicación en el Navegador



2 - Manipulación del BOM



Para la ejecución del ejercicio, se ha desarrollado una aplicación para ayudar al usuario a controlar su tiempo de ejercicio, equipada con un temporizador que permite iniciar, pausar, reiniciar o detener el tiempo durante la actividad.

En primer lugar, se crearon cuatro botones dentro de un contenedor HTML, cada uno con una función específica asociada: **start()** para iniciar el temporizador, **stop()** para detenerlo, **reset()** para reiniciar a cero y **finish()** para mostrar un mensaje de alerta con el tiempo transcurrido.

En segundo lugar, se crea un temporizador al inicio de la aplicación con la función **setTimeout()**, que transcurridos 1 segundos (1000ms), lanza una alerta utilizando **alert()**, mostrando un mensaje de bienvenida.

En tercer lugar, se utilizó la función **setTimer(time)** para descomponer el tiempo dado en horas, minutos, segundos y milisegundos, asegurando así un formato legible en la interfaz de usuario. También se introdujo la función **resetTimer()**, que reinicia el temporizador, estableciendo el tiempo en cero y actualizando la interfaz para mostrar cero horas, cero minutos, cero segundos y cero milisegundos del tiempo transcurrido.

En el desarrollo de esta aplicación, se emplearon elementos del BOM para mejorar su funcionalidad. Específicamente, se utilizaron dos métodos clave: **clearInterval()** para detener intervalos previamente establecidos y **setInterval()** para ejecutar acciones repetidas con un intervalo de tiempo específico. Además, se aprovechó la función **alert()** del BOM para mostrar un mensaje de alerta al usuario. Estos recursos, combinados con otros elementos, contribuyeron significativamente a la usabilidad y eficacia de la aplicación.



```
<body>
  <div class="container">

    <h1>Temporizador para tus Rutinas de Ejercicios</h1>

    <div id="timer">00:00:00:00</div>

    <div class="buttonContainer">
      <!-- &#9658 simbolo de HTML -->
      <button id="start" onclick="start()">Iniciar &#9658</button>
      <button id="stop" onclick="stop()" disabled>Pausar &x23F8</button>
      <button id="reset" onclick="reset()" disabled>Reiniciar &#8634</button>
      <button id="finish" onclick="finish()" disabled>Finalizar &#10003</button>
    </div>
  </div>

<script>
  // Almacena el tiempo de inicio del temporizador
  let startTime;

  // Almacena el intervalo del temporizador
  let interval;

  // Almacena el tiempo transcurrido
  let elapsedTime = 0;

  // Indica si el temporizador está en ejecución o detenido
  let running = false;

  // Se muestra un mensaje al segundo con un alert
  setTimeout(() => {
    alert("Bienvenido, a tu temporizador! Para comenzar, presiona el botón Iniciar.");
  }, 1000)

  // Función para iniciar el temporizador
  const start = () => {
    // Si el temporizador no está en ejecución
    if (!running) {
      // Establece el tiempo de inicio como el tiempo actual menos el tiempo transcurrido
      startTime = Date.now() - elapsedTime;
      // Inicia el intervalo del temporizador, actualizándolo cada 10 milisegundos
      interval = setInterval(updateTimer, 10);
      // Marca el temporizador como en ejecución
      running = true;
      // Habilitado de los botones
      document.getElementById("start").disabled = true;
      document.getElementById("stop").disabled = false;
      document.getElementById("reset").disabled = true;
      document.getElementById("finish").disabled = true;
    }
  }

  const stop = () => {
    // Si el temporizador está en ejecución
    if (running) {
      // Detiene el intervalo
      clearInterval(interval);
      // Marca el temporizador como detenido
      running = false;
      // Habilitado de los botones
      document.getElementById("start").disabled = false;
      document.getElementById("stop").disabled = true;
      document.getElementById("reset").disabled = false;
      document.getElementById("finish").disabled = false;
    }
  }

