DESARROLLO DE APLICACIONES EN RED

Tecnologías JavaScript y AJAX

Objetivos

Poner en práctica los conceptos básicos relacionados con las tecnologías JavaScript y AJAX.

Jose Manuel Pinillos Rubio
pinillosrubio@gmail.com

Asignatura	Datos del alumno	Fecha
Ingeniería de Requisitos	Apellidos: Pinillos Rubio	21 de noviembre de 2022
	Nombre: Jose Manuel	

ÍNDICE

ÍNDICE *2*

INTRODUCCIÓN *3*

JavaScript *4*

Detectar si la cadena de entrada en un palíndromo **4**

Comparar dos números y detectar cual es el mayor **5**

Detectar las vocales que aparecen en un texto **6**

Contabilizar las vocales que aparecen en un texto **8**

Función sinSignos() **11**

AJAX *12*

Detectar la URL de la página cargada en un cuadro de texto **12**

Descargar mediante peticiones AJAX el contenido correspondiente de la URL **13**

Contenidos del archivo 13

Estados de la petición 16

Contenido de las cabeceras 17

Código y texto de respuesta del servidor 17

CONCLUSIONES *19*

Asignatura	Datos del alumno	Fecha
Ingeniería de Requisitos	Apellidos: Pinillos Rubio	21 de noviembre de 2022
	Nombre: Jose Manuel	

INTRODUCCIÓN

En el ámbito de la Ingeniería Informática, el desarrollo de aplicaciones en red desempeña un papel fundamental, exigiendo la integración de tecnologías modernas para garantizar un rendimiento eficiente y una experiencia del usuario óptima. En este contexto, el presente trabajo se centra en la aplicación práctica de los conocimientos adquiridos en las tecnologías JavaScript y AJAX, dos pilares esenciales en el desarrollo web contemporáneo.

En la primera parte, me centro en la programación en JavaScript, abordando una serie de desafíos diseñados para poner a prueba nuestra destreza en la manipulación y gestión de datos. Desde la detección de palíndromos hasta la comparación de números, exploramos las diversas facetas de este lenguaje de programación, destacando su versatilidad y capacidad para resolver problemas variados.

Posteriormente, trabajamos con AJAX (*Asynchronous* JavaScript and XML), una tecnología que revoluciona la interacción entre el usuario y el servidor al permitir solicitudes asíncronas sin necesidad de recargar la página. Mediante una aplicación práctica, hemos abordado escenarios reales, implementando funcionalidades que demuestran la utilidad de AJAX en la obtención de datos de manera dinámica y sin interrupciones perceptibles para el usuario.

La implementación de estos conceptos se ha llevado a cabo en una página web diseñada con HTML y CSS, proporcionando no solo una funcionalidad robusta sino también una interfaz atractiva y amigable para el usuario. A lo largo de esta actividad, detallaré cada ejercicio resuelto, exponiendo el código y los resultados obtenidos, con el objetivo de ofrecer una visión completa de nuestro proceso de desarrollo y aprendizaje.

Para poder interactuar con la WEB creada se puede visitar el siguiente enlace:

- **WEB JavaScript y AJAX**

También se puede visitar el repositorio completo el cual contiene todos los detalles:

- **GitHub/dar_a2**

Asignatura	Datos del alumno	Fecha
Ingeniería de Requisitos	Apellidos: Pinillos Rubio	21 de noviembre de 2022
	Nombre: Jose Manuel	

JavaScript

En este apartado describiré el proceso de desarrollo que he seguido para la resolución de los ejercicios planteados de JavaScript.

Detectar si la cadena de entrada en un palíndromo

El código implementa una función llamada ejer1() que se encarga de verificar si una cadena de texto introducida es un palíndromo o no.

1. **Obtención de la cadena de texto:**

```
let(cadena = document.getElementById("resultado1").value;
```

La función comienza obteniendo el valor de un elemento de la página HTML con el id "ejercicio1". Este elemento es un cuadro de texto (*input*) en el que el usuario ingresa una cadena de texto.

2. **Validación de entrada vacía:**

```
if(cadena == "") {
  document.getElementById("resultado1").innerHTML = `<b style="color:red">Por favor, introduce algo de texto.</b>`;
}
```

Se verifica si la cadena de texto está vacía. Si es así, se actualiza el contenido de otro elemento HTML con el id "resultado1" para mostrar un mensaje en rojo indicando que se debe introducir algo de texto.

