

Desarrollo Web en Entorno Cliente



Comentarios en el código, pruebas y documentación del código.

Desarrollo Web Entorno Cliente

(Caso práctico 1 UD2)

2º DAW

Jesús A. Raigón Muñoz

INDICE

1. Identificación y características de los principales lenguajes relacionados con la programación de clientes web.
2. Comprueba que la estructura de directorios y extensión de los archivos es la adecuada para resolver el problema que se le plantea.
3. Caracterizado y diferenciado de los modelos de ejecución de código en el cliente web y en el servidor.
4. Realización de pruebas unitarias para testeo del código.
5. Creación de comentarios con JSDoc.
6. Resultado de ejecución del código en el navegador.
7. Bibliografía

1. Identificación y características de los principales lenguajes relacionados con la programación de clientes web.

La programación de clientes web implica trabajar con varios lenguajes que se ejecutan en el navegador del usuario permitiendo crear interfaces interactivas y dinámicas.

Los que he utilizado en la tarea son los siguientes:

HTML (HyperText Markup Language):

- Es el lenguaje estándar para crear y estructurar contenido en la web.
- Define la estructura y el contenido de las páginas web que veo en el navegador mediante etiquetas.

CSS (Cascading Style Sheets):

- Son las reglas de estilo CSS que determinan cómo se ven los elementos en mi HTML y controla la disposición de los elementos en la página el diseño, colores, fuentes, etc.
- Se utiliza para diseñar y maquetar páginas web.

JavaScript:

- Un lenguaje de programación de alto nivel que se utiliza para crear interactividad en las páginas web.
- Permite manipular el DOM (Document Object Model), gestionar eventos y realizar tareas asíncronas.

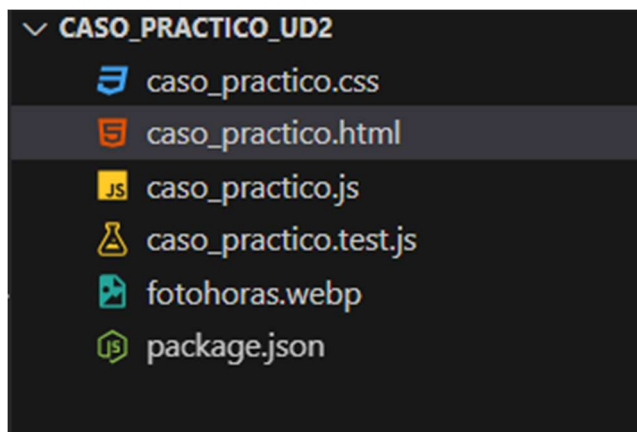
2. Comprueba que la estructura de directorios y extensión de los archivos es la adecuada para resolver el problema que se le plantea.

Para el resolver la tarea en su totalidad he creado 6 tipos de archivos que son los que forman el proyecto.

Considero que la extensión de los archivos es la adecuada porque cada uno tiene una finalidad y que paso a detallar:

1. .html: Este archivo contiene el marcado HTML de mi aplicación. Define la estructura y el contenido de las páginas web que vemos en el navegador.
2. .css: Aquí es donde se guardan las reglas de estilo CSS que determinan cómo se ven los elementos en mi HTML. Controla el diseño, colores, fuentes, etc.
3. .js: Los archivos JavaScript contienen la lógica de la aplicación. Pueden manejar la interactividad, los eventos del usuario y la manipulación del DOM (Document Object Model).
4. .test.js: Este es un archivo de pruebas unitarias escrito en JavaScript. Contiene el código que prueba diferentes funcionalidades de la aplicación para asegurarse de que funcionan correctamente.
5. .json: Los archivos JSON (JavaScript Object Notation) se utilizan para almacenar y transferir datos estructurados de una manera que es fácil de leer para humanos y máquinas. En el contexto de pruebas, pueden contener datos de prueba o configuraciones.
6. .webp: formato archivo que contiene una foto.

Adjunto captura de pantalla con el directorio donde están los archivos anteriormente detallados.