```



```
const reset = () => {
    // Detiene el intervalo
    clearInterval(interval);
    // Resetea el tiempo transcurrido
    resetTimer();
    // Habilitado de los botones
    document.getElementById("start").disabled = false;
    document.getElementById("stop").disabled = true;
    document.getElementById("reset").disabled = true;
    document.getElementById("finish").disabled = true;
}

const finish = () => {
    // Obtiene el tiempo transcurrido y lo muestra en un alert
    const time = document.getElementById("timer").innerHTML;
    alert(`Has finalizado tu ejercicio en ${time}`);
}

const setTimer = (time) => {
    // Convierte el tiempo en horas, minutos, segundos y milisegundos
    let milliseconds = Math.floor((time % 1000) / 10);
    let seconds = Math.floor((time / 1000) % 60);
    let minutes = Math.floor((time / (1000 * 60)) % 60);
    let hours = Math.floor(time / (1000 * 60 * 60));

    // Formatea el tiempo como HH:MM:SS:MS y lo muestra en el elemento HTML
    const timer = pad(hours) + ":" + pad(minutes) + ":" + pad(seconds) + ":" + padMs(milliseconds);
    document.getElementById("timer").textContent = timer
}

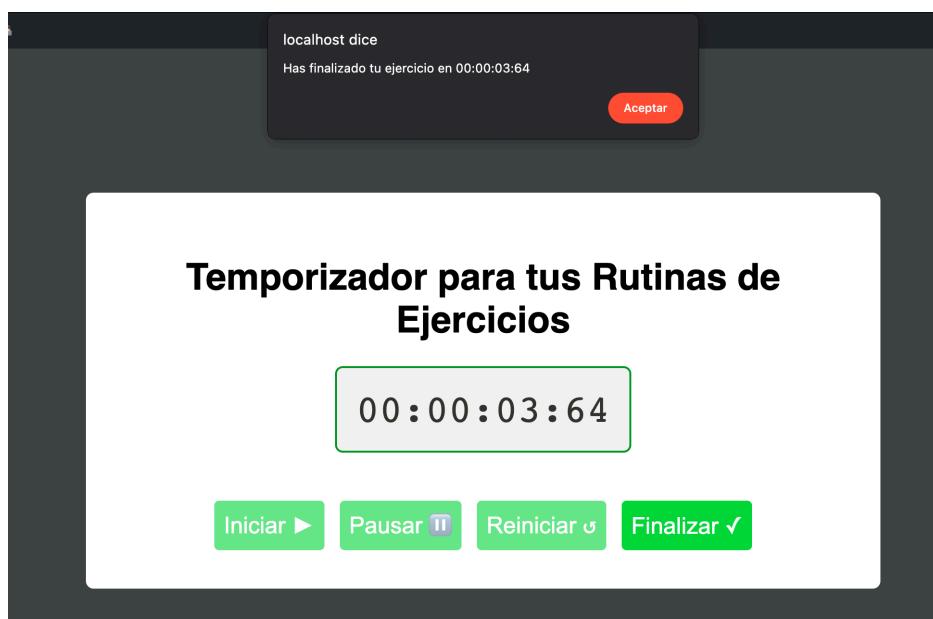
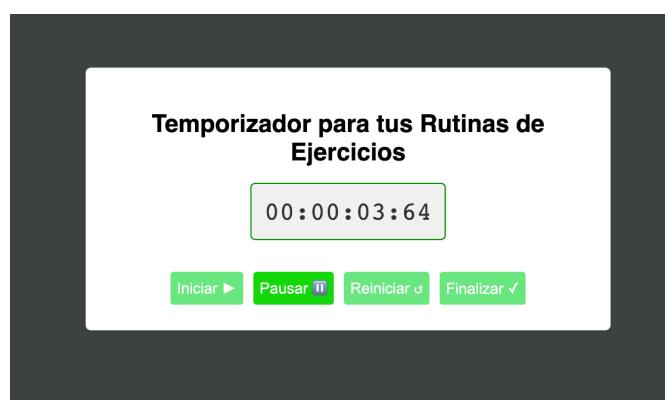
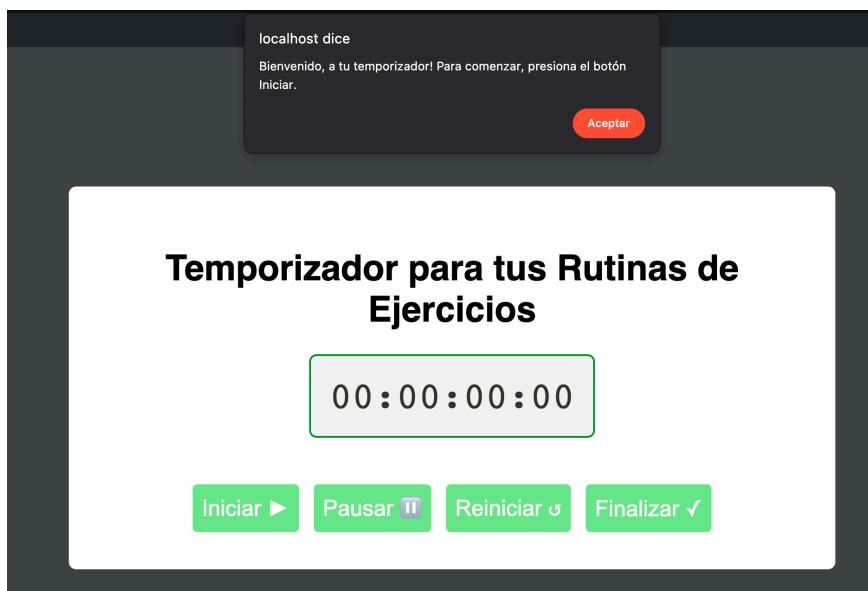
// Función para actualizar el tiempo transcurrido
const updateTimer = () => {
    // Obtiene el tiempo actual
    let currentTime = Date.now();
    // Lo resta al tiempo de inicio
    elapsed = currentTime - startTime;
    // Actualiza el tiempo transcurrido en el HTML
    setTimer(elapsed);
}

// Función para reiniciar el temporizador
const resetTimer = () => {
    // Detiene el intervalo
    clearInterval(interval);
    // Resetea el tiempo transcurrido
    elapsed = 0;
    // Actualiza el tiempo transcurrido en el HTML
    setTimer(0);
    // Marca el temporizador como detenido
    running = false;
}

// Función para formatear el tiempo, con dos cifras
function pad(num) {
    return num.toString().padStart(2, "0");
}

// Función para formatear el tiempo de los milisegundos, con dos cifras
function padMs(num) {
    return num.toString().padStart(2, "0").slice(0, 2);
}
```

