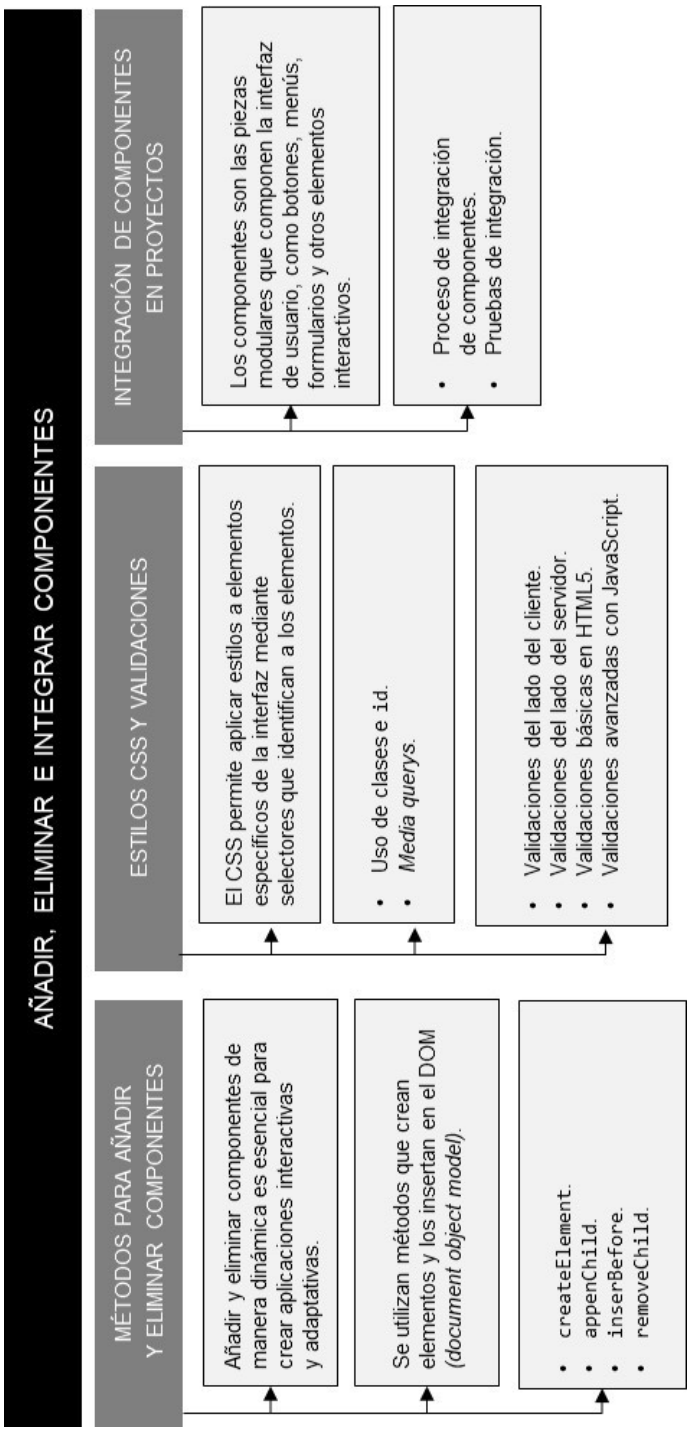
Desarrollo de Interfaces

Tema 2. Añadir, eliminar e integrar componentes

Índice


AÑADIR, ELIMINAR E INTEGRAR COMPONENTES
MÉTODOS PARA AÑADIR Y ELIMINAR COMPONENTES
Añadir y eliminar componentes de manera dinámica es esencial para crear aplicaciones interactivas y adaptativas.
Se utilizan métodos que crean elementos y los insertan en el DOM (document object model).
createElement.
appenChild.
inserBefore.
removeChild.
ESTILOS CSS Y VALIDACIONES
El CSS permite aplicar estilos a elementos específicos de la interfaz mediante selectores que identifican a los elementos.
Uso de clases e id.
Media querys.
Validaciones del lado del cliente.
Validaciones del lado del servidor.
Validaciones básicas en HTML5.
Validaciones avanzadas con JavaScript.
INTEGRACIÓN DE COMPONENTES EN PROYECTOS
Los componentes son las piezas modulares que componen la interfaz de usuario, como botones, menús, formularios y otros elementos interactivos.
Proceso de integración de componentes.
Pruebas de integración.

2.1. Introducción y objetivos

El desarrollo de interfaces de usuario (UI) es un aspecto fundamental en la creación de aplicaciones modernas, donde la usabilidad, la eficiencia y la experiencia del usuario juegan un papel crucial. En este contexto, la capacidad de manipular dinámicamente el **D O M *(document object model)*** para añadir y eliminar componentes se convierte en una herramienta poderosa para los desarrolladores. Esta habilidad permite crear interfaces que no solo son **adaptativas y dinámicas,** sino también más **eficientes** y **alineadas** con las necesidades específicas de los usuarios.

Añadir y **eliminar componentes** del DOM permite que las interfaces **se adapten en tiempo** real a las acciones del usuario, lo que es esencial para aplicaciones que requieren una interacción constante y personalizada. Este enfoque facilita la gestión de recursos, mejora el rendimiento y asegura que la interfaz sea siempre relevante y útil para el usuario.

