nacap

Framework basado en JavaScript



Tabla de CONTENIDOS

Evolución de los *frameworks* de JavaScript en el desarrollo web moderno
Introducción..... 03
Frameworks..... 03
React..... 04
Vue.js..... 05
Angular..... 06
Uso actual de los *frameworks* de JavaScript.... 07
Recursos complementarios..... 08

Guía interactiva: elementos esenciales de los *frameworks* de JavaScript
Introducción..... 09
React: entorno de desarrollo..... 09
Componentes de React..... 10
Propiedades (props)..... 10
Estado *(State)*..... 10
Hooks..... 11
JSX..... 12
Recursos complementarios..... 12

Guía práctica: codificación de componentes en JavaScript con componentes en React
Introducción..... 13
Componentes en React..... 13
Tipos de componentes..... 13
Creación de componentes funcionales..... 14
Props (propiedades)..... 14
Estado *(state)*..... 15
Manejo de eventos..... 15
Ciclo de vida de los componentes..... 16
Recursos complementarios..... 17

Guía práctica: implementación de *CRUD* con local *storage* en JavaScript
Introducción..... 17
¿Qué es local *storage*?..... 17
Configuración del proyecto..... 18
Recursos complementarios..... 22

Guía práctica: acceso a datos externos con *fetch* y *axios* en JavaScript
¿Qué es una *API* (interfaz de programación de aplicaciones)?..... 23
Introducción a fetch..... 25
Introducción a *axios*..... 26
Comparación entre *fetch* y *axios*, incluyendo un ejemplo..... 27
Recursos complementarios..... 29

Evolución de los *frameworks* de JavaScript en el desarrollo web moderno

Introducción

En este recurso exploraremos los elementos esenciales de los *frameworks* de JavaScript a través de una guía interactiva. Nos enfocaremos en los componentes clave, características y ejemplos prácticos de algunos de los *frameworks* más populares, como React, Vue.js y Angular. Esta guía ayudará a comprender cómo estos *frameworks* facilitan el desarrollo de aplicaciones web modernas.

Nos adentraremos en el mundo de los *frameworks* de JavaScript, esenciales para el desarrollo de aplicaciones web modernas. A través de una guía interactiva, exploraremos los elementos clave de los *frameworks* de JavaScript, proporcionando una comprensión profunda de sus componentes, características y aplicaciones prácticas.

Incluiremos recursos adicionales como enlaces a documentación oficial, tutoriales en video y foros de discusión donde podrás resolver dudas y compartir experiencias con otros desarrolladores. Esta guía completa te permitirá adquirir las habilidades necesarias para desenvolverte con confianza en el desarrollo de aplicaciones web modernas utilizando los *frameworks* de JavaScript más populares.

Frameworks

Un *framework* de JavaScript es una biblioteca de código que proporciona una estructura predefinida para desarrollar aplicaciones web.

A continuación, tienes las ventajas de utilizar los *frameworks*:

1. Facilita el desarrollo y mantenimiento de aplicaciones web.
2. Proporciona herramientas y funcionalidades listas para usar.
3. Mejora la organización del código y la productividad del desarrollador.





React

Es una biblioteca de JavaScript de código abierto, desarrollada y mantenida por Facebook, utilizada para construir interfaces de usuario. React se destaca por su enfoque en la creación de componentes reutilizables, lo que permite a los desarrolladores construir aplicaciones web dinámicas y altamente interactivas de manera más eficiente y estructurada. Ahora, conocerás las características principales de React:

Componentes: corresponde a bloques de construcción reutilizables que encapsulan lógica y presentación.

JSX: es la sintaxis que permite escribir HTML dentro de JavaScript.

Estado y props: son el manejo del estado local del componente y paso de datos entre componentes.

Para darle profundidad a tu aprendizaje, conocerás dos ejemplos de código:

Ejemplo componente básico

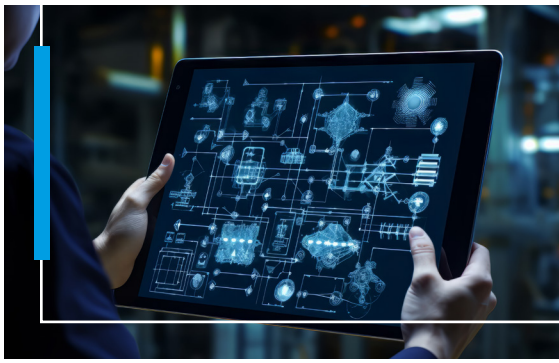
```
import React from 'react';
import ReactDOM from 'react-dom';
 function App() {
  return <h1>¡Hola, mundo!</h1>;
}

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

Ejemplo manejo del estado con *hooks*