Capture de la Aplicación en el Navegador



5 - Ajax con Json y PHP.



Para la ejecución del ejercicio, se ha desarrollado una aplicación de registro para usuarios interesados en un gimnasio. El formulario solicita información básica como nombre, apellido, dirección, teléfono y correo electrónico. Tras completar el registro y enviar el formulario, se muestra al usuario un mensaje de bienvenida junto con una imagen de fondo personalizada.

En primer lugar, se crea un formulario en HTML y se establecen validaciones en HTML5, PHP y JavaScript para garantizar la integridad de los datos ingresados. Posteriormente, se implementa un script en JavaScript para gestionar las interacciones del formulario. Primero, aseguramos que el usuario complete el formulario y haga clic en **Enviar**, lo que desencadena el evento de envío. Luego, se espera a que el documento HTML esté completamente cargado antes de agregar un **EventListener()** al formulario con ID `registroForm`, encargado de manejar el envío del formulario.

En segundo lugar, se evita el envío predeterminado del formulario usando **event.preventDefault()**, lo que permite un control personalizado a través de AJAX. Los datos del formulario se recopilan con **FormData()** para ser enviados al servidor mediante una solicitud AJAX.

En tercer lugar, se crea una instancia de **XMLHttpRequest()** para realizar la solicitud al servidor, configurándose para enviar los datos al archivo PHP **datos.php**. Una vez enviada la solicitud, se establece un manejador de eventos para la respuesta AJAX, verificando su completitud y éxito.

En caso de éxito, se analiza la respuesta **JSON** del servidor con **JSON.parse()**, extrayendo los datos relevantes como el mensaje de bienvenida y los detalles del registro para mostrar al usuario en la página web. En situaciones de error, se muestra un mensaje en la consola del navegador indicando un problema en el proceso de registro y se utiliza **alert()** para notificar al usuario sobre la corrección de los campos del formulario.

index.html

```

<script>
    // Espera a que el documento HTML esté completamente cargado
    document.addEventListener('DOMContentLoaded', () => {
        // Agrega un event listener al formulario con el ID 'registroForm' que escucha el evento de envío del formulario
        document.getElementById('registroForm').addEventListener('submit', (event) => {
            // Evitar el envío del formulario por defecto
            event.preventDefault();

            // Obtiene una referencia al formulario que desencadenó el evento de envío
            const form = event.currentTarget;

            // Verificar si el formulario es válido
            if (form.checkValidity()) {
                // Recopilar datos del formulario
                const formData = new FormData(form);

                // Crear una nueva instancia de XMLHttpRequest para realizar una solicitud AJAX
                const peticion = new XMLHttpRequest();

                // URL del archivo PHP que maneja la solicitud y los datos del formulario
                const url = 'datos.php';

                // Abrir la petición HTTP asíncrona, de tipo POST, en la URL indicada
                peticion.open('POST', url, true);

                // Se crea un manejador de eventos para la respuesta de la solicitud AJAX
                // que se dispara cuando la petición cambia de estado
                peticion.onreadystatechange = () => {
                    // Verificar si la solicitud se ha completado
                    if (peticion.readyState === XMLHttpRequest.DONE) {
                        // Verificar si la solicitud fue exitosa (estado HTTP 200)
                        if (peticion.status === 200) {
                            // Analizar la respuesta JSON del servidor
                            const response = JSON.parse(peticion.responseText);
                            // Construir el mensaje de bienvenida
                            const message = response.message;
                            const userDetails = response.userDetails;

                            // Ocultar el formulario
                            document.getElementById('formContainer').style.display = 'none';
                            // Mostrar el contenedor de bienvenida y los detalles del usuario
                            document.querySelector('.centralContainer').style.display = 'block';
                            document.getElementById('welcomeMessage').textContent = message;
                            document.getElementById('userDetails').textContent = userDetails;

                            // Mostrar un mensaje de error en la consola si la solicitud no fue exitosa
                            console.error('Error en la solicitud: ' + peticion.status);
                        }
                    }
                };
                // Enviar la solicitud AJAX con los datos del formulario
                peticion.send(formData);
            } else {
                // Mostrar un mensaje de alerta si el formulario no es válido
                alert("Por favor, complete correctamente todos los campos del formulario.");
            }
        });
    });

