UDO1UT01

Arquitecturas y Lenguajes de Programación en Clientes Web

Índice

Resultado de aprendizaje, contenido y criterios de evaluación

Resultado de aprendizaje
RA1. Selecciona las arquitecturas y tecnologías de programación sobre clientes web, identificando y analizando las capacidades y características de cada una.
Contenido
1 - Mecanismos de ejecución de código en un navegador web 2 - Capacidades y limitaciones de ejecución 3 - Lenguajes de programación en entorno cliente 4 - Tecnologías y lenguajes asociados 5 - Integración del código con las etiquetas HTML 6 - Herramientas de programación y prueba sobre clientes web. Librerías y frameworks
Criterios de evaluación
a) Se han caracterizado y diferenciado los modelos de ejecución de código en el servidor y en el cliente web b) Se han identificado las capacidades y mecanismos de ejecución de código de los navegadores web. c) Se han identificado y caracterizado los principales lenguajes relacionados con la programación de clientes web d) Se han reconocido las particularidades de la programación de guiones y sus ventajas y desventajas sobre la programación tradicional e) Se han verificado los mecanismos de integración de los lenguajes de marcas con los lenguajes de programación de clientes web. f) Se han reconocido y evaluado las herramientas de programación y prueba sobre clientes web.

Instrucciones

- Desarrolla sobre este mismo documento lo que se pide en cada apartado.
- Si hay código: crea un cuadro de texto, escribe el nombre del fichero y debajo pega el código.
- Pasa el documento final a pdf.
- Nombra el fichero como TuNombre_Apellido1_Apellido2_ UDO1PO01.pdf
- Súbelo a la tarea de CAMPUS.

Actividades

A partir del documento **UD01DOC01 - Arquitecturas y Lenguajes de Programación en Clientes Web** realiza las siguientes actividades:

Actividad 1 (tiempo estimado 25 minutos)

Criterio de evaluación: a, d
Apartado: 1.

Investiga y compara los modelos de ejecución de código en el servidor (por ejemplo, con PHP o Node.js) y en el cliente (JavaScript en el navegador). Luego, responde:

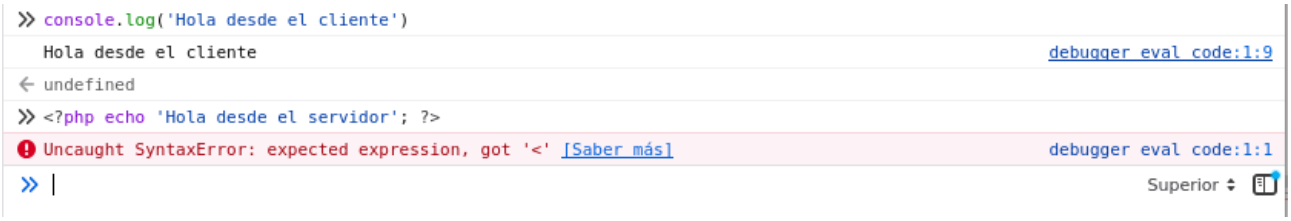
1. ¿En qué se diferencian esencialmente ambos modelos en términos de dónde se ejecuta el código y quién tiene acceso a los resultados?

Cuando la ejecución del código viene desde el lado del cliente es el usuario quien inicia el proceso y proporciona los datos necesarios. Además el cliente suele tener menos potencia de cálculo y procesado, dejando en manos del servidor esas tareas que requieran de mayor capacidad de computo. La petición que se envía desde el cliente llega al servidor donde se harán las validaciones necesarias y se enviara una respuesta al cliente.

En cuanto a la ejecución desde el servidor, tenemos que como se ha mencionado anteriormente ahí se encuentra toda la lógica de negocio, por lo tanto se proporciona, procesan y devuelven las peticiones que se hacen en el cliente.

2. Desde la consola del navegador (F12 → Console), ejecuta el siguiente código JavaScript: console.log('Hola desde el cliente'). Luego, intenta imaginar y explicar por qué no sería posible ejecutar un código PHP como <?php echo 'Hola desde el servidor'; ?> directamente en esa misma consola.

La consola del navegador es una sandbox para código JavaScript. Es por eso que el código en php nos da un error al intentar ejecutarlo (concretamente un erro de sintaxis) mientras que el otro funciona perfectamente.



3. Reflexión: Basándote en lo anterior, enumera al menos una ventaja y una desventaja clave de la programación en el cliente (JavaScript) frente a la programación tradicional en el servidor.

Las ventajas que nos puede ofrecer programar usando un lenguaje como es JavaScript son varias, como por ejemplo la fácil legibilidad y que no es un lenguaje tipado, pero del mismo modo esto nos trae problemas de seguridad que con otro lenguaje como sería Java o PhP en un entorno de servidor evitaríamos. Por ejemplo, en estos dos últimos lenguajes la ejecución nunca llegaría a pasar si hay un fallo con algún tipo de una variable, mientras que en JavaScript se ejecutaría hasta ese punto.