3. Caracterizado y diferenciado de los modelos de ejecución de código en el cliente web y el servidor:

Modelo de Ejecución del Código en el Cliente:

Lenguajes Utilizados: HTML, CSS, y JavaScript.

HTML: Define la estructura y el contenido de la página web.

Características:

- Semántica: Proporciona significado y estructura al contenido mediante etiquetas.
- Accesibilidad: Facilita la creación de contenido accesible para todos los usuarios, incluidos aquellos con discapacidades.

Aplicación en el proyecto:

En el archivo caso_practico.html, se define la estructura básica de la página web. Se incluyen elementos como encabezados (<h1>), párrafos (<p>), imágenes (), y botones (<button>).

CSS: Aplica estilos y diseño a la página web. En el archivo caso_practico.css, he aplicado estilos específicos a elementos como el cuerpo de la página, el contenedor principal, los encabezados, párrafos, y la imagen.

Características:

- Separación de Contenido y Estilo: Mantiene la estructura del contenido y la presentación del estilo en archivos separados.
- Responsive Design: Facilita la creación de diseños que se adaptan a diferentes tamaños de pantalla y dispositivos.

Aplicación en el proyecto:

En el archivo caso_practico.css, se definen los estilos para la página web. Se aplican estilos al cuerpo (body), contenedor principal (#contenido), encabezados (h1), párrafos (p), y la imagen (.borde-imagen)

JavaScript:

JavaScript es un lenguaje de programación de alto nivel que se utiliza para crear interactividad en las páginas web.

Permite modificar el contenido y la estructura de un documento HTML de forma dinámica añadiendo interactividad y comportamiento dinámico.

Características:

- Interactividad: Facilita la creación de páginas web interactivas mediante la manipulación del DOM (Document Object Model).

-Tareas Asíncronas: Permite realizar solicitudes HTTP asíncronas, gestionar eventos, y realizar cálculos sin bloquear la ejecución de otras tareas.

Aplicación en el proyecto:

En mi archivo caso_practico.js, la función mostrarPrompt() muestra un cuadro de diálogo (prompt) solicitando una hora del día y luego muestra un saludo adecuado en inglés y francés basado en la hora introducida.

Se incluye documentación con JSDoc y pruebas unitarias para verificar la funcionalidad del código.

Modelo de Ejecución del Código en el Servidor:

El archivo package.json es muy importante en el mundo Node.js. Este archivo se utiliza para gestionar dependencias, scripts, y metadatos del proyecto. Aquí, describe un caso práctico con información sobre dependencias para Jest, un marco de pruebas unitarias.

1. Lectura del Archivo package.json por el Servidor

-Propósito:

Este archivo no se ejecuta como un script de aplicación en sí, sino que es leído por npm (Node Package Manager) para configurar y gestionar el entorno del proyecto.

-Proceso:

Cuando ejecutas comandos como npm install o npm run test, npm lee este archivo para instalar dependencias (como Jest) y ejecutar scripts definidos (como pruebas unitarias).

2. Scripts en el Servidor

-Gestión de Dependencias:

- El bloque "devDependencies" indica a npm qué paquetes deben instalarse para el desarrollo. En este caso, Jest se instala para realizar pruebas.
- **Ejemplo:** Ejecutar npm install instalará Jest en el entorno de desarrollo, asegurándose de que todas las dependencias necesarias estén disponibles.

-Ejecución de Pruebas:

- El bloque "scripts" define scripts que pueden ser ejecutados por npm. Aquí, el script test está configurado para ejecutar Jest.
- **Ejemplo:** Ejecutar npm run test ejecutará Jest para correr todas las pruebas unitarias definidas en el proyecto.

Diferencias con la Ejecución del Código en el Cliente

1. **En el Cliente:**
 - **Lenguajes:** HTML, CSS, JavaScript.
 - **Ejecución:** Directamente en el navegador del usuario para renderizar la página, aplicar estilos y agregar interactividad.
2. **En el Servidor:**
 - **Lenguajes:** Node.js, PHP, Python, etc.
 - **Ejecución:** Procesos de backend como gestión de datos, ejecución de lógica de negocio, y configuración del entorno de desarrollo a través de npm.