```



```
<!-- Formulario de registro de usuario -->
<div id="formContainer">
    <h2>Registro de Usuario</h2>
    <form id="registroForm">
        <!-- Campo de entrada para el nombre -->
        <label for="nombre">Nombre:</label>
        <input type="text" name="nombre" maxlength="15" pattern="[A-Za-z]{2,15}"
            title="Introduce entre 2 y 15 letras" placeholder="Nombre" required><br><br>

        <!-- Campo de entrada para el apellido -->
        <label for="apellido">Apellido:</label>
        <input type="text" name="apellido" maxlength="20" pattern="[A-Za-z]{2,20}"
            title="Introduce entre 2 y 20 letras" placeholder="Apellido" required><br><br>

        <!-- Campo de entrada para la dirección -->
        <label for="direccion">Dirección:</label>
        <input type="text" name="direccion" placeholder="Dirección" required><br><br>

        <!-- Campo de entrada para el teléfono -->
        <label for="telefono">Teléfono:</label>
        <input type="text" name="telefono" pattern="[0-9]{9}" title="Número de 9 cifras"
            placeholder="Teléfono" required><br><br>

        <!-- Campo de entrada para el correo electrónico -->
        <label for="correo">Correo Electrónico:</label>
        <input type="email" name="correo" placeholder="Correo electrónico" required><br><br>

        <!-- Botón de envío del formulario -->
        <input type="submit" value="Enviar">
    </form>
```

datos.php

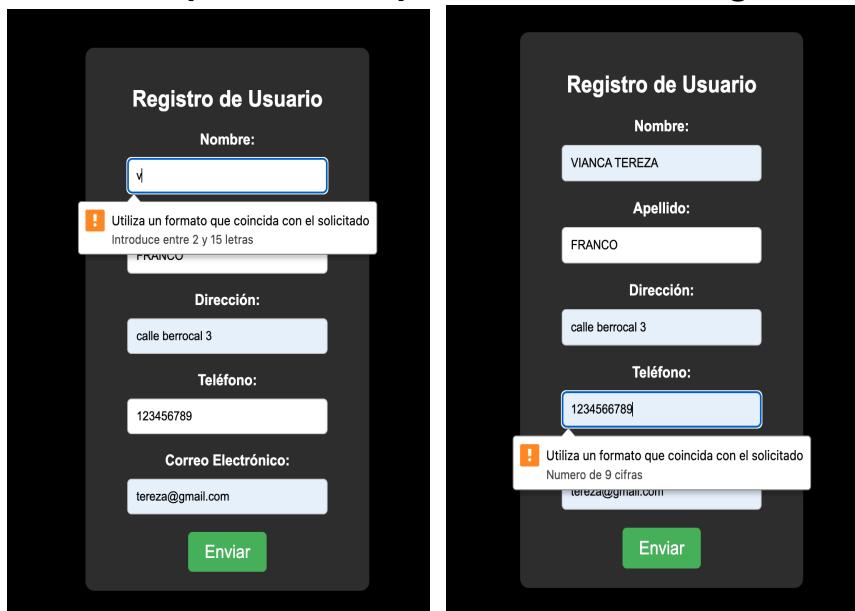
```
<?php
header("Content-Type: application/json; charset=UTF-8");

if (
    $_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST['nombre']) && isset($_POST['apellido']) && isset($_POST['direccion'])
    && isset($_POST['telefono']) && isset($_POST['correo'])
) {
    $nombre = trim($_POST['nombre']);
    $apellido = trim($_POST['apellido']);
    $direccion = trim($_POST['direccion']);
    $telefono = trim($_POST['telefono']);
    $correo = trim($_POST['correo']);

    // Verifica que ninguna variable esté vacía después de quitar los espacios en blanco
    if (!empty($nombre) && !empty($apellido) && !empty($direccion) && !empty($telefono) && !empty($correo)) {

        // Los datos están completos y no contienen espacios en blanco
        $response = array('message' => "¡Enhorabuena, $nombre $apellido! Tu dirección es $direccion, teléfono es $telefono y tu correo es $correo.");
        echo json_encode($response);
    } else {
        // Los datos están vacíos o contienen espacios en blanco
        $response = array('message' => "Error: No se proporcionaron todos los datos requeridos o contienen espacios en blanco.");
        echo json_encode($response);
    }
} else {
    // No se recibieron parámetros
    $response = array('message' => "Error: No se proporcionaron todos los datos requeridos.");
    echo json_encode($response);
}
?>
```