Actividad 2 (tiempo estimado 30 minutos)

Criterio de evaluación: c, e
Apartado: 3 y 5.

Crea un archivo HTML llamado index.html que demuestre las **tres formas principales de integrar JavaScript** en un documento HTML. (Puedes usar alguno hecho en clase)

El archivo debe contener:

1. **Script en línea (Inline):** Dentro de un evento onclick de un botón, que alerte el mensaje: "Script inline ejecutado".

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="./styles/a2.css">
  </head>
  <body>
    <div class="card">
      <h4>Script en línea</h4>
      <button onclick="alert('Script inline ejecutado')">Ejecutar</button>
    </div>
  </body>
</html>
```

En este punto simplemente hemos añadido un bóton con un alert.

2. **Script interno (Embedded):** Dentro de una etiqueta <script> en el <head>, que defina una función llamada saludar(). Esta función debe mostrar en un párrafo con id="saludo" el texto: "¡Hola desde un script interno!".

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="./styles/a2.css" />

    <script>
      function saludar() {
        document.getElementById("psaludo").style.visibility = "visible";
      }
    </script>
```

```
  </head>
  <body>
    <div class="card">
      <h4>Script interno</h4>
      <button onclick="saludar()" id="saludo">Saludo</button>
      <p id="psaludo" class="saludo">¡Hola desde un script interno!</p>
    </div>
  </body>
</html>
```

En cuanto a este código simplemente en vez de poner el script en el mismo botón, hemos utilizado una etiqueta script para mover ahí el código. Simplemente utilizamos el css (en el cual tenemos puesto que el estilo para la clase saludo es visibility: 'hidden'), para que cuando ejecutemos el script con el botón se cambie el estilo a visibility: 'visible'.

3. **Script externo (External):** Incluye un archivo externo llamado script-externo.js usando la etiqueta <script src="..."> al final del <body>. Este archivo debe contener una función que cambie el color de fondo de la página a lightcyan.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <link rel="stylesheet" href="./styles/a2.css">
  </head>
  <body id="bodybg" class="bodybg">
    <div class="card" id="card">
      <h4>Script en línea</h4>
      <button onclick="changeBGColor()">Cambiar color</button>
    </div>
    <script src="./scripts/script-externo.js"></script>
  </body>
</html>
```

En cuanto este código,simplemente hemos movido el script a otro archivo externo. En el cual unicamente cambiamos los valores de los estilo del body.

```
function changeBGColor() {
  document.getElementById("bodybg").style.backgroundColor = "lightCyan";
}
```

Incluye los elementos HTML (botones, párrafos) necesarios para triggerear cada una de estas ejecuciones. Comenta brevemente en el código HTML cada una de las tres secciones.

Actividad 3 (tiempo estimado 25 minutos)

Criterio de evaluación: e, f
Apartado: 5 y 6.
Sigue estos pasos:

1. Copia y pega el siguiente código HTML y JavaScript en un nuevo archivo. Contiene un error intencionado.
2. Abre el archivo en tu navegador y observa el comportamiento incorrecto.
3. Usa las Herramientas de Desarrollo (F12) para encontrar y solucionar el error. Sigue el flujo: a) Revisa la pestaña "Consola" para ver si hay errores.

En la consola como tal no vemos errores al ejecutar el código.

b) Si no hay errores evidentes, usa la pestaña "Sources"/"Debugger" para establecer un punto de ruptura (breakpoint) en la función validarFormulario() y ejecuta el código paso a paso, observando los valores de las variables.

Código con error:

```
<!DOCTYPE html>
<html>
```

```
<head>
  <title>Actividad Depuración</title>
</head>
<body>
  <form onsubmit="return validarFormulario()">
    <input type="text" id="email" placeholder="Tu email">
    <button type="submit">Enviar</button>
  </form>
  <p id="mensaje"></p>

  <script>
    function validarFormulario() {
      let email = document.getElementById('email').value; // Obtiene el valor
      let mensaje = document.getElementById('mensaje');

      // Intenta validar si el email contiene '@'
      if (email.indexOf('@') > 0) { // Error lógico: debería ser > -1
        mensaje.textContent = "Email válido. Formulario enviado.";
        return true;
      } else {
        mensaje.textContent = "Por favor, introduce un email válido.";
        return false; // Evita que el formulario se envíe
      }
    }
  </script>
</body>
</html>
```

4. Describe cuál fue el error, cómo lo identificaste usando las herramientas y escribe la corrección necesaria en el código.

Al ser la condición si el carácter @ se encuentra en el texto que enviamos y además que no sea el primer carácter (posición 0) como tal se ejecuta correctamente. Otra cosa que podríamos considerar un error es que cuando se hace el submit del formulario, es decir la función los devuelve true se reinicia y el usuario pierde la información de que ha ido todo correctamente. Para corregirlo simplemente deberiamos de añadir un un event.preventDefault() en la función del que utiliza el formulario además de pasarle por parámetros el evento.