3. **Manipulación de la cadena:**

```
cadena = cadena.replace(/\s/g, "").toLowerCase();
```

Si la cadena no está vacía, se realiza un preprocesamiento en la cadena eliminando los espacios en blanco y convirtiendo todos los caracteres a minúsculas. Esto es importante para asegurar que la comparación de palíndromos no sea sensible a mayúsculas/minúsculas y no se vea afectada por los espacios en blanco.

4. **Verificación de palíndromo:**

```
if(cadena === cadena.split("").reverse().join("")) {
    document.getElementById("resultado1").textContent = "Es palindromo.";
  }else{
    document.getElementById("resultado1").textContent = "No es palindromo.";
}
```

Se compara la cadena manipulada con su versión invertida. Si son idénticas, se concluye que la cadena es un palíndromo y se actualiza el contenido del elemento con el id "resultado1" para indicar que es un palíndromo. En caso contrario, se informa que no es un palíndromo. La comparación es estricta `(===)`, lo que significa que la igualdad debe ser tanto en valor como en tipo.

Comparar dos números y detectar cual es el mayor

Este código en JavaScript implementa una función llamada ejer2 que compara dos números ingresados por el usuario y muestra un mensaje indicando cuál de los dos números es mayor o si son iguales.

1. **Obtención de los números del formulario:**

```
let num1 = document.getElementById("num1").value;
let num2 = document.getElementById("num2").value;
```

La función comienza obteniendo los valores de dos campos de texto en un formulario. Estos campos de texto tienen los IDs "num1" y "num2", respectivamente, y se espera que el usuario ingrese números en estos campos.

2. **Validación de entrada vacía:**

```
if(num1 == "" || num2 == "") {
  document.getElementById("resultado2").innerHTML = `<b style="color:red">Por favor, introduzca dos números.</b>`;
}
```

Se verifica si al menos uno de los campos de texto está vacío. Si es así, se actualiza el contenido de otro elemento HTML con el id "resultado2" para mostrar un mensaje en rojo indicando que se deben introducir dos números.

3. **Comparación de números:**

```
if(num1 == num2) {
  document.getElementById("resultado2").textContent = "Los números son
iguales.";
}else{
  if(num1 < num2) {
    document.getElementById("resultado2").innerHTML = `<b>${num2}</b> es el
número mayor.`;
  }else{
    document.getElementById("resultado2").innerHTML = `<b>${num1}</b> es el
número mayor.`;
   }
}
```

Si ambos campos de texto tienen valores, se procede a comparar los números. Primero, se verifica si los números son iguales. Si son iguales, se muestra un mensaje indicando que los números son iguales. Si no son iguales, se compara cuál de los dos números es mayor y se actualiza el contenido del elemento con el id "resultado2" para indicar cuál es el número mayor. El formato HTML utilizado para resaltar el número mayor puede ser útil para hacer que el resultado sea más visible.

Detectar las vocales que aparecen en un texto

Este código en JavaScript implementa una función llamada ejer3 que detecta las vocales que aparecen en una frase ingresada por el usuario y muestra un mensaje listando estas vocales en orden.

1. **Obtención de la cadena de texto:**

```
let cadena = document.getElementById("ejercicio3").value;
```

La función comienza obteniendo el valor de un elemento de la página HTML con el id "ejercicio3". Este elemento es un cuadro de texto (input) en el que el usuario ingresa una cadena de texto.

2. **Eliminación de signos de acentuación de las vocales:**

```
cadena = sinSignos(cadena);
```

Aquí, la función `sinSignos()` se utiliza para eliminar los signos diacríticos (tildes y diéresis) de la cadena. Esto se logra mediante una función de reemplazo que utiliza expresiones regulares para mapear caracteres acentuados a sus equivalentes sin acento.

La función `sinSignos()` se detalla en la ***página 11***.

3. **Eliminación de espacios y conversión a minúsculas de la cadena:**

```
cadena = cadena.replace(/\s/g, "").toLowerCase();
```

La cadena procesada se convierte en minúsculas y se eliminan los espacios en blanco.