Capture de la Aplicación en el Navegador




Quiero resaltar que en la implementación de este enunciado, he incorporado la solicitud de emplear tanto validaciones como eventos.

Del enunciado 3- Validación, he utilizado *CheckValidity* (Para verificar si el formulario es válido).

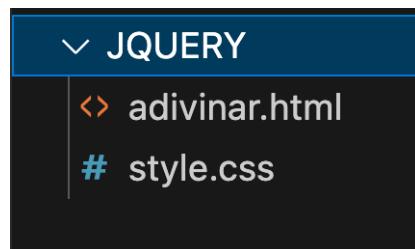
```
// Verificar si el formulario es válido
if (form.checkValidity()) {
    // Recopilar datos del formulario
    const formData = new FormData(form);
```

Del enunciado 4 - Eventos, he utilizado *addEventListener* en dos ocasiones.

```
<script>
    // Espera a que el documento HTML esté completamente cargado
    document.addEventListener('DOMContentLoaded', () => {

        // Agrega un event listener al formulario con el ID 'registroForm' que escucha el evento de envío del formulario
        document.getElementById('registroForm').addEventListener('submit', (event) => {
```

6 - Jquery Efectos (uno o dos).



Para la ejecución del ejercicio, hemos desarrollado una aplicación de juego de memoria en la que el usuario debe recordar y reproducir una secuencia de números mostrada aleatoriamente en cuatro cajas. Al hacer clic en el botón **Jugar**, se muestra una secuencia que el usuario debe replicar. Si tiene éxito, recibe un mensaje de felicitación; de lo contrario, se le informa del error.

En primer lugar, creamos con HTML las cuatro cajas numeradas y un botón **Jugar**, que al hacer clic activa la función **showSequence()**.

En segundo lugar, implementamos el script de JavaScript. Primero, generamos una secuencia aleatoria de 4 elementos, correspondientes al índice de las cajas. Luego, creamos la función **showSequence()** que se encarga de mostrar un mensaje con la secuencia de las cajas durante 1 segundo y luego lo oculta. Para asegurar variedad, utilizamos **shuffleArray()** para barajar la secuencia de manera aleatoria.

Adicionalmente, aplicamos efectos de **hover** y **click** sobre las cajas mediante **jQuery**. Al hacer hover sobre ellas, cambian de color de fondo; mientras que al hacer clic, se produce un efecto de **fadeIn** y **fadeOut**.

