React con JavaScript

Autor:

Kevin Cárdenas.

2023

Índice

1. Fundamentos de JavaScript

JavaScript es un lenguaje de programación utilizado en el desarrollo web para hacer que las páginas web sean interactivas y dinámicas. Es un lenguaje de scripting del lado del cliente que se ejecuta en el navegador web del usuario.

En 1995 JavaScript fue creado por Brendan Eich en tan solo 10 días mientras trabajaba en Netscape Communications Corporation. Originalmente se llamaba "Mocha" y luego "LiveScript" antes de convertirse en JavaScript. En 1996 Netscape entrega JavaScript a la ECMA International (European Computer Manufacturers Association) para estandarización. El estándar resultante se llamó "ECMAScript" para evitar conflictos de marca. En 1997 La primera edición de ECMAScript, ES1, se publica como ECMA-262. Se convierte en la base de JavaScript moderno. En 2002 Se publica la tercera edición de ECMAScript (ES3), que define muchas de las características que se utilizan comúnmente en JavaScript actualmente.

En 2009 Douglas Crockford presenta JSON (JavaScript Object Notation) como una forma ligera y eficiente de intercambiar datos. JSON se convierte en un formato de datos ampliamente utilizado, para en 2011 publicar ECMAScript 5 (ES5), que incluye características como el método `map(), filter(), reduce()` y más. En 2015 Se lanza ECMAScript 2015 (ES6 o ES2015), que es una actualización importante de JavaScript. Introduce clases, módulos, let y const, funciones flecha (=>), y muchas otras características.

La última especificación de ECMAScript, ES2020, se publica con características como el operador de fusión nula (??), el operador de encadenamiento opcional (?.), y más. JavaScript es el lenguaje de programación más utilizado en el mundo. Se ejecuta en todos los navegadores web modernos y se utiliza tanto en el desarrollo front-end como en el back-end con Node.js. También se utiliza en aplicaciones móviles y en el desarrollo de aplicaciones de escritorio.

1.1. Primeros pasos

Para instalar y utilizar JavaScript en un entorno local, debes tener en cuenta que JavaScript es principalmente un lenguaje de programación de cliente y se ejecuta en los navegadores web de los usuarios. No necesitas instalar JavaScript en tu computadora como lo harías con un software tradicional. Sin embargo, necesitarás un editor de código para escribir tus scripts de JavaScript y un navegador web para ejecutarlos.

1. Preparación del entorno.

    a) Descarga e instala Visual Studio Code desde el sitio web oficial.

    b) Crea una carpeta donde desees y hubicate ahí para crear tu proyecto.

    c) Crea un archivo llamado `index.html` y otro llamado `script.js`

2. Escribe Tu Código JavaScript

   a) Abre el archivo `script.js` haciendo doble clic en él en el explorador de archivos.

   b) Escribe tu código JavaScript en este archivo. Aquí tienes un ejemplo:

```
// script.js
function mostrarMensaje() {
    alert("Hola, mundo");
}
```

3. Crea una Página HTML para Ejecutar el JavaScript

   a) Abre el archivo `index.html` haciendo doble clic en él en el explorador de archivos.

   b) Agrega el siguiente código HTML para incluir tu archivo JavaScript:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Ejemplo de JavaScript</title>
</head>
<body>
    <button onclick="mostrarMensaje()">Haz clic</button>
    <script src="script.js"></script>
</body>
</html>
```

   En este ejemplo, hemos creado una página HTML simple con un botón que llama a la función `mostrarMensaje()` cuando se hace clic en él. También hemos incluido el archivo JavaScript `script.js` usando la etiqueta `<script>`.

4. Ejecuta la Página HTML

   a) Guarda tus archivos si no lo has hecho ya.

   b) Enu tu navegador de archivos ve al archivo `index.html` que creamos, y ejecutalo.

   c) Se abrirá una nueva pestaña en tu navegador web y podrás hacer clic en el botón para ver el mensaje emergente que dice “Hola, mundo”.

1.2. Variables y tipos de datos

Declaración de variables

Las variables se utilizan para almacenar y manipular datos en JavaScript. Para declarar una variable, puedes usar las palabras clave *var, let, o const*. Aquí tienes ejemplos de cómo declarar variables:

```
var nombre = "Juan"; // Declarando la variable con var
let edad = 30; // Declarando la variable con let (preferible para variables que cambian)
const PI = 3.1416; // Declarando la constante con const
```

- Declarando la variable con let

```
let variable = "Juan"
console.log(variable) // Se muestra "Juan"
let precioDescuento = 220 //camelCase
console.log(precioDescuento) //Se muestra "220"
variable = 20
console.log(variable)// Se muestra 20
variable = true
console.log(variable)// Se muestra true

let precio
console.log(precio) //Se muestra "undefined"
```

- Declarando la constantes con const

```
  //const
  const constante = "Juan"
  const precioDescuento = 220

  //No se puede reasignar es constantes
  //No pueden iniciar sin un valor
  constante = 20 //Se muestra un error porque no se puede editar el valor de una
    constante despues de iniciarla
```

Tipos de datos

En JavaScript, cada valor tiene un tipo asociado. Hay varios tipos de datos que puedes encontrar:

- Number: Representa tanto números enteros como flotantes
- String: Representa una secuencia de caracteres. Ejemplo: `"Hello, world"`.

- Boolean: Representa un valor verdadero o falso (true o false).
- Undefined: Representa un valor que no ha sido asignado a una variable.
- Arreglos (colecciones ordenadas de datos)
- Null: Representa un valor nulo o "ningún valor".
- BigInt: Representa números enteros más grandes que el límite de $2^53 - 1$ que pueden ser representados con Number.
- Symbol: Representa un valor único que no puede ser cambiado.
- Object: Puede contener cualquier combinación de valores primitivos y objetos. Incluye funciones, arrays y expresiones regulares.
- Function: Una secuencia de instrucciones que realiza una tarea o calcula un valor.

Por ejemplo:

```
let numero = 42; // Numero entero
let precio = 19.99; // Numero decimal
let nombre = "Maria"; // Cadena de texto
let esActivo = true; // Booleano (verdadero)
let persona = { nombre: "Juan", edad: 25 }; // Objeto
let colores = ["rojo", "verde", "azul"]; // Arreglo
let valorNulo = null; // Valor nulo
let valorNoDefinido; // Valor no definido (undefined)
```

Conversión de tipos de datos

En muchas situaciones, es necesario convertir un tipo de dato a otro. Hay dos categorías principales de conversiones en JavaScript: explícitas y coerciones (implícitas).

1. Coerción (Conversión Implícita): Cuando operas con dos valores de diferentes tipos, JavaScript intentará convertir uno de los valores para que los dos valores tengan el mismo tipo. Esto se llama coerción. Por ejemplo:

```
"5" + 3 // Resulta en el string "53"
"5" - 3 // Resulta en el number 2
```

2. Conversión Explícita: Puedes convertir explícitamente valores de un tipo a otro usando métodos proporcionados por JavaScript:

- String a Number:

```
parseInt("123") // 123
parseFloat("123.45") // 123.45
Number("123") // 123
```

- Number a String:

```
(123).toString() // "123"
String(123) // "123"
```

Por ejemplo:

```
let numeroTexto = "42";
let numero = parseInt(numeroTexto); // Convierte una cadena en un numero entero
console.log(numero); // Resultado: 42

let decimalTexto = "3.14";
let decimal = parseFloat(decimalTexto); // Convierte una cadena en un numero decimal
console.log(decimal); // Resultado: 3.14

let valorBooleano = false;
let textoBooleano = String(valorBooleano); // Convierte un booleano en una cadena de texto
console.log(textoBooleano); // Resultado: "false"
```

Operadores y expresiones

1. Operadores aritméticos:

   JavaScript admite operadores aritméticos estándar, como +, -, *, / y % (módulo) para realizar operaciones matemáticas.

   Ejemplo:

```
let num1 = 10;
let num2 = 5;

let suma = num1 + num2; // Suma
let resta = num1 - num2; // Resta
let multiplicacion = num1 * num2; // Multiplicacion
let division = num1 / num2; // Division
let modulo = num1 % num2; // Modulo

console.log(suma, resta, multiplicacion, division, modulo);
```

2. Operadores de comparación:

   JavaScript utiliza operadores de comparación para comparar valores. Los operadores más comunes son == (igual a), != (diferente de), === (igual a en valor y tipo), !== (diferente en valor o tipo), >(mayor que), <(menor que), >= (mayor o igual que), y <= (menor o igual que).

   Ejemplo:

```
let x = 5;
let y = "5";

console.log(x == y); // true (compara valores)
console.log(x === y); // false (compara valores y tipos)
console.log(x != y); // false
console.log(x !== y); // true
console.log(x > 3); // true
console.log(x < 2); // false
```

3. Operadores lógicos:

   Los operadores lógicos (`&&` para "y", `||` para "o", `!` para "no") se utilizan para evaluar expresiones booleanas.

   Ejemplo:

```
let a = true;
let b = false;

console.log(a && b); // false (true y false es false)
console.log(a || b); // true (true o false es true)
console.log(!a); // false (negacion de true es false)
```

1.3. Funciones en JavaScript

Declaración de funciones

En JavaScript, puedes declarar funciones utilizando la palabra clave function. Las funciones son bloques de código que pueden ser reutilizados para realizar una tarea específica. Aquí tienes un ejemplo de cómo declarar una función:

```
function saludar() {
  console.log("Hola, mundo");
}
```

Parámetros y argumentos

Las funciones pueden recibir parámetros, que son variables que actúan como entradas para la función. Los argumentos son los valores reales que se pasan a la función cuando se llama. Por ejemplo:

```
function saludar(nombre) {
    console.log("Hola, " + nombre);
}

saludar("Juan"); // Llama a la funcion con el argumento "Juan"
```

Retorno de valores

Las funciones pueden devolver un valor utilizando la palabra clave return. Esto permite que una función produzca un resultado que puede ser utilizado en otras partes del código. Ejemplo:

```
function suma(a, b) {
    return a + b;
}

let resultado = suma(5, 3); // Llama a la funcion y almacena el resultado en "resultado"
console.log(resultado); // Imprime 8
```

Ámbito de variables (scope)

El ámbito de una variable se refiere a dónde en el código puede ser accedida. En JavaScript, las variables declaradas dentro de una función tienen un ámbito local, lo que significa que solo pueden ser accedidas dentro de esa función. Las variables declaradas fuera de todas las funciones tienen un ámbito global y pueden ser accedidas en cualquier parte del código.

```
let globalVariable = "Soy global"; // Variable global

function ejemplo() {
    let localVariable = "Soy local"; // Variable local
    console.log(localVariable); // Puede accederse dentro de la funcion
}

console.log(globalVariable); // Puede accederse desde cualquier parte
console.log(localVariable); // Genera un error, no puede accederse fuera de la funcion
```

Funciones anónimas y funciones flecha (arrow functions)

Las funciones anónimas son funciones sin nombre y se pueden asignar a variables. Las funciones flecha (=>) son una sintaxis más concisa para declarar funciones anónimas. Ejemplos:

```
// Funcion anonima
let suma = function(a, b) {
    return a + b;
};

// Funcion flecha
let resta = (a, b) => a - b;
```

1.4. Estructuras de control

Sentencias condicionales (if, else if, else)

Las sentencias condicionales permiten que tu código tome decisiones basadas en condiciones específicas. En JavaScript, puedes usar if, else if, y else para construir estructuras condicionales. Aquí tienes un ejemplo:

```
let edad = 18;

if (edad < 18) {
    console.log("Eres menor de edad");
} else if (edad >= 18 && edad < 65) {
    console.log("Eres adulto");
} else {
    console.log("Eres un adulto mayor");
}
```

Sentencias condicionales (if, else if, else)

El operador ternario es una forma más concisa de expresar una sentencia condicional simple, y tiene la estructura ‘condición ? valorSiVerdadero : valorSiFalso‘. A continuación, un ejemplo que refleja el mismo comportamiento del código anterior utilizando operadores ternarios:

```
let mensaje = (edad < 18)
              ? "Eres menor de edad"
              : (edad < 65)
              ? "Eres adulto"
              : "Eres un adulto mayor";
```

```
6
7 console.log(mensaje);
```

El operador ternario puede ser encadenado, pero es importante usarlo con precaución para mantener la legibilidad del código. En el ejemplo anterior, se encadenaron dos operadores ternarios para cubrir las tres posibles condiciones.

Bucles (for, while)

Los bucles te permiten repetir una acción varias veces. JavaScript admite bucles for y while. Aquí tienes ejemplos de ambos:

```
1 // Bucle for
2 for (let i = 0; i < 5; i++) {
3     console.log("Iteracion " + i);
4 }
5
6 // Bucle while
7 let contador = 0;
8 while (contador < 5) {
9     console.log("Iteracion " + contador);
10    contador++;
11 }
```

Sentencia switch

La sentencia switch se utiliza para tomar decisiones basadas en el valor de una expresión. Aquí tienes un ejemplo:

```
1 let dia = "lunes";
2
3 switch (dia) {
4     case "lunes":
5         console.log("Comienza la semana");
6         break;
7     case "viernes":
8         console.log("Casi es fin de semana");
9         break;
10    default:
11        console.log("Es un dia cualquiera");
12 }
```

1.5. Programación Orientada a Objetos

Conceptos básicos de POO:

La Programación Orientada a Objetos es un paradigma de programación que se basa en la idea de modelar el mundo real en términos de objetos. Los objetos son entidades que tienen propiedades (atributos) y métodos (funciones) que operan en esos atributos. Los conceptos clave de POO incluyen:

- `Clases:` Plantillas o "planos" para crear objetos. Definen la estructura y el comportamiento de los objetos.
- `Objetos:` Instancias de clases que contienen datos y funciones asociadas (métodos).
- `Herencia:` Un mecanismo que permite a una clase heredar las propiedades y métodos de otra clase.
- `Encapsulación:` El ocultamiento de los detalles internos de un objeto y la exposición de una interfaz pública para interactuar con él.
- `Polimorfismo:` La capacidad de objetos de diferentes clases de responder al "mismo" método de manera diferente.

Objetos y clases en JavaScript

JavaScript es un lenguaje de programación orientado a objetos. Sin embargo, su acercamiento a la programación orientada a objetos (POO) se diferencia de otros lenguajes como Java o C++. En lugar de clases tradicionales, JavaScript usa objetos y prototipos para modelar estructuras de datos y relaciones de herencia. Un objeto en JavaScript es una colección de pares clave-valor, donde las claves son cadenas (o símbolos) y los valores pueden ser de cualquier tipo.

```
// Definicion de un objeto en JavaScript
let persona = {
nombre: "Juan",
edad: 30,
saludar: function() {
console.log("Hola, soy " + this.nombre);
}
};

// Uso del objeto
console.log(persona.nombre); // Accede a propiedades
persona.saludar(); // Llama a metodos
```

Creación de objetos mediante funciones constructoras

Antes de la introducción de la sintaxis de clases en ES6, las funciones constructoras eran la principal manera de crear "tipos"de objetos y emular clases en JavaScript. Estas funciones, cuando son llamadas con la palabra clave new, crean instancias de objetos con propiedades y métodos específicos.

```
// Definicion de una funcion constructora
function Persona(nombre, edad) {
this.nombre = nombre;
this.edad = edad;
this.saludar = function() {
console.log("Hola, soy " + this.nombre);
};
}

// Creacion de objetos usando la funcion constructora
let juan = new Persona("Juan", 30);
let maria = new Persona("Maria", 25);
```

Creación de objetos mediante clases (introducción a la sintaxis de clase)

Con ECMAScript 2015 (ES6), JavaScript incorporó una sintaxis de clases que proporciona una forma más estructurada y familiar (para quienes vienen de otros lenguajes OOP) de definir y crear objetos.

```
// Definicion de una clase
class Persona {
constructor(nombre, edad) {
this.nombre = nombre;
this.edad = edad;
}
saludar() {
console.log("Hola, soy " + this.nombre);
}
}

// Creacion de objetos usando la clase
let juan = new Persona("Juan", 30);
let maria = new Persona("Maria", 25);
```

Herencia

JavaScript implementa la herencia a través de prototipos. Con la sintaxis de clases de ES6, la herencia se facilita mediante la palabra clave extends. Un objeto puede heredar propiedades y métodos de otro, lo que permite establecer relaciones jerárquicas entre clases.

```
// Definicion de una clase Padre
class Persona {
constructor(nombre, edad) {
this.nombre = nombre;
this.edad = edad;
}
saludar() {
console.log("Hola, soy " + this.nombre);
}
}

// Definicion de una clase Hijo que hereda de Persona
class Estudiante extends Persona {
constructor(nombre, edad, grado) {
super(nombre, edad); // Llama al constructor de la clase padre
this.grado = grado;
}
estudiar() {
console.log(this.nombre + " esta estudiando en el grado " + this.grado);
}
}

let juan = new Estudiante("Juan", 15, "Noveno");
juan.saludar(); // Accede a metodos heredados
juan.estudiar(); // Llama a metodos propios
```

Prototipos y prototipado

JavaScript es inherentemente un lenguaje basado en prototipos. A diferencia de los lenguajes basados en clases, donde las clases definen la estructura y los comportamientos que las instancias de la clase heredarán, en JavaScript, cada objeto puede tener un prototipo del que hereda directamente propiedades y métodos. Estos prototipos son objetos en sí mismos, y pueden tener sus propios prototipos, creando una cadena de prototipos. Al final de esta cadena está Object.prototype, el prototipo base de todos los objetos en JavaScript.

Cuando intentas acceder a una propiedad o método de un objeto, JavaScript primero verifica si el objeto tiene esa propiedad o método. Si no lo encuentra, busca en el prototipo del objeto, y así sucesivamente, hasta llegar al final de la cadena de prototipos. Si la propiedad o método no se encuentra en toda la cadena, se devuelve undefined.

```
let personaPrototipo = {
saludar: function() {
console.log("Hola, soy " + this.nombre);
}
};

// Creacion de objetos usando Object.create() y asignacion del prototipo
let juan = Object.create(personaPrototipo);
juan.nombre = "Juan";
juan.saludar(); // Llama al metodo desde el prototipo

let maria = Object.create(personaPrototipo);
maria.nombre = "Maria";
maria.saludar();
```

Un uso común del prototipado es extender objetos existentes. Por ejemplo, si tienes una biblioteca de funciones que trabaja con objetos tipo "persona", pero deseas agregar comportamientos adicionales (como nuevos métodos) sin modificar la biblioteca original, puedes hacerlo extendiendo el prototipo.

Adicionalmente, es importante mencionar que la propiedad especial `__proto__` (aunque no está estandarizada y se considera obsoleta en favor de `Object.getPrototypeOf() y Object.setPrototypeOf()`) permite acceder y modificar el prototipo de un objeto en tiempo de ejecución.

Finalmente, aunque el prototipado es poderoso, puede ser confuso si no se utiliza adecuadamente. La introducción de la sintaxis de clases en ES6 proporciona una capa de abstracción sobre este sistema de prototipos, ofreciendo una forma más familiar y estructurada de trabajar con herencia en JavaScript.

1.6. Arrays

Un array es una estructura de datos que permite almacenar múltiples valores en una única variable. En JavaScript, estos valores pueden ser de cualquier tipo: números, cadenas, objetos, y hasta otros arrays. Los arrays son objetos del tipo `Array`, pero poseen características y métodos especiales para gestionar las colecciones de datos.

Creación de Arrays

Puedes crear un array de las siguientes maneras:

1. Usando corchetes (`[]`):

```
let frutas = ["manzana", "banana", "cereza"];
```

2. Usando el constructor `Array()`:

```
let numeros = new Array(1, 2, 3, 4, 5);
```

Acceso a los elementos de un Array

Para acceder a un elemento específico, se utiliza el índice del elemento entre corchetes. Los índices comienzan en 0.