Por otro lado, los estilos **CSS *(cascading style sheets)*** desempeñan un rol crucial en la **presentación visual** de estas interfaces. CSS permite a los desarrolladores separar la estructura del contenido de su presentación, lo que facilita la creación de interfaces estéticamente agradables y coherentes a través de todo el proyecto. Además, el uso de ***media queries*** en CSS es clave para el diseño adaptativo, asegurando que las aplicaciones se vean y funcionen bien en una amplia gama de dispositivos, desde teléfonos móviles hasta monitores de escritorio.

Las **validaciones,** tanto en el lado del cliente como en el servidor, son, igualmente, importantes para garantizar la integridad de los datos introducidos por los usuarios. Las **validaciones** del lado del **cliente,** implementadas con HTML5 o JavaScript, ofrecen **retroalimentación inmediata** al usuario, evitando errores antes de que los datos sean enviados al servidor. Esto no solo mejora la experiencia del usuario, sino que también reduce la carga en el *backend* al filtrar datos incorrectos o mal formateados.

Finalmente, el **desarrollo basado en componentes** se ha convertido en una metodología esencial en el desarrollo de UI modernas. Esta técnica permite la creación de aplicaciones a partir de **componentes modulares** y **reutilizables,** lo que no solo acelera el proceso de desarrollo, sino que también mejora la mantenibilidad y escalabilidad del *software.* Componentes como botones, formularios o módulos completos de funcionalidad pueden ser integrados de manera eficiente en diferentes partes de una aplicación o en diferentes proyectos, asegurando la coherencia y calidad en todo el sistema.

Los **objetivos** específicos que se pretenden alcanzar con este tema son:

- **Comprender la manipulación del DOM:** adquirir la capacidad de añadir y eliminar dinámicamente componentes en el DOM, mejorando la interactividad y eficiencia de las interfaces de usuario.
- **Dominar el uso de estilos CSS:** aprender a aplicar y gestionar estilos CSS para controlar la presentación visual de las interfaces, asegurando que las aplicaciones sean estéticamente coherentes y adaptables a diferentes dispositivos a través del diseño adaptativo.

- **Implementar validaciones eficientes:** desarrollar habilidades para implementar validaciones en el lado del cliente utilizando HTML5 y JavaScript, asegurando que los datos introducidos por los usuarios sean correctos y en el formato esperado antes de ser procesados.

- **Integrar componentes en proyectos:** entender la metodología de desarrollo basado en componentes y adquirir la habilidad de integrar componentes modulares y reutilizables en diferentes partes de un proyecto, mejorando la eficiencia, escalabilidad y mantenibilidad del *software.*

- **Crear aplicaciones dinámicas y adaptativas:** consolidar conocimientos sobre la creación de interfaces que se adapten en tiempo real a las necesidades del usuario, ofreciendo una experiencia de usuario fluida, intuitiva y atractiva.

2.2. Métodos para añadir componentes

En el desarrollo de interfaces de usuario, la capacidad de **añadir** y **eliminar componentes** de manera dinámica es esencial para crear **aplicaciones interactivas** y **adaptativas.** Estos métodos permiten que la interfaz se adapte a las **acciones** del **usuario** y a las **necesidades** de la **aplicación** en **tiempo real.** Los componentes pueden ser botones, cuadros de texto, menús o cualquier otro elemento que forme parte de la interfaz gráfica.

Añadir y eliminar componentes en una UI es esencial en aplicaciones que requieren una **adaptación constante** a las **entradas del usuario.** Por ejemplo, una aplicación que muestra formularios dinámicos basados en las opciones seleccionadas por el usuario necesita poder añadir nuevos campos de entrada o eliminar aquellos que ya no son necesarios. Esta capacidad hace que la interfaz sea más intuitiva, limpia y fácil de usar, al presentar al usuario solo las opciones relevantes en cada momento.

Otra razón importante para manipular dinámicamente los componentes es la **optimización** de **recursos** y el **rendimiento** de la **aplicación.** Al añadir componentes solo cuando son necesarios y eliminarlos cuando ya no se utilizan, se puede reducir la carga sobre el navegador y mejorar la velocidad y la eficiencia de la aplicación.

Para **añadir** un **componente** a la interfaz, se utilizan métodos que crean elementos y los insertan en el DOM *(document object model).* Un ejemplo común es el método createElement **,** que permite crear un nuevo nodo de tipo elemento, como un párrafo o un botón. Después, este nuevo nodo se puede añadir al DOM mediante el método appendChild **.**

La **eliminación** de **componentes** se realiza utilizando el método `removeChild`, que elimina un nodo específico del DOM. Este método se usa junto con `getElementById` o `querySelector` para seleccionar el nodo que se desea eliminar.

```
// Añadir un nuevo párrafo al final del cuerpo del documento
var nuevoParrafo = document.createElement("p");
nuevoParrafo.textContent = "Este es un nuevo párrafo añadido dinámicamente.";
document.body.appendChild(nuevoParrafo);

// Eliminar un párrafo existente
var parrafoExistente = document.getElementById("parrafo1");
document.body.removeChild(parrafoExistente);
```

Figura 1. Creación y eliminación de un párrafo al principio de un documento. Fuente: elaboración propia.