Finalmente, la función **checkClick()** determina si la caja seleccionada por el usuario es la correcta dentro de la secuencia. Si la selección es correcta, se incrementa el índice; si se completa toda la secuencia, se muestra un mensaje de felicitación y se reinicia el juego. En caso de selección incorrecta, se muestra un mensaje de error y se da la opción de volver a intentarlo.



```
<body>
  <div>
    <h1>Recuerda la secuencia</h1>
    <div class="box" id="box1">1</div>
    <div class="box" id="box2">2</div>
    <div class="box" id="box3">3</div>
    <div class="box" id="box4">4</div>

    <h3 id="sequence"></h3>
    <br>
    <button onclick="showSequence()">Jugar</button>
  </div>

<script>
  // Generar secuencia aleatoria de 4 elementos, que corresponden al indice de las cajas
  let sequence = shuffleArray([1, 2, 3, 4]);

  // Controla el indice de la caja seleccionada
  let currentIndex = 0;

  // Controla si el juego esta en marcha
  let isPlaying = false;

  // Muestra la secuencia durante 1 segundo y la oculta
  function showSequence() {
    // Cuando se pulse el boton, el juego comienza
    isPlaying = true;
    const sequenceElement = document.getElementById('sequence');

    sequenceElement.innerHTML = "La secuencia es: " + sequence.join(', ');
    setTimeout(() => {
      sequenceElement.innerHTML = ""; // limpia el contenido del elemento que muestra la secuencia.
    }, 1000)
  }

  // Baraja la secuencia
  function shuffleArray(array) {
    return array.sort(() => Math.random() - 0.5);
  }

  // Aplica los efectos de hover y click
  $(document).ready(function () {
    // Hover sobre las cajas, cambia el color de fondo
    $(".box").hover(function () {
      $(this).css("background-color", "darkblue");
    }, function () {
      $(this).css("background-color", "#007bff");
    });

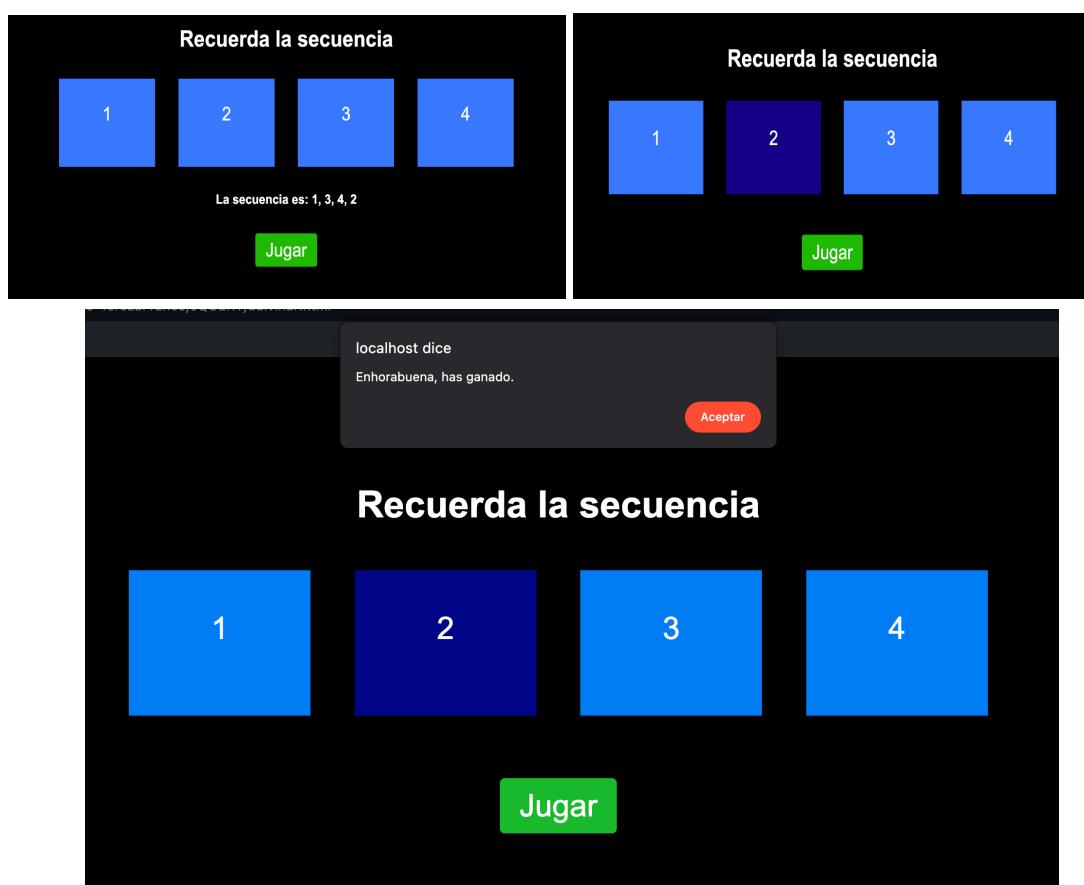
    // Click sobre las cajas, realiza un fadeIn y un fadeOut
    $(".box").click(function () {
      // Solo se realiza si el juego esta en marcha
      if (isPlaying) {
        const boxId = $(this).attr('id').replace('box', '');
        $(this).fadeOut(200).fadeIn(200);
        // Comprueba si la caja seleccionada es correcta
        checkClick(boxId);
      }
    });
  });
}
```

```

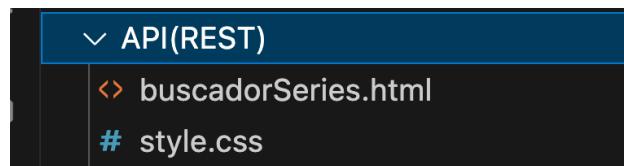
// Comprueba si la caja seleccionada es correcta dentro de la secuencia
function checkClick(boxId) {
  // Si la caja seleccionada es la de la secuencia
  if (parseInt(boxId) === sequence[currentIndex]) {
    // Incrementa el indice
    currentIndex++;
    // Si el indice es la longitud de la secuencia, gana
    if (currentIndex === sequence.length) {
      alert("Enhorabuena, has ganado.");
      // Si el usuario gana, el juego se detiene
      isPlaying = false;
    }
    // Se resetea el indice y se barajan las cajas
    currentIndex = 0;
    sequence = shuffleArray(sequence);
  }
  // Si la caja seleccionada no es la de la secuencia, pierde
  else {
    alert("Ups, caja incorrecta. Has perdido.");
    // Se resetea el indice y se vuelve a mostrar la secuencia
    currentIndex = 0;
    showSequence();
  }
}
};

</script>
  
