Programación Front End – Unidad 2

Descargable

Introducción

¡Bienvenido al curso Programación Front End! En esta segunda unidad, aprenderás a manipular el DOM con JavaScript, una habilidad clave para desarrolladores web. Conocerás cómo seleccionar y modificar elementos, cambiar sus atributos y estilos, y manejar eventos para agregar interactividad a las páginas web. También explorarás el uso de arreglos y objetos en JavaScript para la gestión de datos, así como la implementación de operaciones CRUD, fundamentales para aplicaciones prácticas.

¡Adelante!

Modificación del DOM

El *Document Object Model* (DOM) es una representación estructurada del documento HTML que permite a los desarrolladores manipular su contenido y estructura mediante JavaScript. La capacidad de modificar el DOM es esencial para crear páginas web dinámicas e interactivas.

¿Qué es el DOM?

El DOM es una interfaz de programación para documentos HTML y XML. Representa la página de manera que los programas pueden cambiar la estructura del documento, su estilo y su contenido.

- **Nodos del DOM:** Cada parte de un documento HTML es un nodo del DOM. Los nodos pueden ser elementos, atributos, texto, etc.
- **Árbol del DOM:** El DOM se organiza en una estructura de árbol donde cada nodo es un "nodo hijo" del nodo que lo contiene, llamado "nodo padre".

Selección de elementos del DOM

Para manipular el DOM, primero necesitamos seleccionar los elementos que queremos cambiar.

`getElementById`: Selecciona un elemento por su ID.

```
let elemento = document.getElementById('miElemento');
```

`getElementsByClassName`: Selecciona todos los elementos con una clase específica.

`getElementByName`: Esta función permite seleccionar elementos por su nombre, devolviendo una lista de todos los elementos con el nombre especificado.

```
let camposUsuario = document.getElementsByName("username");
```

querySelector y **querySelectorAll**: Selecciona el primer elemento que coincide con un selector CSS (querySelector) o todos los elementos que coinciden (querySelectorAll).

```
let elemento = document.querySelector('.miClase');
let elementos = document.querySelectorAll('.miClase');
```

Modificación de elementos del DOM

Una vez que hemos seleccionado un elemento, podemos modificarlo.

1. **Modificar el contenido:**
   - **innerHTML:** La propiedad innerHTML permite obtener o establecer el contenido HTML de un elemento. Esto incluye no solo el texto, sino también cualquier elemento HTML dentro del elemento.
   - **textContent:** La propiedad textContent permite obtener o establecer el texto dentro de un elemento. Esta propiedad no interpreta ni incluye etiquetas HTML, solo el texto plano.

```
elemento.innerHTML = '<h1>Nuevo contenido</h1>';
elemento.textContent = 'Nuevo texto';
```

Comparación

- **Contenido HTML:** innerHTML permite la inclusión y manipulación de etiquetas HTML dentro del contenido del elemento, mientras que textContent solo maneja texto plano.
- **Seguridad:** innerHTML puede ser vulnerable a ataques de Cross-Site Scripting (XSS) si se manipula con datos no confiables, ya que puede interpretar y ejecutar código HTML y JavaScript malicioso. textContent, por otro lado, se considera más seguro porque no interpreta el contenido como HTML.
- **Rendimiento:** textContent suele ser más rápido y eficiente cuando se trabaja únicamente con texto, ya que no necesita analizar y renderizar el contenido HTML.

2. **Modificar atributos**

   Podemos cambiar algún atributo de alguna etiqueta HTML, por ejemplo, cambiar la imagen de una etiqueta <img> modificando el atributo src.

```
elemento.setAttribute('src', 'nuevaImagen.jpg');
let valor = elemento.getAttribute('src');
elemento.removeAttribute('src');
```

3. **Modificar estilos**

```
elemento.style.color = 'red';
elemento.style.backgroundColor = 'blue';
```

Manejo de eventos

Los eventos son acciones que ocurren en el navegador, como clics, movimientos del ratón, y envíos de formularios. Podemos usar JavaScript para escuchar estos eventos y responder a ellos.