Resumen

- **Cliente:** La ejecución de código en el cliente implica la interpretación y ejecución de HTML, CSS y JavaScript directamente en el navegador del usuario.
- **Servidor:** En el caso del archivo package.json, el servidor (a través de npm) lee este archivo para configurar el entorno de desarrollo, manejar dependencias, y ejecutar scripts de pruebas.

4.Realización de pruebas unitarias para testeo del código.

Las pruebas unitarias verifican el funcionamiento de unidades individuales de código, como funciones o métodos.

En primer lugar, adjunto captura de pantalla de los códigos incluidos en los diferentes archivos creados que permitirán realizar las pruebas unitarias.

.html

```
caso_practico.html > html > head
1  <!DOCTYPE html>
2  <html lang="es">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Caso práctico ud2</title>
8      <!-- Enlace al documento externo de CSS para aplicar estilos -->
9      <link rel="stylesheet" href="caso_practico.css">
10 </head>
11
12 <body>
13     <header>
14         <h1>Aprende a saludar según las hora del día</h1>
15     </header>
16     <!-- Contenedor principal para organizar el contenido -->
17     <div id="contenido">
18         <!-- Párrafos que informar acerca de la importancia de saber comunicarse en diferentes idiomas -->
19         <p>Hoy en día es muy importante saber comunicarse en diferentes idiomas para poder viajar y relacionarnos con
20             personas de otras culturas.</p><br>
21
22         <p>Así que te proponemos de forma fácil aprender a saludar en <b>inglés y francés</b> según la hora del día que
23             sea.
24         </p>
25         <!-- Imagen con borde aplicado a través de CSS -->
26         <br> <br>
27         <!-- Botón que llama a la función JavaScript mostrarPrompt() al hacer clic -->
28         <button onclick="mostrarPrompt()"> Haz clic para aprender a saludar <b>en inglés y francés</b></button>
29     </div>
30
31     <!-- Enlace al documento externo de JavaScript para interactividad -->
32     <script src="caso_practico.js"></script>
33 </body>
34
35 </html>
```


.CSS

```

1  /*En el body se aplican estilos que que implican a todos los elementos que están dentro del mismo*/
2  body {
3      margin: 0px;
4      padding: 0px;
5      color: #150c00;
6      background-color: rgb(210, 229, 246);
7  }
8  /*Estilos para el contenedor principal*/
9  div {
10     padding: 20px;
11     margin: auto;
12     background: #cccccc;
13     border: 5px solid #000000;
14     width: 50%;
15 }
16 /*Estilo para el encabezado h1*/
17 h1 {
18     font-size: 25px;
19     text-align: center;
20 }
21 /*Estilo para los párrafos*/
22 p {
23     font-family: Arial, Helvetica, sans-serif;
24 }
25 /*Estilo para la imagen insertada*/
26 .borde-imagen {
27     border: 4px dotted hsl(0, 91%, 46%);
28     padding: 5px;
29 }
30
```

.js

```

1  /**
2   * Muestra un prompt para que el usuario introduzca una hora del día
3   * y luego muestra un saludo adecuado en inglés y francés basado en la hora introducida. *
4   * @author Jesús Raigón Muñoz
5   * @title Mostrar Prompt que interactúe con el usuario.
6   * @description Esta función solicita al usuario que introduzca una hora del día a través de un cuadro de diálogo
7   * y luego muestra un saludo adecuado en inglés y francés basado en la hora introducida.
8   * @function mostrarPrompt
9   * @returns {void} No retorna un valor, solo muestra alertas basadas en la entrada del usuario.
10  */
11
12  function mostrarPrompt() {
13      // Solicitar al usuario que introduzca una hora (indistinto números o cadena texto) y la convierte en un número entero.
14      let hora = parseInt(prompt("Introduce una hora del día"));
15      //Verifica si la hora introducida es antes de las 12 del mediodía.
16      if (hora < 12) {
17          alert("A esa hora en inglés se saluda Good Morning y en francés Bonjour");
18          //Verifica si la hora es antes de las 18pm
19      } else if (hora < 18) {
20          alert(
21              "A esa hora en inglés se saluda Good Afternoon y en francés Bon Après-midi"
22          );
23          //Verifica si la hora es mayor de 24 y en caso afirmativo muestra el mensaje informando que no es valido.
24      } else if (hora > 24) {
25          alert("Ese horario no es valido");
26          //Para cualquier hora diferente a las anteriores muestra ese saludo.
27      } else {
28          alert("A esa hora en inglés se saluda Good Evening y en francés Bonsoir");
29      }
30  }
31  module.exports = { mostrarPrompt };