4. **Validación de entrada vacía:**

```
if(cadena == "") {
  document.getElementById("resultado3").innerHTML = `<b style="color:red">Por favor, introduce algo de texto.</b>`;
}
```

Se verifica si la cadena de texto está vacía después del procesamiento. Si es así, se actualiza el contenido de otro elemento HTML con el id "resultado3" para mostrar un mensaje en rojo indicando que se debe introducir algo de texto.

5. **Identificación de vocales:**

```
const vocales = "aeiou";
texto = "";
for (const letra of cadena) {
  if (vocales.includes(letra)) {
    texto = texto + letra + " ";
  }
}
```

Se utiliza un bucle `for` para iterar sobre cada letra de la cadena resultante después de la manipulación. Si la letra es una vocal (está incluida en la cadena de vocales), se agrega a una nueva cadena llamada texto.

6. **Actualización del resultado:**

```
document.getElementById("resultado3").innerHTML = `<b>${texto}</b>`;
```

Finalmente, el contenido del elemento con el id "resultado3" se actualiza para mostrar las vocales encontradas. La estructura HTML `<b>${texto}</b>` se utiliza para resaltar el texto resultante en negrita.

Contabilizar las vocales que aparecen en un texto

Este código en JavaScript implementa una función llamada ejer4 que cuenta y muestra la frecuencia de cada vocal en una cadena de texto introducida por el usuario.

1. **Obtención de la cadena de texto:**

```
let cadena = document.getElementById("ejercicio4").value;
```

La función comienza obteniendo el valor de un elemento de la página HTML con el id "ejercicio4". Este elemento es un cuadro de texto (input) en el que el usuario ingresa una cadena de texto.

2. **Eliminación de signos de acentuación de las vocales:**

```
cadena = sinSignos(cadena);
```

Aquí, la función `sinSignos()` se utiliza para eliminar los signos diacríticos (tildes y diéresis) de la cadena. Esto se logra mediante una función de reemplazo que utiliza expresiones regulares para mapear caracteres acentuados a sus equivalentes sin acento.

La función `sinSignos()` se detalla en la ***página 11***.

3. **Eliminación de espacios y conversión a minúsculas de la cadena:**

```
cadena = cadena.replace(/\s/g, "").toLowerCase();
```

La cadena procesada se convierte en minúsculas y se eliminan los espacios en blanco.

4. **Validación de entrada vacía:**

```
if(cadena == "") {
  document.getElementById("resultado4").innerHTML = `<b style="color:red">Por favor, introduce algo de texto.</b>`;
}
```

Se verifica si la cadena de texto está vacía después del procesamiento. Si es así, se actualiza el contenido de otro elemento HTML con el id "resultado4" para mostrar un mensaje en rojo indicando que se debe introducir algo de texto.

5. **Conteo de vocales:**

```
let a=0, e=0, i=0, o=0, u=0;

for (const texto of cadena) {

  if (texto.includes("a")) {
    a++;
  }

  if (texto.includes("e")) {
    e++;
  }

  if (texto.includes("i")) {
    i++;
  }

  if (texto.includes("o")) {
    o++;
  }

  if (texto.includes("u")) {
    u++;
  }
}
```

Se utiliza un bucle `for` para iterar sobre cada letra de la cadena resultante después de la manipulación y se incrementan los contadores para cada vocal presente en la cadena.

6. **Actualización del resultado:**

```
if (a == 0 && e == 0 && i == 0 && o == 0 && u == 0) {
  document.getElementById("resultado4").textContent = "El texto no contiene vocales.";
}else {
  document.getElementById("resultado4").innerHTML = `La <b>A</b> aparece ${a} veces.<br/>
  La <b>E</b> aparece ${e} veces.<br/>
  La <b>I</b> aparece ${i} veces.<br/>
  La <b>O</b> aparece ${o} veces.<br/>
  La <b>U</b> aparece ${u} veces.`;
}
```

Finalmente, el contenido del elemento con el id "resultado4" se actualiza para mostrar la frecuencia de cada vocal en la cadena, y si no se encontraron vocales, se muestra un mensaje indicando que el texto no contiene vocales.