Agregar un evento

```
elemento.addEventListener('click', function() {
    alert('Elemento clickeado');
});
```

Eliminar un evento

```
function miFuncion() {
    alert('Evento eliminado');
}
elemento.removeEventListener('click', miFuncion);
```

Tabla Resumen de Eventos Comunes en JavaScript

Evento	Descripción
click	Ocurre cuando un elemento es clickeado.
dblclick	Ocurre cuando un elemento es doble clickeado.
mouseover	Ocurre cuando el puntero del ratón se coloca sobre un elemento.
mouseout	Ocurre cuando el puntero del ratón se aleja de un elemento.
keydown	Ocurre cuando una tecla es presionada.
keyup	Ocurre cuando una tecla es liberada.
change	Ocurre cuando el valor de un elemento de formulario cambia.
submit	Ocurre cuando un formulario es enviado.
focus	Ocurre cuando un elemento gana el foco.
blur	Ocurre cuando un elemento pierde el foco.
input	Ocurre cuando un valor de entrada es modificado.

Ejemplos Prácticos

- **Ejemplo 1:** Cambiar el Contenido de un Elemento

HTML:

```
<div id="miDiv">Contenido original</div>
<button onclick="cambiarContenido()">Cambiar Contenido</button>
```

JavaScript:

```
function cambiarContenido() {
    let div = document.getElementById('miDiv');
    div.innerHTML = 'Contenido cambiado';
}
```

- **Ejemplo 2:** Cambiar Estilos con un Evento

  HTML:

```
<div id="miDiv" style="width: 100px; height: 100px; background-color: yellow;"></div>
<button id="miBoton">Cambiar Color</button>
```

  JavaScript:

```
document.getElementById('miBoton').addEventListener('click', function() {
    document.getElementById('miDiv').style.backgroundColor = 'blue';
});
```

VALIDACION DE FORMULARIOS CON JAVASCRIPT

La validación de formularios es un aspecto crucial en el desarrollo web, ya que garantiza que los datos ingresados por los usuarios sean correctos y adecuados antes de ser procesados o almacenados. La validación del lado del cliente con JavaScript mejora la experiencia del usuario al proporcionar retroalimentación inmediata y reducir la carga en el servidor.

¿Qué es la validación de formularios?

La validación de formularios se refiere al proceso de verificar que los datos ingresados por los usuarios cumplen con ciertos criterios antes de ser enviados al servidor. Esto incluye asegurarse de que los campos obligatorios no estén vacíos, que los datos sean del tipo

correcto (como un correo electrónico o un número), y que cumplan con las restricciones de formato.

Tipos de validación

- **Validación del Lado del Cliente:** Se realiza en el navegador del usuario utilizando JavaScript. Proporciona retroalimentación inmediata y reduce la cantidad de solicitudes al servidor.
- **Validación del Lado del Servidor:** Se realiza en el servidor después de que los datos son enviados. Es esencial para la seguridad, ya que no se puede confiar completamente en la validación del lado del cliente.
- **Validación básica con** JavaScript: Para validar formularios con JavaScript, primero necesitamos seleccionar los elementos del formulario y luego aplicar las reglas de validación.

Ejemplo de Formulario HTML:

```
<form id="miFormulario">
  <label for="nombre">Nombre:</label>
  <input type="text" id="nombre" name="nombre"><br><br>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email"><br><br>
  <label for="edad">Edad:</label>
  <input type="number" id="edad" name="edad"><br><br>
  <button type="submit">Enviar</button>
</form>
<div id="errores"></div>
```

Validación de Campos Vacíos:

```
document.getElementById('miFormulario').addEventListener('submit', function(event) {
  let errores = [];
  let nombre = document.getElementById('nombre').value;
  let email = document.getElementById('email').value;
  let edad = document.getElementById('edad').value;
```

```
 if (nombre === '') {
  errores.push('El campo nombre es obligatorio.');
 }

 if (email === '') {
  errores.push('El campo email es obligatorio.');
 }

 if (edad === '') {
  errores.push('El campo edad es obligatorio.');
 }

 if (errores.length > 0) {
  event.preventDefault();
  document.getElementById('errores').innerHTML = errores.join('<br>');
 }
});
```

Validación de Tipos de Datos

Además de verificar si los campos están vacíos, también es importante validar el tipo de datos ingresados.