```

package.json

```
package.json > ...
1  {
2    "name": "caso-practico-ud2",
3    "version": "1.0.0",
4    "description": "Caso práctico para aprender a saludar según la hora del día.",
5    "main": "caso_practico.js",
6    "scripts": {
7      "test": "jest"
8    },
9    "devDependencies": {
10     "jest": "^29.7.0"
11   }
12 }
13
```

.test.js

```
caso_practico.test.js > describe('mostrarPrompt') callback
1  // Importar la función a probar
2  const { mostrarPrompt } = require('./caso_practico');
3
4  // Simular prompt y alert
5  global.prompt = jest.fn();
6  global.alert = jest.fn();
7
8  describe('mostrarPrompt', () => {
9    beforeEach(() => {
10      // Limpiar los mocks antes de cada prueba
11      prompt.mockClear();
12      alert.mockClear();
13    });
14
15    test('debe mostrar Good Morning y Bonjour si la hora es menor que 12', () => {
16      prompt.mockReturnValueOnce('10');
17      mostrarPrompt();
18      expect(alert).toHaveBeenCalledWith("A esa hora en inglés se saluda Good Morning y en francés Bonjour");
19    });
20
21    test('debe mostrar Good Afternoon y Bon Après-midi si la hora es menor que 18 y mayor o igual a 12', () => {
22      prompt.mockReturnValueOnce('15');
23      mostrarPrompt();
24      expect(alert).toHaveBeenCalledWith("A esa hora en inglés se saluda Good Afternoon y en francés Bon Après-midi");
25    });
26
27    test('debe mostrar que el horario no es válido si la hora es mayor que 24', () => {
28      prompt.mockReturnValueOnce('25');
29      mostrarPrompt();
30      expect(alert).toHaveBeenCalledWith("Ese horario no es valido");
31    });
32
33    test('debe mostrar Good Evening y Bonsoir si la hora es mayor o igual a 18 y menor o igual a 24', () => {
34      prompt.mockReturnValueOnce('20');
35      mostrarPrompt();
36      expect(alert).toHaveBeenCalledWith("A esa hora en inglés se saluda Good Evening y en francés Bonsoir");
37    });
38  });
39
```

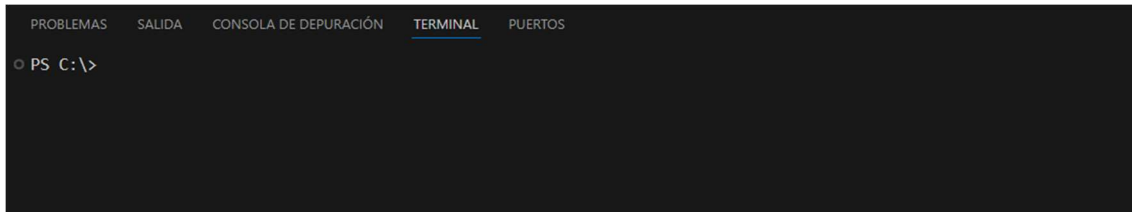
Para hacer las pruebas unitarias de código en JEST sigo estos pasos:

Hay 2 formas de ejecutar las pruebas unitarias:

1. Con Jest Instalado Globalmente

-Abrir la Terminal Integrada en Visual Studio Code:

-Me dirijo a Ver > Terminal o usa el atajo de teclado `Ctrl + `` (comillas invertidas).