```
console.log(frutas[0]); // Salida: manzana
```

Modificar un Array

Puedes cambiar el valor de un elemento usando su índice:

```
frutas[1] = "naranja";
console.log(frutas); // Salida: ["manzana", "naranja", "cereza"]
```

Métodos útiles de Arrays

JavaScript ofrece una variedad de métodos para trabajar con arrays: `push()`, `pop()`, `shift()`, `unshift()`, `splice()`, `slice()`, `join()`, `reverse()`, `sort()`, `map()`, `filter()` y `reduce()`, entre otros.

Arrays multidimensionales

Es posible tener arrays dentro de otros arrays, conocidos como arrays multidimensionales:

```
let matriz = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
console.log(matriz[1][2]); // Salida: 6
```

Iteración sobre Arrays

Para recorrer un array, se pueden usar bucles como `for` o `forEach()`.

```
for(let i = 0; i < frutas.length; i++) {
    console.log(frutas[i]);
}

// Usando forEach
frutas.forEach(function(fruta) {
    console.log(fruta);
});
```

1.7. Manipulación del DOM

El Document Object Model (DOM) es una representación estructurada de una página web. Usando JavaScript, es posible interactuar y modificar el DOM, permitiendo crear páginas web dinámicas.

Acceder a elementos

Supongamos que tenemos el siguiente fragmento de HTML:

```
<div id="miDiv" class="claseDiv">Hola, soy un div.</div>
<p class="claseP">Primer parrafo.</p>
<p class="claseP">Segundo parrafo.</p>
```

Usando JavaScript, podemos acceder a estos elementos de varias maneras:

```
let elementoPorID = document.getElementById("miDiv");
let elementosPorClase = document.getElementsByClassName("claseP");
let primerP = elementosPorClase[0];
```

Modificar elementos

```
elementoPorID.innerHTML = "Contenido cambiado!";
primerP.style.color = "blue";
```

Agregar y eliminar elementos

```
<div id="contenedor"></div>
```

```
let nuevoElemento = document.createElement("p");
nuevoElemento.innerHTML = "Soy un parrafo nuevo.";
document.getElementById("contenedor").appendChild(nuevoElemento);

// Para eliminar
contenedor.removeChild(nuevoElemento);
```

Eventos

Suponiendo que tienes un botón en tu HTML:

```
<button id="miBoton">Haz click</button>
```

Puedes agregar un oyente de eventos a este botón para que, al hacer click, muestre un mensaje:

```
document.getElementById("miBoton").addEventListener("click", function() {
    alert("Gracias por hacer click");
});
```

A través de estos ejemplos, podemos ver cómo JavaScript interactúa con el HTML a través del DOM, permitiendo a los desarrolladores crear experiencias web ricas e interactivas.

Generación de contenido HTML desde JavaScript

JavaScript, a través del DOM (Document Object Model), permite la creación dinámica de elementos HTML y la inserción de estos en una página web existente. Esta capacidad es esencial para la interacción en tiempo real y la adaptación del contenido sin necesidad de recargar toda la página.

Ejemplo práctico: Creación de un formulario dinámico.

Supongamos que queremos generar un formulario de registro dinámicamente cuando un usuario hace clic en un botón.

```
<!-- Estructura HTML inicial -->
<button id="mostrarFormulario">Mostrar Formulario</button>
<div id="formularioContainer"></div>
```

Ahora, usaremos JavaScript para generar el formulario dentro del ‘div‘ con id ‘formularioContainer‘ cuando se haga clic en el botón:

```
document.getElementById("mostrarFormulario").addEventListener("click", function() {
    let contenedor = document.getElementById("formularioContainer");

    // Creacion del elemento form
    let formulario = document.createElement("form");
```

```
    formulario.action = "#"; // Por simplicidad no lo enviaremos a ningun lugar
    formulario.method = "post";

    // Creacion de un input para el nombre
    let inputNombre = document.createElement("input");
    inputNombre.type = "text";
    inputNombre.placeholder = "Introduce tu nombre";
    formulario.appendChild(inputNombre);

    // Creacion de un boton de envio
    let botonEnviar = document.createElement("input");
    botonEnviar.type = "submit";
    botonEnviar.value = "Registrar";
    formulario.appendChild(botonEnviar);

    // agregar el formulario al contenedor
    contenedor.appendChild(formulario);
});
```

Con el código anterior, cuando un usuario haga clic en el botón "Mostrar Formulario", se generará dinámicamente un formulario simple con un campo de texto para el nombre y un botón de envío, y se insertará dentro del ‘div‘ con id ‘formularioContainer‘.

Es importante destacar que, aunque generar contenido HTML a través de JavaScript es poderoso y flexible, también puede ser menos eficiente que tener el contenido estático en el HTML, especialmente si se genera una gran cantidad de contenido dinámico. Por lo tanto, es esencial utilizar esta técnica con prudencia y optimizar el rendimiento cuando sea necesario.

2. Fundamentos de HTML y CSS

2.1. Introducción a HTML (Hypertext Markup Language)

HTML (Hypertext Markup Language) es el lenguaje estándar utilizado para crear documentos web. Se trata de un lenguaje de marcado que se utiliza para estructurar el contenido de una página web y definir cómo se debe presentar. Los documentos HTML contienen elementos y etiquetas que describen la estructura del contenido y permiten la inclusión de texto, imágenes, enlaces y otros elementos interactivos.

Estructura básica de un documento HTML:

Un documento HTML básico consta de dos partes principales: el encabezado (<head>) y el cuerpo (<body>). El encabezado suele contener información sobre la página, como el título, enlaces a hojas de estilo CSS y metadatos. El cuerpo contiene el contenido visible de la página.