```
<form onsubmit="validarFormulario(event)">
  <input type="text" id="email" placeholder="Tu email" />
  <button type="submit">Enviar</button>
</form>
<p id="mensaje"></p>

<script>
```

```
  function validarFormulario(event) {
    event.preventDefault();

    let email = document.getElementById("email").value;
    let mensaje = document.getElementById("mensaje");

    if (email.indexOf("@") > 0) {
      mensaje.textContent = "Email válido. Formulario enviado.";
      return true; // Se envia el formulario
    } else {
      mensaje.textContent = "Por favor, introduce un email válido.";
      return false; // Evita que el formulario se envíe
    }
  }
</script>
```

Actividad 4 (tiempo estimado 10 minutos)

Criterio de evaluación: b
Apartado: 2.

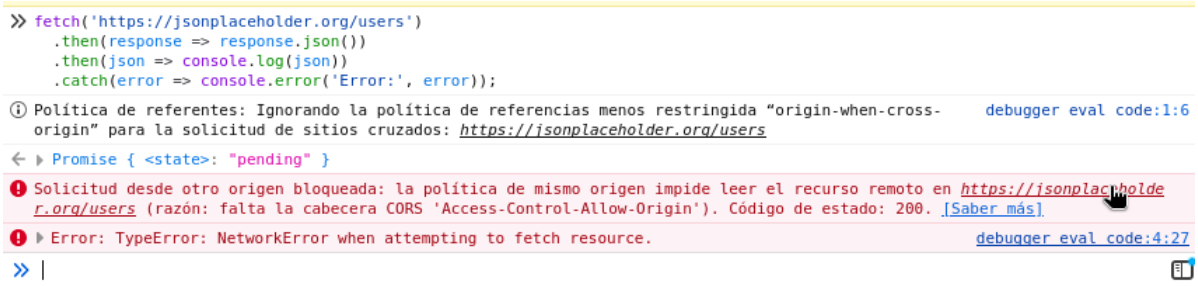
Los navegadores web ejecutan código JavaScript en un entorno seguro y "enjaulado" conocido como *sandbox*. Este mecanismo es fundamental para las capacidades y limitaciones de ejecución. Investiga y realiza lo siguiente:

1.- Experimento Práctico - La Política del Mismo Origen (Same-Origin Policy):

a) Abre las herramientas de desarrollo de tu navegador (F12) y ve a la pestaña "Consola".
b) Visita un sitio web popular como https://www.wikipedia.org .
c) Intenta ejecutar el siguiente código en la consola para intentar leer información de otro origen (dominio):