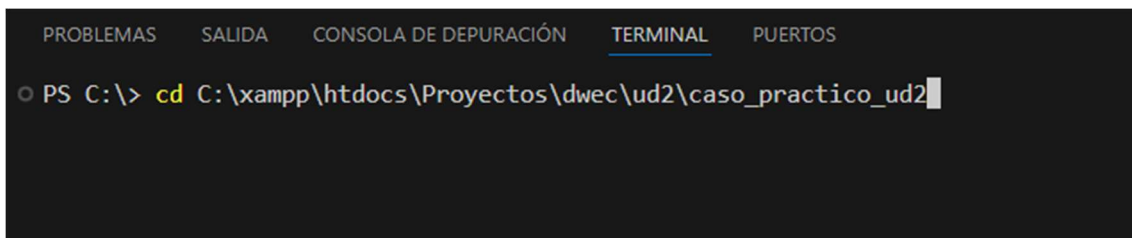
```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
PS C:\>
```

-Navegar al Directorio del Proyecto:

Uso el siguiente comando para cambiar al directorio donde se encuentra mi proyecto:

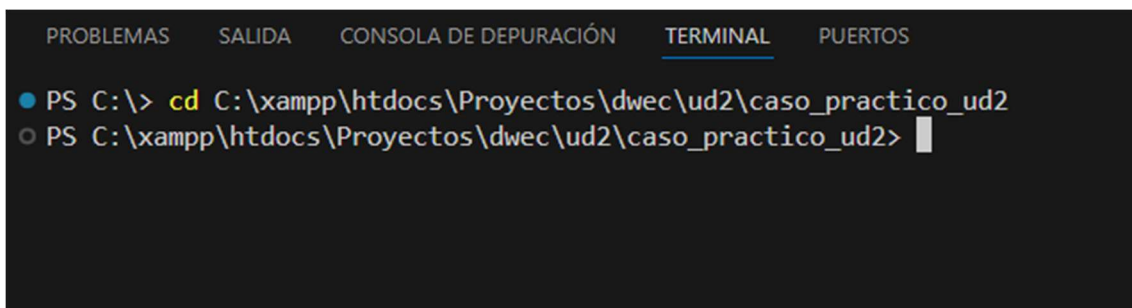
```
cd C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2
```

Tengo el proyecto en la carpeta de htdocs de xampp y esa es la ruta.



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
PS C:\> cd C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2
```

Al meter esa ruta pico enter y sale la segunda línea



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
● PS C:\> cd C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2
PS C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2>
```

Una vez en el directorio correcto, ejecuta el siguiente comando para correr las pruebas:

Escribo `jest` donde el cursor + intro

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

● PS C:\> cd C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2
○ PS C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2> jest
```

Y sale el resultado del test

```
● PS C:\> cd C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2
● PS C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2> jest
PASS ./caso_practico.test.js
  mostrarPrompt
    ✓ debe mostrar Good Morning y Bonjour si la hora es menor que 12 (6 ms)
    ✓ debe mostrar Good Afternoon y Bon Après-midi si la hora es menor que 18 y mayor o igual a 12 (2 ms)
    ✓ debe mostrar que el horario no es válido si la hora es mayor que 24 (1 ms)
    ✓ debe mostrar Good Evening y Bonsoir si la hora es mayor o igual a 18 y menor o igual a 24 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        0.751 s, estimated 1 s
Ran all test suites.
○ PS C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2>
```

2.Directamente en el archivo package.json marcando depurar.

```
caso_practico.html  package.json X
package.json > {} devDependencies
1  {
2    "name": "caso-practico-ud2",
3    "version": "1.0.0",
4    "description": "Caso práctico para aprender a saludar según la hora del día.",
5    "main": "caso_practico.js",
6    "scripts": {
7      "test": "jest"
8    },
9    "devDependencies": {
10     "jest": "^29.7.0"
11   }
12 }
13
```

La salida por consola indica que está todo ok.

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS

> caso-practico-ud2@1.0.0 test
> jest

Debugger attached.
PASS ./caso_practico.test.js
  mostrarPrompt
    ✓ debe mostrar Good Morning y Bonjour si la hora es menor que 12 (8 ms)
    ✓ debe mostrar Good Afternoon y Bon Après-midi si la hora es menor que 18 y mayor o igual a 12 (2 ms)
    ✓ debe mostrar que el horario no es válido si la hora es mayor que 24 (2 ms)
    ✓ debe mostrar Good Evening y Bonsoir si la hora es mayor o igual a 18 y menor o igual a 24 (2 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.355 s
Ran all test suites.
Waiting for the debugger to disconnect...
Waiting for the debugger to disconnect...
PS C:\xampp\htdocs\Proyectos\dwec\ud2\caso_practico_ud2>
```