```
<!DOCTYPE html>
<html>
<head>
    <title>Titulo de la Pagina</title>
</head>
<body>
    <h1>Encabezado principal</h1>
    <p>Este es un parrafo de ejemplo.</p>
    <!-- Mas contenido aqui -->
</body>
</html>
```

Etiquetas HTML: cabeza, cuerpo, párrafos, encabezados, listas, enlaces, imágenes, etc.:

HTML ofrece una variedad de etiquetas para estructurar y formatear el contenido. Algunas etiquetas comunes incluyen:

- <h1>, <h2>, <h3>, ... <h6>: Encabezados de diferentes niveles.
- <p>: Párrafos de texto.
- : Lista no ordenada.
- : Lista ordenada.
- : Elemento de lista.

- <a href=ÜRL»Texto del enlace: Enlaces a otras páginas web.
- <img src=ÜRL.ªlt="Texto alternativo»: Imágenes.
- y : Elementos de división y de línea, respectivamente, que se utilizan para estructurar el diseño y aplicar estilos.

Atributos HTML: src, href, alt, class, id, etc.

Las etiquetas HTML a menudo tienen atributos que modifican su comportamiento o aspecto. Algunos atributos comunes incluyen:

- src: Define la fuente de una imagen o un archivo multimedia.
- href: Especifica la URL de destino de un enlace.
- alt: Proporciona un texto alternativo para las imágenes (útil para accesibilidad).
- class: Asigna una o más clases CSS a un elemento para aplicar estilos.
- id: Asigna un identificador único a un elemento.
- Otros atributos específicos para cada etiqueta.

Anidación de elementos HTML

Los elementos HTML pueden anidarse dentro de otros elementos. Esto permite crear estructuras jerárquicas y definir relaciones entre elementos. Por ejemplo:

```
<ul>
  <li>Elemento 1</li>
  <li>Elemento 2</li>
  <li>Elemento 3</li>
</ul>
```

Comentarios en HTML

Puedes añadir comentarios en tu código HTML para documentar o desactivar partes del código. Los comentarios no son visibles para los usuarios y se crean utilizando <!– y –>.

```
<!-- Este es un comentario en HTML -->
<p>Este es un parrafo visible.</p>
```

2.2. Introducción a CSS (Cascading Style Sheets)

CSS (Cascading Style Sheets) es un lenguaje de diseño utilizado para controlar la presentación y el estilo de los documentos HTML. Permite definir cómo se deben mostrar los elementos HTML, como el color, la fuente, el espaciado, el diseño y otros aspectos visuales. CSS se utiliza para separar la estructura y el contenido del diseño de una página web, lo que facilita la consistencia y la personalización de la apariencia de un sitio web.

Estilos en línea vs. hojas de estilo externas

CSS se puede aplicar de diferentes maneras en una página web. Los estilos en línea se definen directamente en el elemento HTML utilizando el atributo `style`, mientras que las hojas de estilo externas se almacenan en archivos CSS independientes que se vinculan con el documento HTML. Las hojas de estilo externas suelen ser más eficientes y fáciles de mantener, ya que permiten aplicar estilos consistentes a múltiples páginas web.

Estilos en Línea (Inline Styles):

Los estilos en línea se aplican directamente a elementos HTML utilizando el atributo `style`. A continuación, se muestra un ejemplo de cómo se aplican estilos en línea a un elemento `<p>`:

```
<!DOCTYPE html>
<html lang="es">

<head>
</head>

<body>
    <p style="color: blue; font-size: 16px;">Este es un parrafo con estilo en linea.</p>
</body>

</html>
```

En este caso, el texto dentro del párrafo se mostrará en color azul y tendrá un tamaño de fuente de 16 píxeles.

Hojas de Estilo Externas:

Las hojas de estilo externas se crean como archivos CSS independientes y se vinculan con el documento HTML mediante la etiqueta `<link>`. Aquí hay un ejemplo de cómo vincular una hoja de estilo externa llamada `estilos.css`:

```
<!DOCTYPE html>
```

```html
<html lang="es">

<head>
    <link rel="stylesheet" type="text/css" href="estilos.css">
</head>

<body>
    <p>Este es un parrafo con estilo de hoja de estilo externa.</p>
</body>

</html>
```

En el archivo `estilos.css`, puedes definir estilos que afecten a múltiples elementos HTML:

```css
/* estilos.css */
p {
    color: green;
    font-size: 18px;
}
```

En este ejemplo, todos los elementos `<p>` en la página tendrán un color de texto verde y un tamaño de fuente de 18 píxeles gracias a la hoja de estilo externa.

Selectores CSS: selectores de elemento, de clase y de ID

Los selectores CSS se utilizan para apuntar a elementos específicos en un documento HTML y aplicarles estilos. Los tipos comunes de selectores incluyen:

- Selectores de elemento: Apuntan a todos los elementos de un tipo específico. Por ejemplo, el selector p se aplicaría a todos los párrafos en un documento HTML y les aplicaría un estilo común.

```css
p {
  color: blue;
  font-size: 16px;
}
```

- Selectores de clase: Apuntan a elementos que tienen una clase específica. Supongamos que tienes elementos con la clase destacado, puedes aplicar un estilo a todos los elementos con esta clase.

```css
.destacado {
  background-color: yellow;
  font-weight: bold;
}
```

- Apuntan a un elemento con un identificador único. Si tienes un elemento con el ID encabezado, puedes aplicar un estilo solo a ese elemento.

```
#encabezado {
  font-size: 24px;
  color: green;
}
```

Supongamos que tienes el siguiente HTML con elementos que utilizan los selectores CSS mencionados anteriormente:

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="estilos.css">
</head>
<body>
    <p>Este es un parrafo normal.</p>
    <p class="destacado">Este es un parrafo con clase "destacado".</p>
    <p id="encabezado">Este es un parrafo con ID "encabezado".</p>
</body>
</html>
```

En este ejemplo:

- El primer párrafo <p>no tiene ninguna clase ni ID, por lo que se aplicarán los estilos predeterminados definidos en el archivo CSS.
- El segundo párrafo <p>tiene la clase "destacado", por lo que se aplicarán los estilos definidos en el selector de clase .destacado del archivo CSS.
- tiene la clase "destacado", por lo que se aplicarán los estilos definidos en el selector de clase .destacado del archivo CSS.

Propiedades CSS: color, fondo, fuente, margen, relleno, etc.

Las propiedades CSS controlan diferentes aspectos de la presentación de los elementos. Algunas propiedades comunes incluyen:

- color: Define el color del texto.
- font-family: Especifica la fuente utilizada para el texto.

- margin y padding: Controlan el espacio alrededor y dentro de un elemento.
- Otras propiedades como border, width, height, text-align, entre otras.

Modelo de caja en CSS

El modelo de caja en CSS se refiere a cómo se representan los elementos HTML, incluyendo su contenido, relleno, borde y margen. Comprender el modelo de caja es fundamental para el diseño web efectivo, ya que permite controlar el espacio y el diseño de los elementos en la página.