```

Capture de la Aplicación en el Navegador



7 - Consumición de API



Para la ejecución del ejercicio, hemos desarrollado una aplicación que permite a los usuarios buscar su serie favorita (en inglés) utilizando una **API gratuita** disponible en <https://www.tvmaze.com/api>

En primer lugar, creamos un campo de entrada y un botón para buscar información. El campo de entrada permite al usuario ingresar el nombre de la serie, mientras que el botón de búsqueda activa la función **performSearch()** cuando se hace clic en él.

En segundo lugar, cuando el usuario hace clic en el botón de búsqueda, se muestra un mensaje en la página para informar que la búsqueda está en curso. Luego, utilizamos la función **fetch()** para enviar una solicitud **GET** a la API de TVMaze, incluyendo el nombre de la serie como parámetro en la **URL** de la solicitud.

Cuando recibimos la respuesta de la API, que está en formato **JSON**, la procesamos para extraer información relevante sobre la serie, como el título completo, la sinopsis y la calificación. Cabe destacar, que el **JSON** de la respuesta contiene más de una serie dentro de un Array, ordenado por la serie que más se parezca a nuestra búsqueda por título. Por ello, siempre extraemos el primer elemento de este Array. Esta información se utiliza para crear elementos HTML que mostrarán los detalles de la serie en la página web, utilizando métodos como **createElement()** para crear nuevos elementos e **innerHTML()** para establecer el contenido de los elementos.

Si durante este proceso ocurre algún error, como una conexión perdida o una respuesta inesperada de la API, manejamos este problema mostrando un mensaje de error en la página para informar al usuario de la situación y sugerir posibles acciones a seguir.

```

<body>
  <div class="container">
    <h1>Buscador de Series</h1>
    <p>Por favor, introduzca el nombre de una serie para obtener su resumen (en inglés) y su calificación.</p>
    <input type="text" name="search" id="input">
    <button id="submit" onclick="performSearch()">Buscar</button>
    <div id="show"></div>
  </div>

  <script>
    // Realiza la búsqueda de una serie y muestra sus datos en la página
    const performSearch = () => {
      // Obtiene el nombre de la serie que ha sido introducido
      const input = document.getElementById('input').value;

      // Mostramos un mensaje de espera mientras se realiza la búsqueda
      const show = document.getElementById("show")
      show.innerHTML = "<br>Cargando..."

      // Realiza la búsqueda usando fetch, mediante una petición GET
      fetch(`https://api.tvmaze.com/search/shows?q=${input}`)
        .then(response => response.json()) // Convierte la respuesta a formato JSON
        .then(data => {
          // Obtiene los datos de la serie
          const show = data[0].show;
          // Limpia el elemento HTML de la serie
          cleanShowElem()
          // Crea el elemento HTML de la serie
          createShowElem(show)
        }).catch(error => onError(error));
    }

    // Maneja los errores
    const onError = (error) => {
      // Mostrar detalle del error por consola
      console.error(error);

      // Limpia el elemento HTML de la serie
      cleanShowElem()

      // Muestra un error por pantalla
      const movieElem = document.getElementById("show")
      movieElem.innerHTML = "<br> Ha ocurrido un problema al realizar la búsqueda"
    }

    // Limpia el elemento HTML de la serie
    const cleanShowElem = () => {
      const movieElem = document.getElementById("show")
      movieElem.innerHTML = ""
    }
  </script>