Métodos para añadir componentes

El proceso de añadir componentes a una interfaz se realiza a través de la **manipulación** del **DOM** utilizando **JavaScript.** Los desarrolladores pueden crear nuevos elementos utilizando el método `createElement`, que genera un nuevo nodo en el DOM. Este nodo puede ser un párrafo, un botón, una lista o cualquier otro tipo de elemento HTML *(hypertext markup language).*

El **primer paso** para añadir un componente es crear el elemento en el DOM. Esto se realiza con `document.createElement(tagName)`, donde `tagName` es el tipo de elemento que se desea crear, como `div`, `p`, `button`, etc.

```
var nuevoElemento = document.createElement("div");
```

Figura 2. Creación del elemento en el DOM. Fuente: elaboración propia.

Después de crear el elemento, se pueden establecer sus **propiedades,** como el texto, el estilo o cualquier otro atributo. Por ejemplo, se puede añadir contenido de texto a un `div` usando `textContent` o establecer un `id` para el nuevo elemento.

```
nuevoElemento.textContent = "Este es un nuevo div añadido dinámicamente.";
nuevoElemento.id = "miNuevoDiv";
```

Figura 3. Establecimiento de propiedades. Fuente: elaboración propia.

Finalmente, el elemento creado y configurado se añade al DOM en la posición deseada utilizando el método `appendChild`, que lo inserta como el último hijo de un nodo especificado.

```
document.body.appendChild(nuevoElemento);
```

Figura 4. Posicionamiento en el DOM. Fuente: elaboración propia.

Además de `appendChild`, existen otros métodos como `insertBefore`, que permite insertar un nuevo nodo antes de un nodo existente, lo que da mayor control sobre la posición del nuevo componente dentro del DOM.

Métodos para eliminar componentes

Eliminar componentes del DOM es, asimismo, importante, especialmente para gestionar recursos y mantener la interfaz ordenada. El método principal para eliminar un nodo del DOM es `removeChild`, que se utiliza en conjunto con la selección del nodo a eliminar.

Primero, es necesario **identificar el nodo que se desea eliminar** utilizando métodos como getElementById o querySelector .

```
var nodoAEliminar = document.getElementById("miNuevoDiv");
```

Figura 5. Selección del nodo a eliminar. Fuente: elaboración propia.

Una vez seleccionado, el nodo se elimina de su nodo padre utilizando removeChild . Es importante recordar que para usar removeChild , se necesita acceder al nodo padre del nodo que se desea eliminar.

```
document.body.removeChild(nodoAEliminar);
```

Figura 6. Eliminación del nodo. Fuente: elaboración propia.

En casos donde se necesite **eliminar** un **grupo de nodos** o **todos los nodos hijos de un elemento,** se pueden utilizar **bucles** o el método innerHTML para vaciar el contenido de un nodo padre.

A continuación, se presenta un ejemplo completo que muestra cómo crear y eliminar dinámicamente componentes en una página web:

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Manipulación Dinámica del DOM</title>
</head>
<body>
    <button onclick="crearElemento()">Añadir Div</button>
    <button onclick="eliminarElemento()">Eliminar Div</button>
    <div id="contenedor"></div>

    <script>
        function crearElemento() {
            var nuevoDiv = document.createElement("div");
            nuevoDiv.textContent = "Este es un nuevo div añadido.";
            nuevoDiv.style.margin = "10px 0";
            nuevoDiv.id = "miDiv";
            document.getElementById("contenedor").appendChild(nuevoDiv);
        }

        function eliminarElemento() {
            var elemento = document.getElementById("miDiv");
            if (elemento) {
                document.getElementById("contenedor").removeChild(elemento);
            }
        }
    </script>
</body>
</html>
```

Figura 7. Creación y eliminación dinámica de componentes en una página web. Fuente: elaboración propia.

En este ejemplo, dos botones permiten al usuario añadir un nuevo `div` al contenedor o eliminarlo. Este tipo de manipulación es común en aplicaciones donde la interfaz necesita adaptarse dinámicamente, como en formularios que añaden campos según la necesidad del usuario o en aplicaciones interactivas que responden a las acciones del usuario en tiempo real.

2.3. Estilos CSS y validaciones en el desarrollo de interfaces gráficas

En el desarrollo de interfaces de usuario (UI), los **estilos CSS** *(cascading style sheets)* y las **validaciones** son componentes fundamentales que, en conjunto, mejoran tanto la **apariencia** como la **funcionalidad** de las aplicaciones web. Mientras que **CSS** se encarga de la **estética** y el **diseño visual,** las **validaciones** aseguran que los **datos** introducidos por los usuarios sean **correctos** y **coherentes,** garantizando una experiencia de usuario más fluida y libre de errores.

CSS es el lenguaje de estilo utilizado para describir la presentación de un documento HTML. A través de CSS, los desarrolladores pueden controlar la **apariencia** de los **elementos** de la interfaz, como el color, el tamaño, la disposición y otros aspectos visuales. La separación entre la estructura del contenido (HTML) y la presentación visual (CSS) es uno de los principios clave del desarrollo web moderno, permitiendo una mayor flexibilidad y mantenibilidad del código.

CSS permite aplicar **estilos** a elementos específicos de la interfaz mediante selectores que identifican a los elementos por su nombre de etiqueta, `id`, clase o, incluso, su estado (como `:hover` o `:focus`). Los **estilos** pueden ser **aplicados** directamente **en el documento HTML,** en el encabezado dentro de una etiqueta `<style>`, o en **archivos CSS externos** que se vinculan al HTML.

A continuación, se muestra un ejemplo donde se define el estilo de un botón, estableciendo su color de fondo, el color del texto, el tamaño del *padding* y otros detalles. Además, se define un cambio en el color de fondo cuando el usuario pasa el cursor sobre el botón, proporcionando una retroalimentación visual interactiva.