Ejemplo: Validación de un Correo Electrónico

```
function esEmailValido(email) {
 let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
 return regex.test(email);
}

document.getElementById('miFormulario').addEventListener('submit', function(event) {
 let errores = [];
 let email = document.getElementById('email').value;
```

```
 if (!esEmailValido(email)) {
  errores.push('El formato del email es incorrecto.');
 }

 if (errores.length > 0) {
  event.preventDefault();
  document.getElementById('errores').innerHTML = errores.join('<br>');
 }
});
```

Validación de Rango Numérico

Para campos numéricos, podemos validar que el valor ingresado esté dentro de un rango específico.

Ejemplo: Validación de Edad entre 18 y 65 Años.

```
function esEdadValida(edad) {
 let min = 18;
 let max = 65;
 return edad >= min && edad <= max;
}

document.getElementById('miFormulario').addEventListener('submit', function(event) {
 let errores = [];
 let edad = document.getElementById('edad').value;

 if (!esEdadValida(edad)) {
  errores.push('La edad debe estar entre 18 y 65 años.');
 }

 if (errores.length > 0) {
  event.preventDefault();
  document.getElementById('errores').innerHTML = errores.join('<br>');
 }
});
```

Prevención del Envío del Formulario

Para evitar que el formulario se envíe si hay errores, se utiliza el método preventDefault() dentro del manejador de eventos del formulario.

Ejemplo Completo de Validación:

```
document.getElementById('miFormulario').addEventListener('submit', function(event) {
  let errores = [];
  let nombre = document.getElementById('nombre').value;
  let email = document.getElementById('email').value;
  let edad = document.getElementById('edad').value;

  if (nombre === '') {
    errores.push('El campo nombre es obligatorio.');
  }

  if (email === '') {
    errores.push('El campo email es obligatorio.');
  } else if (!esEmailValido(email)) {
    errores.push('El formato del email es incorrecto.');
  }

  if (edad === '') {
    errores.push('El campo edad es obligatorio.');
  } else if (!esEdadValida(edad)) {
    errores.push('La edad debe estar entre 18 y 65 años.');
  }

  if (errores.length > 0) {
    event.preventDefault();
    document.getElementById('errores').innerHTML = errores.join('<br>');
  }
});
```

```
function esEmailValido(email) {
  let regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  return regex.test(email);
}

function esEdadValida(edad) {
  let min = 18;
  let max = 65;
  return edad >= min && edad <= max;
}
```

UTILIZACIÓN DE ARREGLOS Y OBJETOS EN JAVASCRIPT

Los arreglos y objetos son estructuras de datos fundamentales en JavaScript. Permiten almacenar y manipular conjuntos de datos de manera eficiente, lo cual es esencial para el desarrollo de aplicaciones web dinámicas. Esta semana, exploraremos cómo utilizar arreglos y objetos en JavaScript para desarrollar aplicaciones.

Arreglos en JavaScript

Los arreglos son colecciones ordenadas de elementos que pueden ser de cualquier tipo de dato. Cada elemento en un arreglo tiene un índice, comenzando desde 0.

Creación y Manipulación de Arreglos

Declaración de Arreglos:

```
let numeros = [1, 2, 3, 4, 5];
let frutas = ["manzana", "naranja", "plátano"];
```

Acceso a Elementos del Arreglo:

```
let primeraFruta = frutas[0]; // "manzana"
```

Métodos Comunes de Arreglos:

Arreglo	Función	Ejemplo
push	Añadir un elemento al final del arreglo.	frutas.push("uva");
pop	Eliminar el último elemento del arreglo.	frutas.pop();
shift	Eliminar el primer elemento del arreglo.	frutas.shift();
unshift	Añadir un elemento al inicio del arreglo.	frutas.unshift("fresa");
map	Crear un nuevo arreglo con los resultados de la llamada a una función para cada elemento.	let numerosDobles = numeros.map(num => num * 2);
filter	Crear un nuevo arreglo con todos los elementos que cumplan una condición.	let numerosPares = numeros.filter(num => num % 2 === 0);
reduce	Aplicar una función a un acumulador y cada valor del arreglo (de izquierda a derecha) para reducirlo a un único valor.	let suma = numeros.reduce((acumulador, num) => acumulador + num, 0);

Objetos en JavaScript

Los objetos son colecciones de propiedades, y cada propiedad es una asociación entre un nombre (clave) y un valor.