2.3. Integración de HTML, CSS y JavaScript

Desarrollemos una pagina que basado un un formulario genere un mensaje.

Generemos los siguientes archivos en la misma carpeta:

1. `estilos.css` Carga con los estilos.

```css
/* estilos.css */
body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin: 20px;
}

h1 {
    color: #333;
}

form {
    border: 1px solid #ccc;
    padding: 20px;
    max-width: 400px;
    margin: 0 auto;
}

label {
    display: block;
    margin-bottom: 10px;
}

input[type="text"],
```

```
textarea {
    width: 100%;
    padding: 10px;
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
}

button {
    background-color: #0074D9;
    color: #fff;
    padding: 10px 20px;
    border: none;
    cursor: pointer;
}

button:hover {
    background-color: #0056b3;
}

#resultadoSaludo {
    margin-top: 20px;
    font-weight: bold;
}
```

2. `index.html` Carga con la información, que en este caso es el formulario.

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="estilos.css">
</head>
<body>
    <h1>Formulario de Saludo</h1>

    <form id="formularioSaludo">
        <label for="nombre">Nombre:</label>
        <input type="text" id="nombre" required>

        <label for="mensaje">Mensaje:</label>
        <textarea id="mensaje" required></textarea>
```

```
15
16          <button type="submit">Generar</button>
17      </form>
18
19      <div id="resultadoSaludo"></div>
20
21      <script src="script.js"></script>
22  </body>
23  </html>
```

3. `script.js` Carga con la lógica.

```
1  // script.js
2
3  // 1. Seleccionamos el formulario y el elemento donde mostraremos el resultado
4  var formulario = document.getElementById("formularioSaludo");
5  var resultadoSaludo = document.getElementById("resultadoSaludo");
6
7  // 2. Agregamos un evento al formulario cuando se envia (se hace clic en el boton "
       Generar")
8  formulario.addEventListener("submit", function(event) {
9      // 3. Prevenimos el comportamiento predeterminado del formulario, que seria
       enviarlo y recargar la pagina
10     event.preventDefault();
11
12     // 4. Capturamos el valor del campo "nombre" y "mensaje" del formulario
13     var nombre = document.getElementById("nombre").value;
14     var mensaje = document.getElementById("mensaje").value;
15
16     // 5. Creamos un mensaje de saludo combinando el nombre y el mensaje del usuario
17     var saludo = "Hola, " + nombre + "! Tu mensaje es: " + mensaje;
18
19     // 6. Actualizamos el contenido del elemento "resultadoSaludo" con el mensaje de
       saludo
20     resultadoSaludo.innerHTML = saludo;
21 });
```

3. Instalaciones

Para comenzar a trabajar con React, es necesario tener instaladas las siguientes herramientas:

- **Google:** El motor de búsqueda más popular del mundo. Se puede usar para encontrar información sobre React y otros temas relacionados.
- **VS Code:** Un editor de código gratuito y de código abierto. Es una excelente opción para trabajar con React, ya que tiene una amplia gama de extensiones y funciones que facilitan el desarrollo de aplicaciones React.
- **Postman:** Una herramienta de prueba API gratuita y de código abierto. Se puede usar para probar las API de React.
- **Git:** Un sistema de control de versiones gratuito y de código abierto. Se puede usar para administrar el código de su aplicación React.
- **Node.js:** Un entorno de ejecución JavaScript de código abierto y multiplataforma. Es necesario para ejecutar aplicaciones React.
- **VS Code Extensions:** Las siguientes extensiones de VS Code son útiles para trabajar con React:
  - **Auto Rename Tag:** Esta extensión agrega un botón a la barra de herramientas que permite renombrar rápidamente las etiquetas HTML.
  - **ES7 React/Redux/GraphQL/React-Native snippets:** Esta extensión proporciona fragmentos de código para los componentes React más comunes.
  - **Auto Close Tag:** Esta extensión cierra automáticamente las etiquetas HTML abiertas.

3.1. Instalación de Node.js y npm

Node.js es un entorno de ejecución JavaScript de código abierto y multiplataforma. Es necesario para ejecutar aplicaciones React.

Para instalar Node.js en Linux, puedes usar el siguiente comando:

```
sudo apt install nodejs
```

Una vez que hayas instalado Node.js, también debes instalar npm, el gestor de paquetes de Node.js. Puedes instalar npm con el siguiente comando:

```
sudo apt install npm
```

3.2. Hola Mundo en React

Abre una terminal y navega hasta la carpeta donde deseas crear el proyecto React. Ejecuta el siguiente comando para inicializar el proyecto:

```
npx create-react-app my-app
```

Este comando creará una carpeta llamada `my-app` con un proyecto React inicializado.

A continuación, entra en el archivo `src/App.js` y escribe el siguiente código:

```
import React from "react";

function App() {
  return (
    <div>
      <h1>Hola, mundo!</h1>
    </div>
  );
}

export default App;
```

Este código crea un componente React simple que muestra un encabezado con el texto “Hola, mundo!”.

Asegúrate de que el archivo `src/index.js` tenga la siguiente forma:

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

reportWebVitals();
```

Finalmente, para ejecutar el código, abre una terminal en la carpeta del proyecto y ejecuta el siguiente comando:

```
npm start
```

Esto iniciará un servidor de desarrollo y abrirá automáticamente tu aplicación en un navegador web. Deberías ver el mensaje “Hola Mundo” en la página.

3.3. La Extensión .jsx

La extensión de archivo .jsx es ampliamente reconocida en el mundo de la programación web y se asocia principalmente con el desarrollo de aplicaciones front-end utilizando React, una popular biblioteca de JavaScript. En esta sección, exploraremos en detalle qué es .jsx y cómo se utiliza en el contexto de React.

.jsx es una extensión de archivo utilizada para archivos de JavaScript extendido (JavaScript XML). Se utiliza para escribir componentes de interfaz de usuario en React de una manera más declarativa y fácil de entender. A diferencia de JavaScript puro, .jsx permite la incorporación de elementos HTML y XML directamente en el código JavaScript.

La utilización de .jsx ofrece varias ventajas:

- **Declaratividad:** .jsx permite definir la estructura de la interfaz de usuario de manera más clara y declarativa, lo que facilita la comprensión del código.
- **Componentización:** Con .jsx, puedes crear componentes reutilizables que encapsulan la lógica y la interfaz de usuario, lo que facilita la modularidad y el mantenimiento del código.
- **Integración con React:** .jsx es la forma preferida de escribir componentes en React. React utiliza .jsx para representar visualmente la interfaz de usuario y manipularla de manera eficiente.