```
import React, { useState } from 'react';
function Contador() {
 const [contador, setContador] = useState(0);
   return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}
export default Contador;
```

Vue.js



Es un framework progresivo de JavaScript para la construcción de interfaces de usuario. Fue creado por Evan You y es conocido por su simplicidad y flexibilidad, permitiendo a los desarrolladores adoptar gradualmente sus características según las necesidades del proyecto.

Ahora conocerás las características principales de *Vue.js*:

Componentes: es una estructura modular que facilita la reutilización del código.

Directivas: son instrucciones especiales en el HTML que se enlazan con el DOM.

Reactividad: es el sistema que actualiza automáticamente el DOM cuando cambian los datos.

Para darle profundidad a tu aprendizaje, conocerás dos ejemplos de código:

Ejemplo componente básico

```
<div id=”app”>
  <h1>{{ mensaje }}</h1>
</div>
<script>
 var app = new Vue({
    el: ‘#app’,
    data: {
      mensaje: ‘¡Hola, mundo!’
    }
  });
</script>
```

Ejemplo manejo del estado

```
<div id=”contador”>
  <p>Contador: {{ contador }}</p>
  <button @click=”incrementar”>Incrementar</button>
</div>
<script>
  var contador = new Vue({
    el: ‘#contador’,
    data: {
      contador: 0
    },
    methods: {
      incrementar: function() {
        this.contador++;
      }
    }
  });
</script>
```



Angular

Es un *framework* de desarrollo web de código abierto mantenido por Google, diseñado para facilitar la construcción de aplicaciones web robustas y de gran escala. Angular proporciona una plataforma completa con un conjunto de herramientas y bibliotecas que permiten desarrollar aplicaciones dinámicas de una sola página *(SPA)* con un rendimiento y escalabilidad óptimos.

Ahora conocerás las características principales de angular:

Componentes: son unidades independientes de la interfaz de usuario.

Templates: *HTML* con y directivas que pbindingsermiten mostrar datos y manejar eventos.

Inyección de dependencias: es un sistema que permite gestionar servicios y dependencias de manera eficiente.

Para darle profundidad a tu aprendizaje, conocerás dos ejemplos de código:

Ejemplo componente básico

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: '<h1>{{ mensaje }}</h1>',
})
export class AppComponent {
  mensaje = '¡Hola, mundo!';
}
```

Ejemplo manejo del estado

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-contador',
  template: `
    <div>
      <p>Contador: {{ contador }}</p>
      <button (click)="incrementar()">Incrementar</button>
    </div>
  `,
})
export class ContadorComponent {
  contador = 0;

  incrementar() {
    this.contador++;
  }
}
```

Uso actual de los *frameworks* de JavaScript

React sigue siendo uno de los *frameworks* más populares debido a su flexibilidad y gran ecosistema de herramientas y bibliotecas. En la actualidad, existen varias empresas que utilizan *framewoks* para programar sus códigos, React por ejemplo es utilizado por:

Facebook: React fue desarrollado por Facebook y se utiliza ampliamente en sus aplicaciones.

Instagram: Utiliza React para su interfaz de usuario interactiva y rápida.

Netflix: Usa React para su alto rendimiento y capacidad de modularidad.

Vue.js ha ganado popularidad por su enfoque progresivo y facilidad de integración en proyectos existentes. En la actualidad, existen varias empresas que utilizan *framewoks* para programar sus códigos, *Vue.js* por ejemplo es utilizado por:

Alibaba: utiliza *Vue.js* para su plataforma de comercio electrónico.

Xiaomi: emplea *Vue.js* en diversas aplicaciones y sitios web.

GitLab: usa *Vue.js* para su interfaz de usuario.

Angular sigue siendo popular en entornos empresariales debido a su robustez y características completas para el desarrollo de aplicaciones web a gran escala. En la actualidad, existen varias empresas que utilizan *framewoks* para programar sus códigos, angular por ejemplo es utilizado por:

Google: angular es desarrollado y mantenido por Google, y se utiliza en muchas de sus aplicaciones internas.

Microsoft: utiliza Angular en varias de sus aplicaciones empresariales.

IBM: emplea Angular para desarrollar aplicaciones empresariales robustas.

Recursos complementarios

A continuación, tienes material complementario para que soportes tu conocimiento:

- Documentación oficial de React: https://reactjs.org/docs/getting-started.html
- Documentación oficial de Vue.js: https://vuejs.org/v2/guide/
- Documentación oficial de Angular: https://angular.io/docs
- Tutorial de React en español: https://es.reactjs.org/tutorial/tutorial.html
- Tutorial interactivo Vue.js: https://vuejs.org/v2/guide/#Declarative-Rendering

Hemos explorado los elementos esenciales de los *frameworks* de JavaScript mediante una guía interactiva. Hemos aprendido sobre las características y ventajas de React, *Vue.js* y Angular, y cómo estos *frameworks* son utilizados por grandes empresas en 2024.

Estos conocimientos son fundamentales para entender cómo los *frameworks* de JavaScript facilitan el desarrollo de aplicaciones web modernas y su impacto en la industria actual.

Guía interactiva: elementos esenciales de los *frameworks* de JavaScript

Introducción

En esta semana, exploraremos los elementos fundamentales de un *framework* de JavaScript, utilizando React como ejemplo. React es una biblioteca de JavaScript para construir interfaces de usuario interactivas. Este taller práctico proporcionará una comprensión básica de cómo trabajar con componentes, estados y propiedades en React.

Esta guía completa te permitirá adquirir las habilidades necesarias para desenvolverte con confianza en el desarrollo de aplicaciones web modernas utilizando los *frameworks* de JavaScript más populares.

React: entorno de desarrollo



Para comenzar a trabajar con React utilizando Vite, es necesario configurar el entorno de desarrollo. Vite es un entorno de desarrollo rápido y ligero que permite crear aplicaciones React de manera eficiente.

Instalar Node.js y npm

- Asegúrate de tener Node.js y npm instalados en tu máquina.
- Puedes descárgalo desde nodejs.org

Crear un nuevo proyecto React con Vite

Abre tu terminal y ejecuta el siguiente comando:

```
npm create vite@latest mi-app --template react
```

Navega al directorio del proyecto:

```
cd mi-app
```

Instalar dependencias

Ejecuta el siguiente comando para instalar las dependencias necesarias:

```
npm install
```

Iniciar el servidor de desarrollo

Ejecuta el siguiente comando para iniciar el servidor de desarrollo:

```
npm run dev
```

La aplicación estará disponible en http://localhost:3000

Componentes de React

En este taller, te enfocarás en los componentes funcionales, los cuales son la base de React junto con los componentes de clase. Se explorarán en profundidad las características y ventajas de los componentes funcionales, así como su uso en el desarrollo de aplicaciones.

Además, aprenderás a utilizar *hooks*, como useState y useEffect, para manejar el estado y los efectos en los componentes funcionales. También verás cómo pasar propiedades a los componentes funcionales y cómo trabajar con el contexto en React.

Durante el taller, realizarás ejercicios prácticos para que puedas aplicar lo aprendido y adquirir experiencia en la creación de componentes funcionales en React. Al final del taller, estarás preparado para utilizar de manera efectiva los componentes funcionales en tus proyectos de desarrollo web.

Propiedades (props)

Los props son argumentos que se pasan a los componentes con el fin de personalizar su funcionamiento y apariencia. Es importante tener en cuenta que:

Los props son de solo lectura, lo que significa que no se pueden modificar dentro del componente.



Esto garantiza la consistencia y la integridad de los datos que se pasan a través de los props, asegurando un flujo de información eficiente y seguro en la aplicación.

Estado *(State)*

Es un objeto que representa la información local del componente. El estado puede cambiar con el tiempo y afectar la renderización del componente. En componentes funcionales, se maneja usando el *hook* useState. El estado es esencial en la programación de componentes ya que:

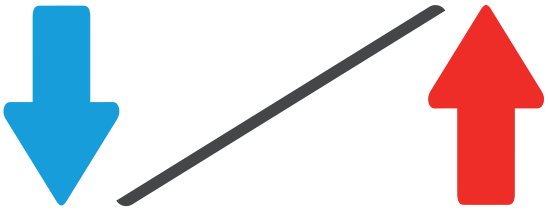
Representa la información local y puede cambiar con el tiempo, lo cual afecta directamente la renderización del componente.

En el caso de los componentes funcionales, el estado se gestiona a través del hook useState, permitiendo así la actualización dinámica de la información y la interacción fluida con el usuario.

Hooks

Son funciones especiales de React que permiten usar características de React, como el estado y el ciclo de vida, en componentes funcionales. El hook useState de React permite:

agregar y gestionar estado dentro de un componente funcional, lo que nos da la posibilidad de actualizar y mostrar información dinámica en nuestra aplicación.



realizar acciones secundarias en componentes funcionales, como, por ejemplo, gestionar el ciclo de vida de un componente. Esto nos brinda la posibilidad de llevar a cabo operaciones como peticiones a servidores, suscripciones a eventos, actualizaciones de estado, entre otras tareas, de manera más controlada y eficiente en nuestra aplicación.

A continuación, conocerás los principales hooks utilizados:

useState: es utilizada para el manejo de estado en componentes funcionales.

useEffect: es utilizada para el manejo de efectos secundarios (similar a los métodos de ciclo de vida en componentes de clase).

useContext: es utilizada para el manejo de contexto para pasar datos a través del árbol de componentes sin tener que pasar props manualmente en cada nivel.

useReducer: es utilizada para el manejo de estado complejo mediante un reductor (similar a Redux).



JSX

Es una poderosa extensión de JavaScript que permite a los desarrolladores escribir código de una manera que se asemeja a *HTML*, pero dentro de JavaScript.

Esta sintaxis facilita la creación de interfaces de usuario de forma más intuitiva y legible, ya que combina la familiaridad de *HTML* con la potencia y flexibilidad de JavaScript.