```
fetch('https://jsonplaceholder.org/users')
  .then(response => response.json())
  .then(json => console.log(json))
  .catch(error => console.error('Error:', error));
```

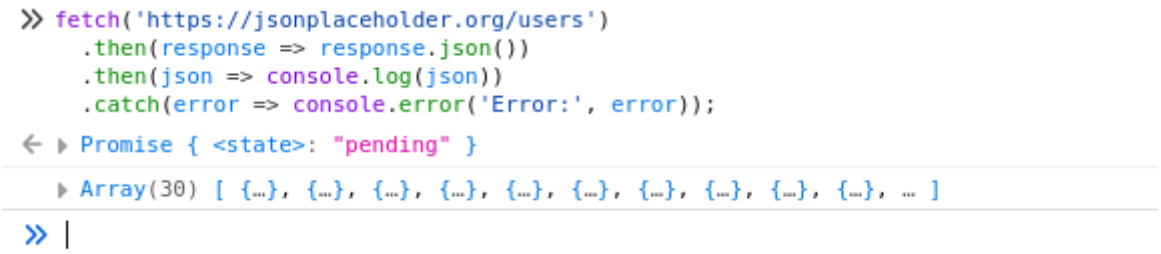
d) **Observa y analiza:** ¿Qué sucede? Si obtienes un error, cópialo y explícalo brevemente. ¿Qué mecanismo de seguridad del navegador está impidiendo o permitiendo esta acción?



Al ejecutar el código proporcionado en la consola mientras estamos en la página de wikipedia nos esta dando un error de CORS, ya que estamos intentando acceder a un recurso que es de un dominio distinto y por lo tanto nos bloquean la petición de manera controlada con un código de estado 200.

e) **Contrasta:** Ahora visita la página de prueba de JSONPlaceholder https://jsonplaceholder.org y ejecuta el mismo código en su consola. ¿Ocurre lo mismo? ¿Por qué?

En este caso la petición se ejecuta correctamente ya que estamos atacando el mismo dominio y por lo tanto no se nos bloquea por CORS.



Rúbrica

Criterio	Excelente (10)	Bien (7)	Suficiente (5)	Insuficiente (0)
a) Se han caracterizado y diferenciado los modelos de ejecución de código en el	Explicación clara y detallada de las diferencias de ejecución cliente/servidor,	Explicación correcta, pero con ejemplos poco desarrollados o	Explicación básica, con confusión parcial entre cliente y	No identifica diferencias o la explicación es

Criterio	Excelente (10)	Bien (7)	Suficiente (5)	Insuficiente (0)
servidor y en el cliente web	con ejemplos precisos y correctos. **(1,5 puntos)**	faltan detalles menores. **(1,05 puntos)**	servidor. **(0,75 puntos)**	incorrecta. **(0 puntos)**
b) Se han identificado las capacidades y mecanismos de ejecución de código de los navegadores web	Respuesta completa sobre sandbox, políticas de seguridad y limitaciones, incluyendo resultados correctos de experimentos prácticos. **(1,5 puntos)**	Respuesta correcta, con omisión de algún detalle menor o ejemplo incompleto. **(1,05 puntos)**	Identifica capacidades básicas, pero con errores conceptuales sobre el sandbox o Same-Origin Policy. **(0,75 puntos)**	No identifica capacidades ni limitaciones, respuestas incorrectas. **(0 puntos)**
c) Se han identificado y caracterizado los principales lenguajes relacionados con la programación de clientes web	Mención completa de HTML, CSS, JavaScript, con sus características, ventajas y desventajas, y ejemplos claros de integración. **(1 puntos)**	Mención correcta de los lenguajes, pero falta detalle en características o ejemplos. **(0,7 puntos)**	Identifica al menos un lenguaje correctamente, pero sin explicar características. **(0,5 puntos)**	No identifica correctamente los lenguajes o sus características. **(0 puntos)**
d) Se han reconocido las particularidades de la programación de guiones y sus ventajas y desventajas sobre la programación tradicional	Reflexión clara, identificando ventajas y desventajas de JS frente a la programación tradicional en servidor, con ejemplos. **(1 puntos)**	Reflexión correcta, pero con pocos ejemplos o argumentos poco desarrollados. **(0,7 puntos)**	Reflexión superficial, ventajas/desventajas poco claras **(0,5 puntos)**.	No identifica particularidades ni ventajas/desventajas. **(0 puntos)**
e) Se han verificado los mecanismos de integración de los lenguajes de marcas con los lenguajes de programación de clientes web	Ejemplo HTML con integración inline, interna y externa completamente funcional, correctamente comentado y explicado. **(1,5 puntos)**	Ejemplo funcional, pero algún comentario o explicación incompleta. **(1,05 puntos)**	Ejemplo parcial, alguna integración no funciona o explicación insuficiente. **(0,75 puntos)**	No integra correctamente los scripts, ni comentarios ni explicaciones. **(0 puntos)**
f) Se han reconocido y evaluado las herramientas de programación y prueba sobre clientes web	Identificación precisa de herramientas (consola, debugger, breakpoints) y uso correcto para depuración del error del formulario. **(1 puntos)**	Identificación correcta, pero uso parcial de herramientas o explicación incompleta. **(0,7 puntos)**	Reconoce algunas herramientas, pero sin aplicar correctamente a la depuración. **(0,5 puntos)**	No identifica ni utiliza herramientas correctamente. **(0 puntos)**
Ejecución práctica y depuración	Código corregido y funcionando perfectamente, con explicación clara de error, cómo se identificó y cómo se solucionó. **(1,05 puntos)**	Código funcional, explicación correcta pero incompleta o poco detallada sobre la identificación del error. **(1,05 puntos)**	Código parcialmente funcional, explicación confusa o incompleta del error. **(0,75 puntos)**	Código no funcional, sin explicación válida del error. **(0 puntos)**
Experimentación Same-Origin y fetch	Explicación completa del error o éxito de la petición, relacionando correctamente con políticas de seguridad y sandbox. **(1 puntos)**	Explicación correcta, pero detalle parcial sobre políticas de seguridad. **(0,7 puntos)**	Explicación básica o con errores conceptuales sobre sandbox/SOP. **(0,5 puntos)**	No comprende el resultado ni las políticas de seguridad. **(0 puntos)**

	Se retrasa más de 5 días	Se retrasa entre 4 y 5 días	Se retrasa entre 2 y 3 días	Se retrasa 1 día.	Entrega en plazo
Entrega tarea en plazo	- 5	- 3	-2	-1	0