```

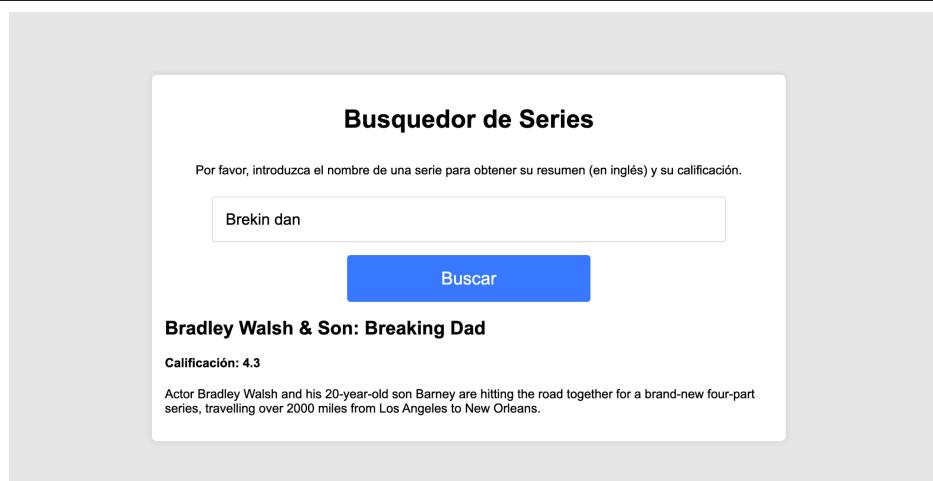
```
// Crea el elemento HTML de la serie
const createShowElem = (show) => {
    // Crea el elemento HTML para mostrar el nombre de la serie
    const name = document.createElement("h2")
    name.innerHTML = show.name;

    // Crea el elemento HTML para mostrar la calificación de la serie
    const rating = document.createElement("h4")
    rating.innerHTML = "Calificación: " + show.rating.average;

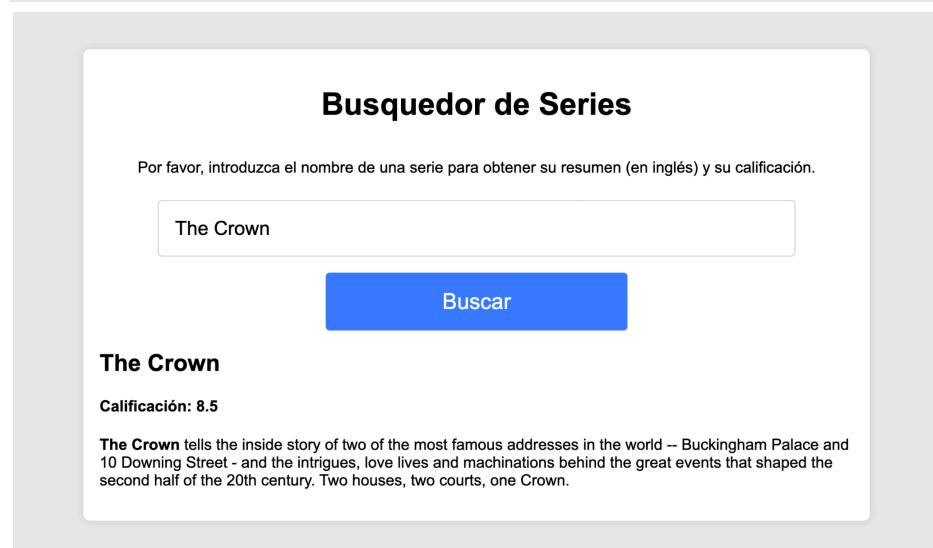
    // Crea el elemento HTML para mostrar la sinopsis de la serie
    const summary = document.createElement("div")
    summary.innerHTML = show.summary;

    // Añade el elemento HTML de la serie a la página
    const showElem = document.getElementById("show")
    showElem.appendChild(name);
    showElem.appendChild(rating);
    showElem.appendChild(summary);
}

</script>
</body>
```



The screenshot shows a search interface titled "Busqueda de Series". It prompts the user to enter a series name in English to get its summary and rating. A text input field contains "Brekin dan", and a blue "Buscar" button is below it. The search results for "Bradley Walsh & Son: Breaking Dad" are displayed, showing a rating of 4.3 and a brief description about Bradley Walsh and his son traveling from Los Angeles to New Orleans.



The screenshot shows the same search interface. After entering "The Crown" into the search bar and clicking "Buscar", the results for "The Crown" are shown. The result includes a rating of 8.5 and a detailed description of the series, which tells the story of Buckingham Palace and 10 Downing Street over the second half of the 20th century.

Finalmente, se agregó código CSS a cada ejercicio para darle estilo y personalidad, mejorando su aspecto visual y la experiencia del usuario.