Función sinSignos()

La función `sinSignos()` se define utilizando una expresión de función inmediatamente invocada (IIFE, por sus siglas en inglés) para crear un ámbito cerrado. Esto ayuda a encapsular las variables locales y evitar contaminar el ámbito global.

1. **Mapa de caracteres acentuados y sin signos diacríticos:**

```
let de = 'ÁÃÀÄÂÉËÈÊÍÏÌÎÓÖÒÔÚÜÙÛÑáãàäâéëèêíïìîóöòôúüùû',
a = 'AAAAAEEEEIIIIOOOOUUUUNaaaaaeeeeiiiioooouuuu';
```

Se definen dos cadenas, de que contiene caracteres acentuados, y a que contiene los equivalentes sin acento correspondientes.

2. **Expresión regular para buscar caracteres acentuados:**

```
re = new RegExp('['+de+']' , 'ug');
```

Se crea una expresión regular que busca cualquier caracter presente en la cadena de. La bandera 'u' se utiliza para manejar correctamente los caracteres Unicode, y 'g' indica que la búsqueda debe ser global.

3. **Función de remplazo:**

```
return texto =>
  texto.replace(
  re,
  match => a.charAt(de.indexOf(match))
);
```

La función devuelve otra función que toma un texto como parámetro y utiliza el método `replace` para reemplazar cada caracter acentuado con su equivalente sin acento.

4. **Invocación inmediata (IIFE):**

```
})();
```

La IIFE asegura que la función `sinSignos()` se ejecute inmediatamente después de ser definida, devolviendo la función interna que realiza el reemplazo de caracteres acentuados por sus equivalentes sin acento.

AJAX

En este apartado describiré el proceso de desarrollo que he seguido para la resolución de los ejercicios planteados de AJAX.

Este código en JavaScript se centra en la manipulación de peticiones HTTP asíncronas (AJAX) para cargar contenido de una URL y mostrar información sobre la solicitud y su respuesta.

Detectar la URL de la página cargada en un cuadro de texto

1. **Inicialización de variables:**

```
let statesList = ['NO INICIALIZADA', 'ABIERTA', 'CABECERAS RECIBIDAS', 'CARGANDO',
'COMPLETADA'];
let initialTime = 0;
```

Se inicializan dos variables: `statesList`, que contiene descripciones de los estados posibles de una solicitud AJAX, e `initialTime`, que se utilizará para medir el tiempo que tarda en completarse la solicitud.

2. **Evento window.onload:**

```
window.onload = function() {
  let source = document.getElementById('code');
  source.value = location.href;

  document.getElementById('enviar').onclick = loadContent;
  document.getElementById('web').onclick = showWEB;
  document.getElementById('html').onclick = showHTML;
}
```

La función `window.onload` es un evento que se dispara cuando el contenido de la página web ha sido completamente cargado, incluyendo todos los recursos como imágenes, scripts, hojas de estilo, etc. Cuando este evento ocurre, se ejecuta la función asociada a `window.onload`.

Utilizo `window.onload` para asegurarme de que ciertas operaciones se realicen solo después de que la página web haya cargado completamente.

Asignatura	Datos del alumno	Fecha
Ingeniería de Requisitos	Apellidos: Pinillos Rubio	21 de noviembre de 2022
	Nombre: Jose Manuel	

Descargar mediante peticiones AJAX el contenido correspondiente de la URL

1. **Obtención de la URL y asignación al campo de texto:**

```
let source = document.getElementById('code');
  source.value = location.href;
```

Se toma el valor actual de la URL y se establece en el campo de texto con el id “code".

Contenidos del archivo

1. **Asignación de funciones a eventos de clic:**

```
document.getElementById('enviar').onclick = loadContent;
document.getElementById('web').onclick = showWEB;
document.getElementById('html').onclick = showHTML;
```

Se asignan funciones (`loadContent`, `showWEB`, y `showHTML`) a los eventos de clic de los elementos con los ID 'enviar', 'web', y 'html', respectivamente. Esto significa que cuando se hace clic en estos elementos, se ejecutarán las funciones asociadas.

1.1. Función `showWEB()`:

La función `showWEB()` es una parte del código que se ejecuta cuando se hace clic en el botón con el id 'web'. Su objetivo es mostrar el contenido de una respuesta HTTP recibida como parte de una solicitud AJAX, ocultar un botón y mostrar otro.