```css
/* Estilos para un botón */
button {
    background-color: #4CAF50;
    color: white;
    padding: 10px 20px;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

/* Estilo para el botón cuando el ratón pasa sobre él */
button:hover {
    background-color: #45a049;
}
```

Figura 8. Definición de estilo de un botón mediante CSS. Fuente: elaboración propia.

Uso de clases e id

Mientras que un `id` es **único** para un solo elemento en la página, una **clase** *(class)* se puede aplicar a **múltiples elementos.** Esto permite aplicar estilos comunes a diferentes partes de la interfaz sin duplicar el código.

```css
/* Estilos para todos los elementos con la clase 'destacado' */
.destacado {
    font-weight: bold;
    color: #FF5733;
}

/* Estilos para el contenedor principal con un id específico */
#contenedorPrincipal {
    width: 80%;
    margin: 0 auto;
    padding: 20px;
    border: 1px solid #ddd;
}
```

Figura 9. Definición de estilos usando clases. Fuente: elaboración propia.

En este ejemplo, todos los elementos que llevan la clase «destacado» tendrán el texto en negrita y de color naranja, mientras que el `div` con el `id` contenedorPrincipal tendrá un estilo específico que centraliza su contenido y define sus dimensiones.

Diseño responsive

El diseño *responsive,* una técnica crucial en la era de los dispositivos móviles, se logra mediante CSS y permite que las interfaces se **adapten** a diferentes tamaños de pantalla. Mediante el uso de **consultas de medios** *(media queries),* los desarrolladores pueden definir cómo deben cambiar los estilos dependiendo del ancho de la pantalla.

```css
/* Estilo para pantallas con un ancho máximo de 600px */
@media only screen and (max-width: 600px) {
    #contenedorPrincipal {
        width: 100%;
        padding: 10px;
    }
}
```

Figura 10. Ejemplo de *Media Query.* Fuente: elaboración propia.

Aquí, el contenedor principal ocupará el 100 % del ancho de la pantalla en dispositivos con un ancho menor o igual a 600 px, ajustando también su *padding* para una mejor visualización en pantallas pequeñas.

Tipos de validaciones

Mientras que los estilos CSS se encargan de la presentación, las validaciones son esenciales para asegurar que los **datos** introducidos por los usuarios en los formularios sean **correctos** y sigan el formato esperado. Esto es crucial para prevenir errores y asegurar que los datos procesados por la aplicación sean válidos y útiles.

Existen, principalmente, **dos tipos de validaciones:**

- **Validaciones del lado del cliente:** estas validaciones se realizan directamente en el navegador del usuario antes de que los datos se envíen al servidor. Se implementan utilizando HTML5, JavaScript o combinaciones de ambos. La principal ventaja de estas validaciones es que proporcionan **retroalimentación inmediata** al usuario, evitando enviar datos incorrectos al servidor.

- **Validaciones del lado del servidor:** se realizan después de que los datos han sido enviados al servidor. Estas validaciones son cruciales para la **seguridad,** ya que no se puede confiar completamente en las validaciones del lado del cliente, que pueden ser omitidas o manipuladas.

Validaciones básicas en HTML5

HTML5 proporciona una serie de atributos que facilitan la **validación de datos sin necesidad de JavaScript.** Atributos como `required`, `pattern`, `minlength`, `maxlength`, `type`, y `min/max` permiten validar la entrada del usuario de manera sencilla.

```
<form id="miFormulario">
    <label for="email">Correo Electrónico:</label>
    <input type="email" id="email" name="email" required><br><br>

    <label for="edad">Edad:</label>
    <input type="number" id="edad" name="edad" min="18" max="99" required><br><br>

    <button type="submit">Enviar</button>
</form>
```

Figura 11. Ejemplo de validación en HTML5. Fuente: elaboración propia.

En este formulario, el campo de correo electrónico *(e-mail)* está validado para aceptar solo direcciones de correo válidas, mientras que el campo de edad *(number)* requiere un número entre dieciocho y noventa y nueve.

Validaciones avanzadas con JavaScript

Para **validaciones más complejas,** como verificar que una contraseña cumpla con ciertos criterios o que dos campos coincidan (como confirmar una contraseña), se utiliza JavaScript. Estas validaciones permiten un control más granular y personalizado.