Para comprender mejor cómo .jsx simplifica el desarrollo en React, consideremos el siguiente ejemplo de un componente React que permite a los usuarios ingresar un mensaje personalizado y lo muestra en la pantalla en tiempo real:

```
import React, { useState } from "react";

function MessageGenerator() {
  // Estado para almacenar el valor del input
  const [inputValue, setInputValue] = useState("");

  // Funciin para manejar cambios en el input
  const handleInputChange = (e) => {
    setInputValue(e.target.value);
  };
```

```
11
12    return (
13      <div>
14        <h1>Generador de Mensajes</h1>
15        <p>Ingresa un mensaje personalizado:</p>
16        <input
17          type="text"
18          value={inputValue}
19          onChange={handleInputChange}
20          placeholder="Escribe tu mensaje aqui"
21        />
22        {/* Mensaje generado a partir del input */}
23        {inputValue && <p>Tu mensaje: {inputValue}</p>}
24      </div>
25    );
26  }
27
28  export default MessageGenerator;
```

En este ejemplo, hemos creado un componente llamado “MessageGenerator”. Los usuarios pueden ingresar un mensaje en un campo de texto, y el mensaje se muestra en la pantalla en tiempo real a medida que escriben. Esto es posible gracias a .jsx, que permite la interacción del usuario y la actualización dinámica de la interfaz de usuario en respuesta a los cambios en el estado.

Este ejemplo ilustra cómo .jsx en React simplifica la creación de componentes interactivos y cómo se pueden utilizar elementos HTML directamente en el código de JavaScript para crear aplicaciones web modernas y atractivas.

Vamos a desglosar el código del ejemplo “MessageGenerator” y explicarlo con más detalle:

```
1  import React, { useState } from "react";
```

1. Importamos las bibliotecas necesarias. “React” se utiliza para crear componentes de React, y “useState” es un gancho (hook) de React que permite agregar estado a los componentes funcionales.

```
1  function MessageGenerator() {
```

2. Definimos un nuevo componente React llamado “MessageGenerator”. Este componente será una función de JavaScript que renderizará elementos en la interfaz de usuario.

```
1  % Estado para almacenar el valor del input
2  const [inputValue, setInputValue] = useState("");
```

3. Declaramos un estado utilizando “useState”. “inputValue” será una variable de estado que almacenará el valor del campo de entrada de texto. “setInputValue” es una función que utilizaremos para actualizar el estado “inputValue”. Inicialmente, “inputValue” se establece en una cadena vacía (“”).

```
% Funcion para manejar cambios en el input
const handleInputChange = (e) => {
  setInputValue(e.target.value);
};
```

4. Definimos una función llamada “handleInputChange” que se ejecutará cada vez que el usuario escriba algo en el campo de entrada de texto. Esta función toma el evento “e” como argumento y utiliza “setInputValue” para actualizar el estado “inputValue” con el valor del campo de entrada de texto (“e.target.value”). En resumen, esta función actualiza dinámicamente el estado “inputValue” a medida que el usuario escribe en el campo.

```
  return (
    div>
      h1>Generador de Mensajes</h1>
      p>Ingresa un mensaje personalizado:</p>
      input
        type="text"
        value={inputValue}
        onChange={handleInputChange}
        placeholder="Escribe tu mensaje aqui"
      />
      % Mensaje generado a partir del input
      {inputValue && <p>Tu mensaje: {inputValue}</p>\}
    \div>
  );
}
```

4. Fundamentos de React

React es una biblioteca de JavaScript utilizada para crear interfaces de usuario interactivas y dinámicas. Antes de profundizar en características más avanzadas, es crucial establecer una base sólida en los conceptos fundamentales de React.

La documentación oficial de React es un recurso invaluable para comenzar a aprender React. Asegúrese de visitar el sitio web de React (Documentación oficial) y sumergirse en la documentación. Esta documentación proporciona una introducción completa a los conceptos esenciales, la arquitectura y las mejores prácticas de React.

4.1. Componentes en React:

Los componentes son la piedra angular de React. Son bloques de construcción reutilizables que permiten crear interfaces de usuario. En React, hay dos tipos principales de componentes: funcionales y de clase.

Componentes Funcionales

Los componentes funcionales son funciones de JavaScript que devuelven elementos JSX. Son más simples y concisos que los componentes de clase. Se pueden utilizar para componentes que no necesitan estado o métodos de ciclo de vida.

```
import React from 'react';

function MiComponente(props) {
  return (
    <div>
      <h1>{props.titulo}</h1>
      <p>{props.descripcion}</p>
    </div>
  );
}
```

Componentes de Clase

Los componentes de clase son clases de JavaScript que extienden la clase React.Component. Se utilizan cuando se necesita estado interno o métodos de ciclo de vida.

```
import React from 'react';
```

```
2
3  class MiComponente extends Component {
4    render() {
5      return (
6        <div>
7          <h1>{this.props.titulo}</h1>
8          <p>{this.props.descripcion}</p>
9        </div>
10     );
11   }
12 }
```

Props (Propiedades)

Las props (abreviatura de propiedades) son objetos que permiten pasar datos de un componente padre a un componente hijo. Son inmutables, lo que significa que no pueden ser modificadas por el componente hijo.

Ejemplo de uso de props en un componente funcional:

```
1  function Saludo(props) {
2    return <h1>Hola, {props.nombre}</h1>;
3  }
4
5  // Uso del componente con props
6  <Saludo nombre="Juan" />
```

Ejemplo de uso de props en un componente de clase:

```
1  class Saludo extends React.Component {
2    render() {
3      return <h1>Hola, {this.props.nombre}</h1>;
4    }
5  }
6
7  // Uso del componente con props
8  <Saludo nombre="Maria" />
```

En estos ejemplos, nombre es una prop que se pasa al componente Saludo. El componente utiliza esta prop para mostrar un mensaje personalizado.

1.4 Estado y Ciclo de Vida de los Componentes:

Explore cómo gestionar el estado local en componentes y cómo se actualiza. Comprenda el ciclo de vida de los componentes de React y cuándo se ejecutan los métodos importantes como componentDidMount y componentDidUpdate. 1.5 JSX (JavaScript XML):

Profundice en JSX, que se utiliza para definir la estructura de la interfaz de usuario en React. Aprenda cómo JSX se compila en JavaScript. 1.6 Manejo de Eventos:

Descubra cómo manejar eventos en React, como clics de botones o cambios de formulario. Aprenda a actualizar el estado o ejecutar funciones en respuesta a eventos. 1.7 Condicionales y Listas:

Aprenda cómo realizar renderizado condicional en React para mostrar contenido basado en condiciones. Comprenda cómo mapear listas de datos en elementos JSX. 1.8 Formularios en React:

Explore cómo trabajar con formularios en React y cómo manejar la entrada del usuario. Aprenda sobre el estado de los componentes de formulario y la actualización de datos. 1.9 Comunicación entre Componentes:

Comprenda cómo los componentes padres pueden pasar datos y funciones a los componentes hijos a través de "props". Explore cómo los componentes pueden comunicarse entre sí utilizando el "levantamiento de estado.º el uso de Context API. 1.10 Pruebas Unitarias en React (Opcional):

Si está interesado en pruebas, aprenda a escribir pruebas unitarias para sus componentes React utilizando bibliotecas como Jest y React Testing Library.