RESULTADO DEL TEST

Indico lo que significa cada parte del resultado:

1. PASS ./caso_practico.test.js:

Esto indica que el archivo de prueba llamado caso_practico.test.js se ejecutó correctamente y pasó todas las pruebas incluidas.

2. Pruebas específicas:

-mostrarPrompt: Esto es el nombre de un grupo de pruebas o una función que se está probando.

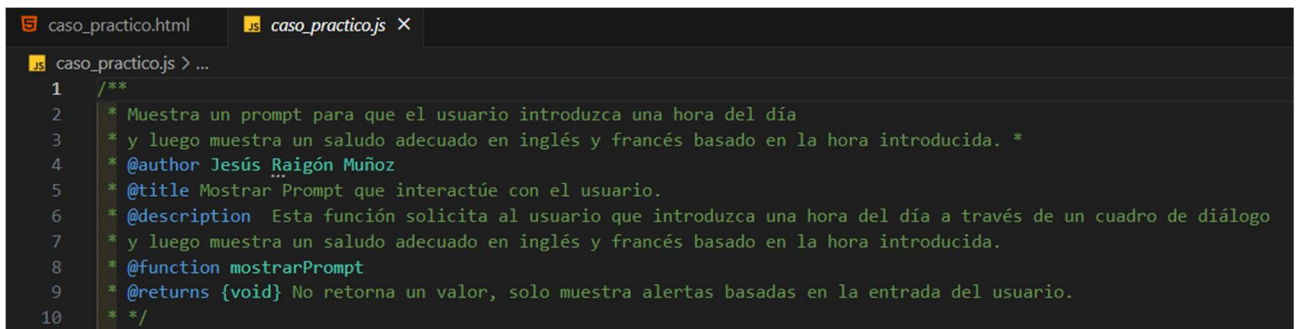
-✓ debe mostrar Good Morning y Bonjour si la hora es menor que 12 (8 ms): Esta prueba específica verifica si la función mostrarPrompt muestra "Good Morning" y "Bonjour" cuando la hora es menor que 12. La prueba pasó (indicado por el "✓") y tomó 8 milisegundos en ejecutarse.

-✓ debe mostrar Good Afternoon y Bon Après-midi si la hora es menor que 18 y mayor o igual a 12 (2 ms): Esta prueba verifica si se muestra "Good Afternoon" y "Bon Après-midi" cuando la hora es menor que 18 y mayor o igual a 12. Pasó y tomó 2 milisegundos.

-✓ debe mostrar que el horario no es válido si la hora es mayor que 24 (1 ms): Esta prueba verifica que se muestra un mensaje de error cuando la hora es mayor que 24. Pasó y tomó 2 milisegundo.

-✓ debe mostrar Good Evening y Bonsoir si la hora es mayor o igual a 18 y menor o igual a 24 (1 ms): Esta prueba verifica si se muestra "Good Evening" y "Bonsoir" cuando la hora es mayor o igual a 18 y menor o igual a 24. Pasó y tomó 2 milisegundos.

5. Creación de comentarios con JSDoc.



```
1  /**
2   * Muestra un prompt para que el usuario introduzca una hora del día
3   * y luego muestra un saludo adecuado en inglés y francés basado en la hora introducida. *
4   * @author Jesús Raigón Muñoz
5   * @title Mostrar Prompt que interactúe con el usuario.
6   * @description Esta función solicita al usuario que introduzca una hora del día a través de un cuadro de diálogo
7   * y luego muestra un saludo adecuado en inglés y francés basado en la hora introducida.
8   * @function mostrarPrompt
9   * @returns {void} No retorna un valor, solo muestra alertas basadas en la entrada del usuario.
10  */
```

JSDoc es muy útil para crear documentación automática de tu código y facilitar la comprensión a otros desarrolladores, especialmente los nuevos que se incorporan al equipo.