```javascript
document.getElementById("miFormulario").onsubmit = function(event) {
    var email = document.getElementById("email").value;
    var edad = document.getElementById("edad").value;

    if (!email.includes("@")) {
        alert("Por favor, introduce una dirección de correo válida.");
        event.preventDefault(); // Evita el envío del formulario
    }

    if (edad < 18 || edad > 99) {
        alert("Por favor, introduce una edad válida entre 18 y 99 años.");
        event.preventDefault(); // Evita el envío del formulario
    }
};
```

Figura 12. Ejemplo de validación con JavaScript. Fuente: elaboración propia.

Este ejemplo de JavaScript verifica que el correo electrónico introducido contiene una @ y que la edad se encuentra dentro del rango especificado. Si alguna validación falla, se muestra un mensaje de alerta y se evita que el formulario se envíe.

2.4. Integración de componentes en proyectos

La integración de componentes en proyectos de desarrollo de interfaces de usuario es un aspecto crucial que permite a los desarrolladores crear aplicaciones robustas, escalables y mantenibles. Los **componentes** son las **piezas modulares** que componen la interfaz de usuario, como botones, menús, formularios y otros elementos interactivos. La integración eficaz de estos componentes no solo mejora la eficiencia del desarrollo, sino que también asegura que las aplicaciones sean coherentes y fáciles de actualizar o expandir.

El **desarrollo basado en componentes** (*component-based development* [CBD]) es una metodología que se centra en la creación de aplicaciones a partir de la integración de **componentes independientes** y **reutilizables.** Cada componente encapsula una funcionalidad específica de la aplicación, lo que permite que los desarrolladores ensamblen aplicaciones complejas como si estuvieran construyendo con bloques de Lego.

Los componentes pueden ser tan simples como un botón o tan complejos como un módulo de autenticación completo. Estos componentes están diseñados para ser **autónomos,** lo que significa que pueden **funcionar independientemente** de otros componentes. Esto facilita su **reutilización** en diferentes partes del proyecto o incluso en diferentes proyectos.

Algunas de las principales **ventajas** del desarrollo basado en componentes son:

- **Reutilización:** los componentes pueden ser reutilizados en múltiples proyectos, lo que ahorra tiempo y recursos.
- **Mantenibilidad:** los componentes modulares facilitan la actualización y el mantenimiento de las aplicaciones, ya que cada componente puede ser modificado sin afectar al resto del sistema.

- **Consistencia:** el uso de componentes garantiza que las aplicaciones sean consistentes en términos de apariencia y comportamiento, ya que estos componentes se pueden reutilizar en diferentes partes de la aplicación.
- **Escalabilidad:** las aplicaciones basadas en componentes son más fáciles de escalar, ya que los nuevos componentes pueden ser añadidos sin interferir con los existentes.

Proceso de integración de componentes

La integración de componentes en un proyecto sigue un **proceso sistemático** que garantiza que cada componente funcione correctamente dentro del contexto de la aplicación.

El **primer paso** en la integración de componentes es **seleccionar** los componentes adecuados para el proyecto. Esto implica evaluar los requisitos de la aplicación y decidir qué componentes pueden satisfacer mejor esas necesidades. Los componentes pueden ser **desarrollados internamente, adquiridos de terceros** o, incluso, ser de **código abierto.** Es crucial que los componentes seleccionados sean **compatibles** con el entorno de desarrollo del proyecto y que se integren sin problemas con los demás componentes.

Una vez seleccionados, los componentes deben ser **configurados** para que se adapten al estilo y funcionalidad específica del proyecto. Esto puede incluir la personalización de propiedades visuales (como colores, tamaños y tipografía), así como la configuración de eventos y comportamientos específicos.

```jsx
import React from 'react';
import Button from './components/Button';

function App() {
  return (
    <div>
      <h1>Bienvenido a mi aplicación</h1>
      <Button text="Haz clic aquí" color="blue" onClick={() => alert('Botón clicado!')} />
    </div>
  );
}