Paso 2: Enrutamiento en React

2.1 React Router: Estudie React Router (https://reactrouter.com/) para aprender a gestionar el enrutamiento en aplicaciones de una sola página (SPA). Paso 3: Gestión del Estado

3.1 Estado en React: Comprenda cómo gestionar el estado en React utilizando el estado local de componentes.

3.2 Redux o Context API: Explore bibliotecas como Redux o la API de Context para administrar el estado global en aplicaciones más grandes.

Paso 4: Estilos en React

4.1 CSS en JavaScript: Aprenda a aplicar estilos a componentes utilizando bibliotecas como Styled Components o Emotion. Paso 5: Pruebas en React

5.1 Jest y React Testing Library: Aprenda a escribir pruebas unitarias y de integración para sus componentes React utilizando Jest y React Testing Library. Paso 6: Aplicaciones del Mundo Real

6.1 Construcción de proyectos pequeños: Empiece a construir proyectos pequeños utilizando React para aplicar sus conocimientos.

6.2 Contribución a proyectos de código abierto: Considere contribuir a proyectos de código abierto basados en React para obtener experiencia práctica.

Paso 7: Avanzar en React

7.1 Hooks: Profundice en los Hooks de React, como useState y useEffect, para mejorar su manejo del estado y los efectos secundarios.

7.2 Suspense y Lazy Loading: Explore las características más avanzadas de React, como Suspense y Lazy Loading, para optimizar el rendimiento de su aplicación.

Paso 8: Recursos Adicionales

8.1 Lectura recomendada: Considere la lectura de libros como "Learning React"de Alex Banks y Eve Porcello, y "Pro React"de Cassio de Sousa Antonio.

8.2 Comunidad y redes sociales: Únase a la comunidad de React en redes sociales, foros y grupos de desarrollo para mantenerse actualizado y hacer preguntas.

Paso 9: Práctica Continua

9.1 Construcción de proyectos más grandes: A medida que adquiera más experiencia, trabaje en proyectos más grandes y desafiantes para mejorar sus habilidades.

9.2 Portafolio personal: Cree un portafolio personal en línea para mostrar sus proyectos de React.

Paso 10: Mantenerse Actualizado

10.1 Actualizaciones de React: Manténgase al día con las actualizaciones y las nuevas características de React a medida que la biblioteca evoluciona.

5. Patrones de diseño en React

Los patrones de diseño son soluciones probadas para problemas comunes en el desarrollo de software. En el contexto de React, existen varios patrones de diseño que se utilizan para estructurar y organizar aplicaciones React de manera eficiente y sostenible.

5.1. Reducer

Reducer es un patron de diseño que se utiliza para manejar el estado de una aplicación. En el contexto de React, un reducer es una función que toma el estado actual y una acción, y devuelve un nuevo estado. Los reducers son comunes en aplicaciones basadas en Redux, una biblioteca de gestión de estado para aplicaciones JavaScript o Typescript.

La idea principal detrás de los reducers es que permiten manejar el estado de una manera predecible

y escalable. Al dividir el estado en piezas más pequeñas y manejar las actualizaciones a través de acciones, los reducers pueden simplificar la lógica de la aplicación y facilitar el mantenimiento.

Para nuestra Arquitetura tendremos 5 modulos principales:

- **Models** (Modelos): Contiene la definición de los datos, un modelo en este caso es un objeto que representa una entidad de la aplicación, como un usuario, un producto, una publicación, etc. No necesitan pruebas unitarias.
- **Services:** (Servicios): Contiene servicios transparentes con el backend, estos servicios se encargan de realizar peticiones HTTP, usualmene no contienen lógica de negocio, y tampoco utilidades. No necesitan pruebas unitarias.
- **Context:** (Contexto): Contiene el contexto de la aplicación, es decir, el estado global de la aplicación, y las acciones que pueden modificar ese estado. Necesitan pruebas unitarias, pero es puramente funcional (.ts o .js).
- **Reducers:** (Reductores): Contiene los reductores, que son funciones puras que toman el estado actual y una acción, y devuelven un nuevo estado. Necesitan pruebas unitarias, pero es puramente funcional (.ts o .js).
- **UI:** (Interfaz de Usuario): Contiene los componentes de la interfaz de usuario, estos componentes se encargan de mostrar la información y de manejar la interacción del usuario. Necesitan pruebas unitarias, en este caso no es puramente funcional (.tsx o .jsx).
- **App:** (Aplicación): Contiene el punto de entrada de la aplicación, y se encarga de inicializar el contexto y renderizar la interfaz de usuario.

El patrón de diseño de reductor es especialmente útil para manejar el estado global de una aplicación, ya que permite dividir la lógica de la aplicación en piezas más pequeñas y manejar las actualizaciones de estado de manera predecible y escalable.

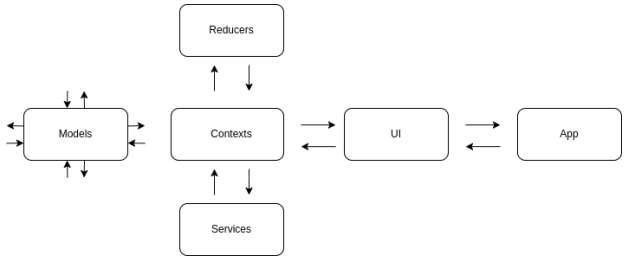

Figura 1: Diagrama de flujo de la arquitectura con el patrón de diseño de Reducer

En este diagrama se muestra el flujo de la arquitectura con el patrón de diseño de Reducer. La aplicación comienza en el punto de entrada, donde se inicializa el contexto y se renderiza la interfaz de usuario. Los componentes de la interfaz de usuario interactúan con el contexto para leer y actualizar el estado global de la aplicación. Los reductores manejan las actualizaciones de estado en respuesta a las acciones, y los servicios se encargan de realizar peticiones al backend. Los modelos definen la estructura de los datos que se utilizan en la aplicación.