Creación y Manipulación de Objetos

Declaración de Objetos:

```
let persona = {
  nombre: "Juan",
```

```
  edad: 30,
  profesion: "Desarrollador"
};
```

Acceso a Propiedades del Objeto:

```
let nombre = persona.nombre; // "Juan"
let edad = persona["edad"]; // 30
```

Modificación de Propiedades:

```
persona.edad = 31;
persona["profesion"] = "Ingeniero de Software";
```

Añadir Nuevas Propiedades:

```
persona.pais = "España";
```

Eliminar Propiedades:

```
delete persona.profesion;
```

Aplicaciones CRUD utilizando Arreglos y Objetos

CRUD se refiere a las operaciones Crear, Leer, Actualizar y Eliminar. Estas operaciones son fundamentales en muchas aplicaciones web.

Ejemplo de Aplicación CRUD

Imaginemos una aplicación de gestión de tareas.

Crear Tarea:

```
let tareas = [];

function crearTarea(nombre, descripcion) {
  let nuevaTarea = {
    id: Date.now(),
    nombre: nombre,
    descripcion: descripcion,
    completada: false
  };
  tareas.push(nuevaTarea);
```

```
}
```

Leer Tareas:

```
function leerTareas() {
  return tareas;
}
```

Actualizar Tarea:

```
function actualizarTarea(id, datosActualizados) {
  let tarea = tareas.find(t => t.id === id);
  if (tarea) {
    Object.assign(tarea, datosActualizados);
  }
}
```

Eliminar Tarea:

```
function eliminarTarea(id) {
  tareas = tareas.filter(t => t.id !== id);
}
```

Ejemplos Prácticos

Ejemplo 1: Manipulación de Arreglos

```
let numeros = [1, 2, 3, 4, 5];
let numerosDobles = numeros.map(num => num * 2);
console.log(numerosDobles); // [2, 4, 6, 8, 10]
```

Ejemplo 2: Manipulación de Objetos

```
let persona = {
  nombre: "Juan",
  edad: 30,
  profesion: "Desarrollador"
};

persona.edad = 31;
persona.pais = "España";
delete persona.profesion;

console.log(persona);
```

Ejemplo 3: Aplicación CRUD

```
let tareas = [];

function crearTarea(nombre, descripcion) {
  let nuevaTarea = {
    id: Date.now(),
    nombre: nombre,
    descripcion: descripcion,
    completada: false
  };
  tareas.push(nuevaTarea);
}

function leerTareas() {
  return tareas;
}

function actualizarTarea(id, datosActualizados) {
  let tarea = tareas.find(t => t.id === id);
  if (tarea) {
    Object.assign(tarea, datosActualizados);
  }
}

function eliminarTarea(id) {
  tareas = tareas.filter(t => t.id !== id);
}

crearTarea("Aprender JavaScript", "Estudiar los conceptos básicos de JavaScript");
crearTarea("Hacer ejercicio", "Salir a correr durante 30 minutos");

console.log(leerTareas());

actualizarTarea(tareas[0].id, { completada: true });

console.log(leerTareas());

eliminarTarea(tareas[1].id);

console.log(leerTareas());
```

En síntesis, Esta semana hemos explorado la manipulación del DOM utilizando JavaScript, una habilidad fundamental para cualquier desarrollador web. La validación de formularios es un aspecto crucial en el desarrollo web, ya que garantiza que los datos ingresados por los usuarios sean correctos y adecuados antes de ser procesados o almacenados. La validación del lado del cliente con JavaScript mejora la experiencia del usuario al proporcionar retroalimentación inmediata y reducir la carga en el servidor.

Por otro lado, la validación de formularios con JavaScript es una técnica esencial para mejorar la calidad y seguridad de los datos ingresados por los usuarios. Proporciona una experiencia de usuario más fluida y reduce la carga en el servidor. A través de ejemplos prácticos, los estudiantes pueden entender cómo implementar validaciones básicas y avanzadas en sus proyectos.

Bibliografía

- **Antani, V. (2023).** *Mastering JavaScript*. Packt Publishing.
- **Flanagan, D. (2011).** *JavaScript: The definitive guide: Activate your web pages (6th ed.)*. O'Reilly Media.
- **JavaScript tutorial. (n.d.). W3schools.com.** Retrieved June 11, 2024, from https://www.w3schools.com/js/