export default App;
```

Figura 13. Ejemplo de configuración de un componente en React. Fuente: elaboración propia.

En este ejemplo, un componente Button se integra en la aplicación principal, y sus propiedades (texto, color, y comportamiento al hacer clic) se configuran según las necesidades del proyecto.

Pruebas de integración

Antes de implementar los componentes en un entorno de producción, es esencial realizar **pruebas de integración** para asegurar que todos los componentes funcionen correctamente juntos. Estas pruebas deben verificar que los componentes interactúan adecuadamente, que no hay conflictos de dependencias, y que la interfaz de usuario es coherente y funcional.

La integración exitosa de componentes también implica la creación de una **documentación clara** y **detallada.** Esta documentación debe incluir información sobre cómo configurar y utilizar cada componente, así como instrucciones para su mantenimiento y actualización.

E l **mantenimiento** de los componentes integrados es un aspecto continuo del proceso de desarrollo. Los componentes deben ser **actualizados regularmente** para corregir errores, mejorar su rendimiento o adaptarlos a nuevos requisitos del proyecto.

Existen numerosas **herramientas** y **entornos de desarrollo** que facilitan la integración de componentes en proyectos de UI. Algunos de los más utilizados incluyen:

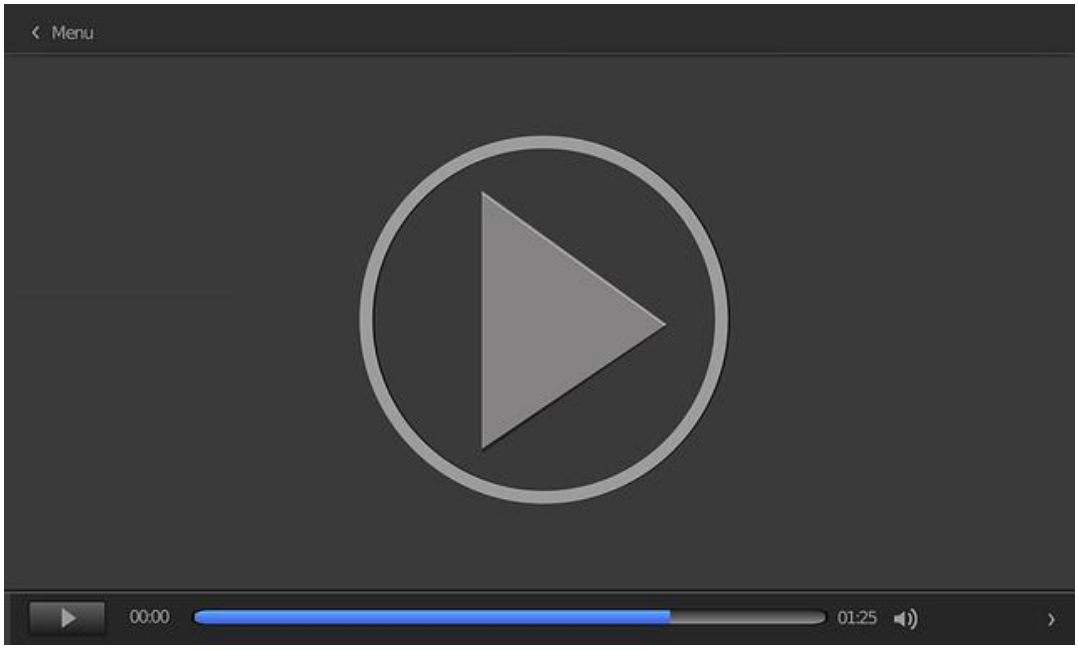
- **React:** una biblioteca de JavaScript ampliamente utilizada para construir interfaces de usuario basadas en componentes. React facilita la creación de componentes reutilizables y su integración en aplicaciones complejas.
- **Angular:** un *framework* de desarrollo que soporta la creación de aplicaciones a través de la integración de componentes modulares. Angular proporciona un conjunto robusto de herramientas para gestionar componentes a gran escala.
- **Vue.js:** otro *framework* popular para el desarrollo de UI. Vue.js es conocido por su simplicidad y flexibilidad en la integración de componentes.
- **Bootstrap:** un *framework* CSS que proporciona componentes predefinidos que pueden ser fácilmente integrados en cualquier proyecto para acelerar el desarrollo de interfaces.

Estas herramientas no solo permiten la integración eficiente de componentes, sino que también ofrecen **funciones adicionales** como la **gestión del estado, enrutamiento** y **pruebas automatizadas,** lo que facilita el desarrollo de aplicaciones complejas.

Cómo remover o eliminar elementos en el DOM con JavaScript

JuanDavid_Dev. (2023, enero 24). *09 Cómo remover o eliminar elementos del DOM con JavaScript* [Vídeo]. YouTube. https://www.youtube.com/watch?v=HYJbHsPFB60