1.1.1. Ocultar y mostrar elementos:

```
document.getElementById("web").style.display = "none";
```

Se cambia la propiedad de estilo *display* del elemento con el ID 'web' para ocultarlo (se establece en "*none*").

```
document.getElementById("html").style.display = "inline";
```

Se cambia la propiedad de estilo *display* del elemento con el ID 'html' para mostrarlo (se establece en "*inline*").

1.1.2. Verificar el estado de la solicitud AJAX:

```
if(request.readyState == 4 && request.status == 200) {
  let contents = document.getElementById('contenido');
    contents.innerHTML = request.responseText
}
```

Se verifica si el estado de la solicitud AJAX (`request.readyState`) es igual a 4 (que indica que la solicitud ha sido completada).

Se verifica si el código de estado de la respuesta HTTP (`request.status`) es igual a 200 (que indica que la solicitud fue exitosa).

1.1.3. Mostrar el contenido de la respuesta:

```
let contents = document.getElementById('contenido');
contents.innerHTML = request.responseText
```

Si la solicitud AJAX ha sido completada con éxito, se obtiene el elemento con el ID 'contenido'. El contenido de este elemento se establece como el texto de la respuesta HTTP `request.responseText`.

1.2. Función `showHTML()`:

La función `showHTML()` es una parte del código que se ejecuta cuando se hace clic en el botón con el id 'web'. Su objetivo es mostrar el contenido de una respuesta HTTP en formato de texto sin formato, ocultar un botón y mostrar otro.

1.2.1. Ocultar y mostrar elementos:

```
document.getElementById("html").style.display = "none";
```

Se cambia la propiedad de estilo *display* del elemento con el ID 'html' para ocultarlo (se establece en "*none*").

```
document.getElementById("web").style.display = "inline";
```

Se cambia la propiedad de estilo *display* del elemento con el ID 'web' para mostrarlo (se establece en "*inline*").

1.2.2. Verificar el estado de la solicitud AJAX:

```
if(request.readyState == 4 && request.status == 200) {
  let contents = document.getElementById('contenido');
    contents.innerHTML = request.responseText
}
```

Se verifica si el estado de la solicitud AJAX (`request.readyState`) es igual a 4 (que indica que la solicitud ha sido completada).

Se verifica si el código de estado de la respuesta HTTP (`request.status`) es igual a 200 (que indica que la solicitud fue exitosa).

1.2.3. Mostrar el contenido de la respuesta:

```
let contents = document.getElementById('contenido');
contents.innerHTML = request.responseText
```

Si la solicitud AJAX ha sido completada con éxito, se obtiene el elemento con el ID 'contenido'. El contenido de este elemento se establece como el texto puro de la respuesta HTTP `request.responseText`.

1.3. Función `showContent()`:

La función `showContent()` es la función de retorno de llamada (*callback*) que se asigna al evento `onreadystatechange` de la solicitud `XMLHttpRequest`. Esta función se ejecuta cada vez que el estado de la solicitud cambia.

1.3.1. Cálculo del tiempo de respuesta:

```
let finalTime = new Date();
let miliseconds = finalTime - initialTime;
```

Se toma el valor actual de la URL y se establece en el campo de texto con el id “code".

1.3.2. Mostrar contenido de la respuesta si la solicitud ha sido completada con éxito:

```
if(request.readyState == 4 && request.status == 200) {
  let contents = document.getElementById('contenido');
  contents.textContent = request.responseText;
  document.getElementById("web").style.display = "inline";
  showHeaders();
  showStateCodes();
}
```

Se verifica si la solicitud ha alcanzado el estado 4 (completada) y si el código de estado HTTP es 200 (éxito).

Si ambas condiciones son verdaderas, se obtiene el elemento HTML con el ID 'contenido'.

Se establece el contenido de este elemento como el texto de la respuesta HTTP (`request.responseText`), que es el contenido de la página solicitada.

Se hace visible el botón con el ID 'web'.

Se llaman a las funciones `showHeaders` y `showStateCodes` para mostrar las cabeceras de la respuesta y el código de estado respectivamente.