Partes del Comentario JSDoc

1. Descripción General:

Propósito: Describe brevemente lo que hace la función, que es mostrar un prompt para que el usuario introduzca una hora y luego mostrar un saludo basado en la hora.

Uso del Comentario Multilínea (/ ... */):** Este tipo de comentario es interpretado por herramientas como JSDoc para generar documentación automática.

2. Etiquetas JSDoc:

@author: se utiliza para indicar el autor del código o de la función que se está documentando. Esta etiqueta ayuda a atribuir correctamente el trabajo a la persona que lo escribió, lo cual puede ser útil para el mantenimiento del código, especialmente en proyectos colaborativos donde varios desarrolladores contribuyen al mismo proyecto.

@title: proporciona un título descriptivo para la función o sección del código. Aunque no es una etiqueta estándar de JSDoc, se utiliza comúnmente para mejorar la claridad y el entendimiento del código.

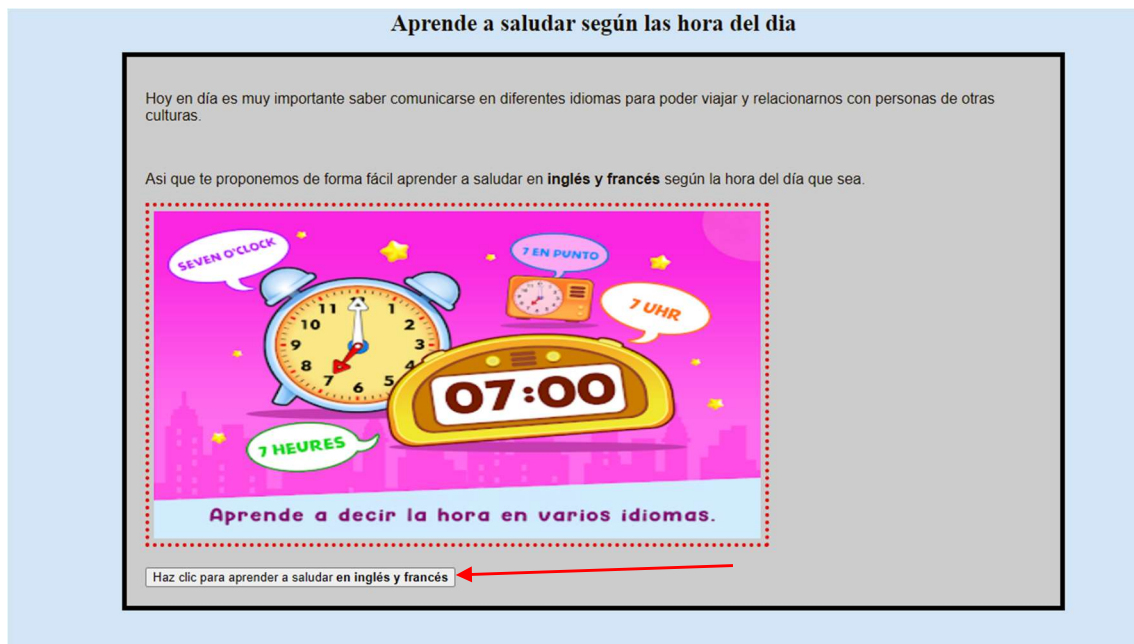
@description: proporciona una descripción detallada de la función, método, o bloque de código. Esta etiqueta ayuda a otros desarrolladores a entender el propósito y funcionamiento de la función.

@function: Indica que este bloque de comentarios está documentando una función.

@returns: Describe lo que retorna la función. En este caso, especifica que la función no retorna ningún valor (void), sino que muestra alertas basadas en la entrada del usuario.