Este vídeo explica cómo eliminar elementos por ID o por clase, y cómo puedes remover elementos de una lista o de un formulario.



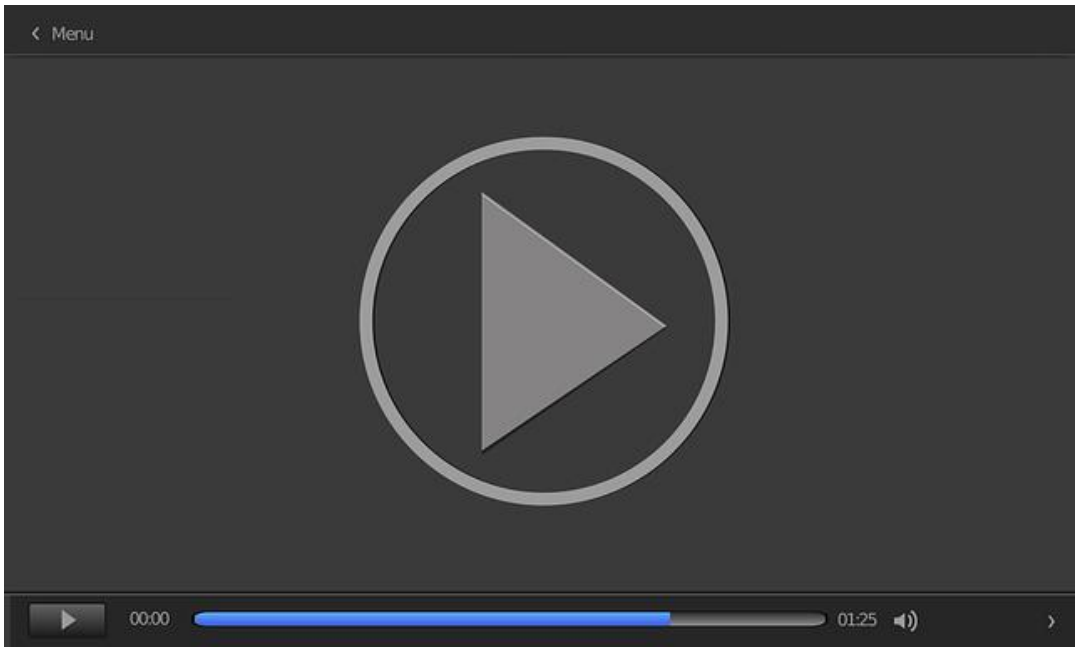
Accede al vídeo:

https://www.youtube.com/embed/HYJbHsPFB60

Aprende a agregar y crear elementos en el DOM con JavaScript

JuanDavid_Dev. (2023, enero 24). *10 Aprende a agregar y crear elementos en el DOM con JavaScript* [Vídeo]. YouTube. https://www.youtube.com/watch?v=MKzHruwcNOQ

Este vídeo explica cómo agregar y crear elementos en el DOM usando JavaScript, así como a agregarlos a un elemento existente.



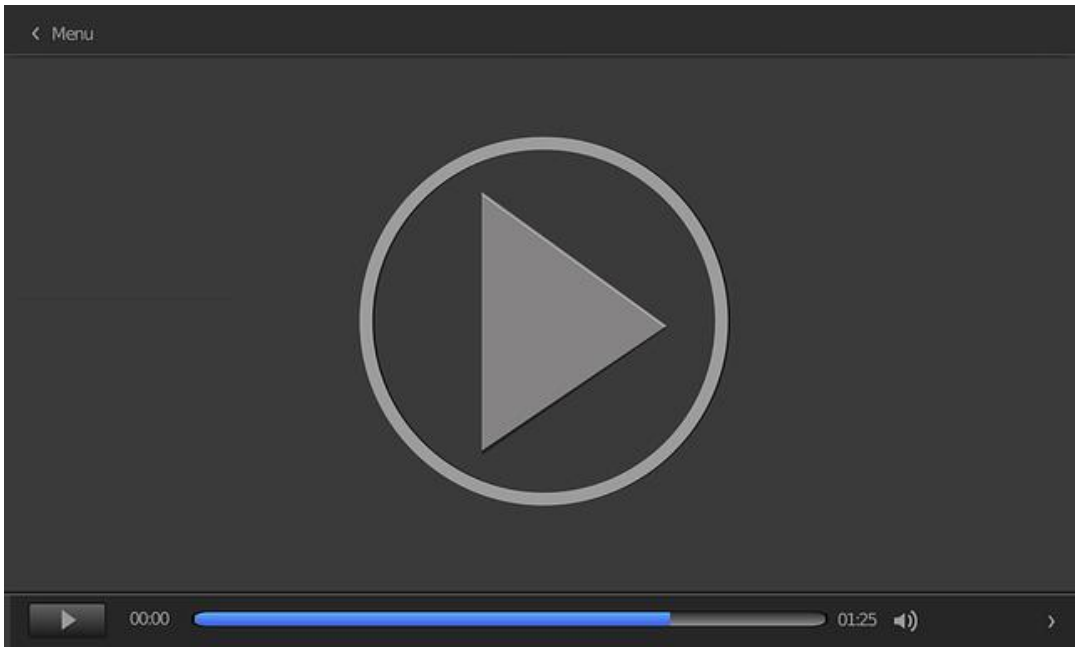
Accede al vídeo:

https://www.youtube.com/embed/MKzHruwcNOQ

Aprende a validar formularios con JavaScript y expresiones regulares

FalconMasters. (2020, julio 23). *Aprende a Validar Formularios con Javascript y Expresiones Regulares* [Vídeo]. YouTube. https://www.youtube.com/watch?v=s3pC93LgP18

En este vídeo se explica cómo validar formularios de forma profesional utilizando JavaScript y expresiones regulares.



Accede al vídeo:

https://www.youtube.com/embed/s3pC93LgP18

Entrenamiento 1

- Planteamiento del ejercicio

Crea una página web que permita al usuario añadir nuevos elementos de lista ( ) a una lista ordenada ( ) y eliminar el último elemento de la lista.

- Desarrollo paso a paso

- **Crear el archivo HTML:** incluye una lista ordenada ( ) y dos botones: uno para añadir un nuevo elemento y otro para eliminar el último elemento.
- **Escribe el JavaScript para añadir un elemento:** utiliza createElement y appendChild para añadir un nuevo elemento a la lista.
- **Escribe el JavaScript para eliminar un elemento:** utiliza removeChild para eliminar el último elemento de la lista.
- **Verifica el funcionamiento:** prueba la página web para asegurarte de que los elementos se añaden y eliminan correctamente.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Añadir y Eliminar Elementos</title>
</head>
<body>
    <h2>Lista de Tareas</h2>
    <ol id="lista">
        <li>Tarea 1</li>
        <li>Tarea 2</li>
    </ol>
    <button onclick="añadirElemento()">Añadir Elemento</button>
    <button onclick="eliminarElemento()">Eliminar Último Elemento</button>

    <script>
        function añadirElemento() {
            var nuevaTarea = document.createElement("li");
            nuevaTarea.textContent = "Nueva Tarea";
            document.getElementById("lista").appendChild(nuevaTarea);
        }

        function eliminarElemento() {
            var lista = document.getElementById("lista");
            if (lista.children.length > 0) {
                lista.removeChild(lista.lastElementChild);
            }
        }
    </script>
</body>
</html>
```