Estados de la petición

1. **Mostrar estados de la solicitud en un elemento HTML:**

```
let states = document.getElementById('estados');
states.innerHTML += request.readyState +" - [" + miliseconds + " ms.] " +
statesList[request.readyState] + "<br/>";
```

Se obtiene el elemento HTML con el ID 'estados'.

Se agrega información sobre el estado actual de la solicitud, el tiempo transcurrido y una descripción del estado a este elemento. Esta información se concatena al contenido existente usando el operador `+=` para que se muestre en varias líneas.

Contenido de las cabeceras

La función `showHeaders()` es parte del código y tiene como objetivo mostrar las cabeceras de la respuesta HTTP recibida como parte de una solicitud AJAX.

1. **Obtención del elemento HTML para las cabeceras:**

   ```
   let headers = document.getElementById('cabeceras');
   ```

   Se obtiene el elemento HTML con el ID 'cabeceras', donde se mostrarán las cabeceras de la respuesta.

2. **Obtención y asignación de las cabeceras de la respuesta:**

   ```
   headers.innerHTML = request.getAllResponseHeaders();
   ```

   Se utiliza el método `getAllResponseHeaders()` de la solicitud `XMLHttpRequest` (*request*) para obtener todas las cabeceras de la respuesta HTTP.

   El contenido de estas cabeceras se asigna como el contenido HTML del elemento con el ID 'cabeceras'.

Código y texto de respuesta del servidor

La función `showStateCodes()` es parte del código y su propósito es mostrar el código de estado de la respuesta HTTP recibida como parte de una solicitud AJAX.

1. **Obtención del elemento HTML para el código de estado:**

   ```
   let code = document.getElementById('codigo');
   ```

   Se obtiene el elemento HTML con el ID 'codigo', donde se mostrará el código de estado de la respuesta.

Asignatura	Datos del alumno	Fecha
Ingeniería de Requisitos	Apellidos: Pinillos Rubio	21 de noviembre de 2022
	Nombre: Jose Manuel	

2. **Obtención y asignación del código de estado de la respuesta:**

```
code.innerHTML = request.status + " " + request.statusText;
```

Se utiliza la propiedad `status` de la solicitud `XMLHttpRequest` (*request*) para obtener el código de estado de la respuesta HTTP.

Se utiliza la propiedad `statusText` para obtener la descripción textual asociada con el código de estado.

El contenido de estos valores se asigna como el contenido HTML del elemento con el ID 'codigo'.El contenido de estas cabeceras se asigna como el contenido HTML del elemento con el ID 'cabeceras'.

CONCLUSIONES

En esta actividad, he aplicado los conocimientos adquiridos en el desarrollo de aplicaciones en red, centrándome en el uso de tecnologías clave como JavaScript y AJAX. A través de una serie de ejercicios prácticos, he abordado desafíos que abarcan desde la manipulación de cadenas hasta la implementación de solicitudes asíncronas para cargar contenido dinámico.

En el ámbito de JavaScript, he demostrado habilidades para resolver problemas que van desde la detección de palíndromos hasta la gestión de flujos de control para determinar el mayor de dos números. Además, he desarrollado funciones para analizar y contar vocales en frases, brindando soluciones eficientes mediante el uso de bucles y estructuras condicionales.

En el contexto de AJAX, he construido una página web interactiva que permite a los usuarios cargar contenido de una URL específica de manera asíncrona. Esta aplicación no solo presenta la capacidad de realizar solicitudes HTTP, sino que también ofrece una interfaz intuitiva que muestra información crucial sobre el proceso de solicitud, como estados, tiempos de respuesta, cabeceras y códigos de estado.

Las funciones `showHeaders` y `showStateCodes` destacan la capacidad de la aplicación para proporcionar detalles adicionales sobre la respuesta del servidor, mostrando las cabeceras y el código de estado respectivamente. Además, las funciones `showWEB` y `showHTML` ofrecen una experiencia de usuario fluida al alternar entre las vistas de contenido HTML y texto sin formato.

Esta actividad no solo ha sido una oportunidad para aplicar los conceptos teóricos de desarrollo en red, sino también para demostrar habilidades prácticas en la implementación de soluciones interactivas y dinámicas mediante el uso de tecnologías fundamentales en el desarrollo web moderno.