6. Resultado de ejecución del código en el navegador.

Al ejecutar en el navegador el archivo .html comprobamos según imagen que disponemos de un botón en la parte inferior que permite interactuar en el caso de ser pulsado.



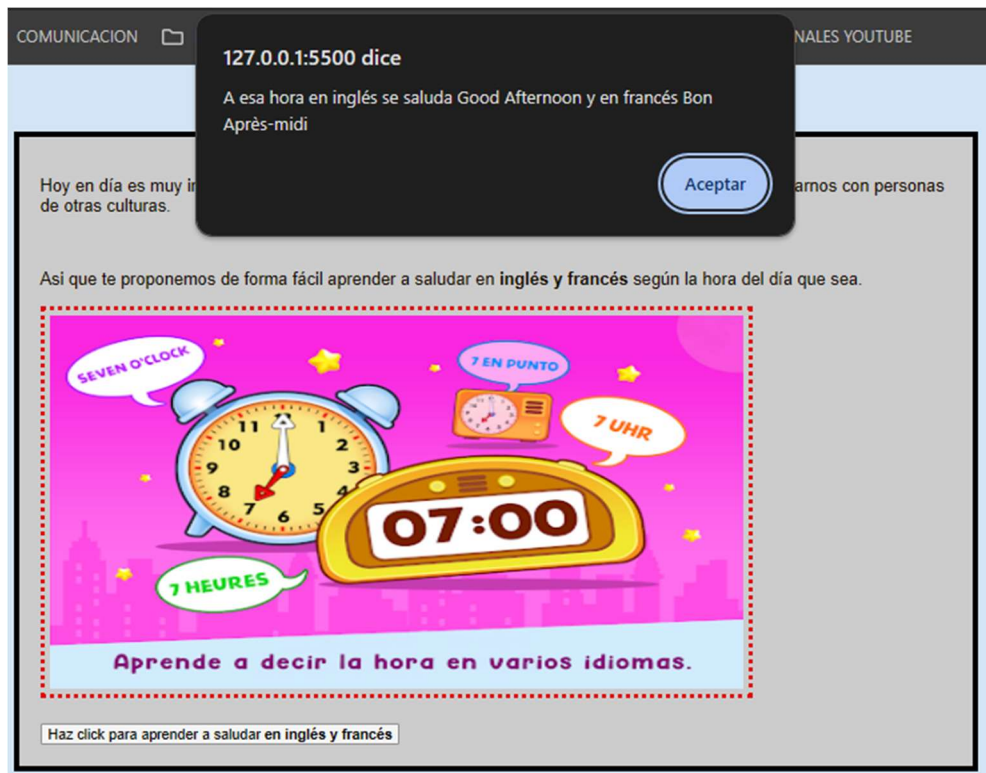
A continuación, sale un prompt o alerta que nos solicita introducir información en la barra para retornar una respuesta según el dato introducido.



Al introducir las 11.00 am sale el mensaje correspondiente...y que debemos pulsar aceptar para que desaparezca.



Al introducir las 14.00 sale el mensaje correspondiente.



Si introducimos un horario no valido como 50 sale este mensaje.



7.Bibliografía

A continuación, adjunto diversos links y libros que he utilizado como referencias bibliográficas para realizar el caso práctico.

[1.2.- Ejecución de código en el servidor y en el cliente. | DWES01.- Plataformas de programación web en entorno servidor. Aplicaciones LAMP.](#)

[978-84-11920-01-8 El desarrollo web desde el entorno del cliente.pdf](#)

"HTML5: Guía Definitiva" por Arkaitz Garro

"Fundamentos de Sistemas Operativos" por académicos de la Universidad Nacional Autónoma de México

"CSS3 y JavaScript Avanzado" por OpenLibra

"Desarrollo de Aplicaciones Web" por OpenLibra

"Guía Completa de CSS3" por OpenLibra

"Web Performance Optimization" por OpenLibra

"36 Pasos Básicos para Desarrollar un Sitio Web" por OpenLibra

Pdf de la unidad 2 de la asignatura Desarrollo Web Entorno cliente (CESUR 24-24)