Figura 14. Solución. Fuente: elaboración propia.

Entrenamiento 2

- Planteamiento del ejercicio

Crea un formulario en HTML que requiera que el usuario ingrese una dirección de correo electrónico válida y un número de teléfono de diez dígitos. Implementa validaciones usando solo HTML5.

- Desarrollo paso a paso

  - **Crea el archivo HTML:** define un formulario con campos para el correo electrónico y el número de teléfono.
  - **Aplica validaciones HTML5:** usa el atributo `required` para hacer que los campos sean obligatorios y `pattern` para asegurarte de que el número de teléfono tiene diez dígitos.
  - **Prueba el formulario:** verifica que el formulario no se envía si las validaciones no se cumplen.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Validación de Formulario</title>
</head>
<body>
    <form>
        <label for="email">Correo Electrónico:</label>
        <input type="email" id="email" name="email" required><br><br>

        <label for="telefono">Número de Teléfono:</label>
        <input type="tel" id="telefono" name="telefono" pattern="[0-9]{10}" required><br><br

        <button type="submit">Enviar</button>
    </form>
</body>
</html>
```

Figura 15. Solución. Fuente: elaboración propia.

Entrenamiento 3

- Planteamiento del ejercicio

Crea una página que tenga un botón que cambie el color de fondo de un párrafo al hacer clic en él. Usa JavaScript para modificar el estilo CSS del párrafo.

- Desarrollo paso a paso

  - **Crea el archivo HTML:** incluye un párrafo y un botón.
  - **Escribe el JavaScript:** usa `style.backgroundColor` para cambiar el color de fondo del párrafo cuando se haga clic en el botón.
  - **Prueba el funcionamiento:** verifica que el color de fondo cambie al hacer clic en el botón.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Cambiar Color de Fondo</title>
</head>
<body>
    <p id="miParrafo">Este es un párrafo cuyo color de fondo cambiará.</p>
    <button onclick="cambiarColor()">Cambiar Color de Fondo</button>

    <script>
        function cambiarColor() {
            var parrafo = document.getElementById("miParrafo");
            parrafo.style.backgroundColor = "lightblue";
        }
    </script>
</body>
</html>
```

Figura 16. Solución. Fuente: elaboración propia.

Entrenamiento 4

- Planteamiento del ejercicio

Crea una página web que muestre un `div` con un color de fondo diferente según el ancho de la pantalla: azul si la pantalla es mayor a 600 px de ancho y verde si es menor.

- Desarrollo paso a paso

- **Crea el archivo HTML:** define un `div` en la página.
- **Aplica estilos CSS:** usa *media queries* para definir diferentes colores de fondo según el ancho de la pantalla.
- **Prueba la página:** redimensiona la ventana del navegador para verificar que el color de fondo cambia según el tamaño de la pantalla.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Diseño Responsivo</title>
    <style>
        #miDiv {
            width: 100%;
            height: 200px;
        }

        @media (max-width: 600px) {
            #miDiv {
                background-color: green;
            }
        }

        @media (min-width: 601px) {
            #miDiv {
                background-color: blue;
            }
        }
    </style>
</head>
<body>
    <div id="miDiv"></div>
</body>
</html>
```

Figura 17. Solución. Fuente: elaboración propia.

Entrenamiento 5

- Planteamiento del ejercicio

Crea un componente de botón reutilizable que se pueda integrar en diferentes partes de una página web. Este botón debe recibir como parámetro el texto que mostrará y la acción que ejecutará al hacer clic.

- Desarrollo paso a paso

  - **Crear el archivo HTML:** define un contenedor donde se integrará el botón reutilizable.
  - **Escribe el JavaScript para crear el componente:** define una función que cree el botón y lo integre en el DOM.
  - **Prueba la reutilización:** usa la función para crear varios botones con diferentes textos y acciones.

- Solución

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Componente Reutilizable</title>
</head>
<body>
    <div id="contenedorBotones"></div>

    <script>
        function crearBoton(texto, accion) {
            var boton = document.createElement("button");
            boton.textContent = texto;
            boton.onclick = accion;
            document.getElementById("contenedorBotones").appendChild(boton);
        }

        crearBoton("Alerta 1", function() { alert("Has hecho clic en el Botón 1"); });
        crearBoton("Alerta 2", function() { alert("Has hecho clic en el Botón 2"); });
        crearBoton("Alerta 3", function() { alert("Has hecho clic en el Botón 3"); });
    </script>
</body>
</html>
```

Figura 18. Solución. Fuente: elaboración propia.