Al utilizar JSX en un proyecto de React, esta sintaxis se transforma en llamadas a la función React.createElement mediante herramientas como Babel. Esto permite a los desarrolladores:

Crear componentes de forma eficiente y dinámica, lo que resulta en una experiencia de usuario más fluida y atractiva.

Gracias a JSX, la creación de interfaces de usuario en aplicaciones web se vuelve más sencilla y eficaz, permitiendo a los programadores concentrarse en la lógica de la aplicación en lugar de preocuparse por la manipulación del DOM.

Recursos complementarios

A continuación, tienes algunos enlaces de referencia de material complementario sobre React:

- Documentación oficial de React: https://reactjs.org/
- React Hooks Cheat Sheet: https://react-hooks-cheatsheet.com/

Hemos explorado los elementos fundamentales de React, un framework de JavaScript. Hemos aprendido a configurar un entorno de trabajo utilizando Vite, y hemos introducido conceptos clave como componentes, props, estado y hooks. Estos conceptos son esenciales para el desarrollo de aplicaciones web modernas y dinámicas.

Guía práctica: codificación de componentes en JavaScript con componentes en React

Introducción

En esta semana, explorarás los elementos fundamentales de un *framework* de JavaScript, utilizando React como ejemplo. React es una biblioteca de JavaScript para construir interfaces de usuario interactivas. Este conocimiento práctico te proporcionará una comprensión básica de cómo trabajar con componentes, estados y propiedades en React.

Explorarás en detalle el desarrollo de componentes en React, desde la creación de componentes simples hasta la composición de componentes más complejos. Aprenderás a utilizar props y *state* para pasar datos entre componentes, a manejar eventos y a trabajar con el ciclo de vida de los componentes.

Además, profundizarás en la importancia de la reutilización de componentes y en las buenas prácticas para su diseño y organización. Al finalizar esta semana, serás capaz de **crear interfaces de usuario robustas y escalables utilizando React** y sus poderosas herramientas de desarrollo de componentes.

Componentes en React



Los componentes en React son unidades independientes y reutilizables que encapsulan la lógica y la presentación de una parte de la interfaz de usuario. Pueden ser de dos tipos: componentes funcionales y de clase.

Tipos de componentes

En React, los componentes son las unidades básicas de construcción de la interfaz de usuario. Existen varios tipos de componentes, cada uno con sus propias características y casos de uso específicos. A continuación, se describen los principales tipos de componentes en React:

Componentes funcionales: son funciones de JavaScript que retornan elementos React (normalmente escritos en JSX) y aceptan props como argumento.

Componentes de clase: son clases de ES6 que extienden de React.Component y tienen un método render que retorna elementos React.

Para profundizar en los componentes de React, revisa la documentación oficial en el siguiente enlace: https://es.reactjs.org/docs/components-and-props.html



Creación de componentes funcionales

Los **componentes funcionales son funciones simples** que **retornan JSX**. Son la forma más común de **definir componentes en React**, especialmente desde la **introducción de *hooks***.

En este ejemplo, saludo es un componente funcional que recibe props y retorna un encabezado con un saludo. Aquí tienes el código que se utiliza:

```
import React from 'react';
function Saludo(props) {
  return <h1>Hola, {props.nombre}</h1>;
}
export default Saludo;
```

Props (propiedades)

Son **argumentos que se pasan a los componentes para personalizarlos**. Los props son de **solo lectura** y **no se pueden modificar** dentro del componente.

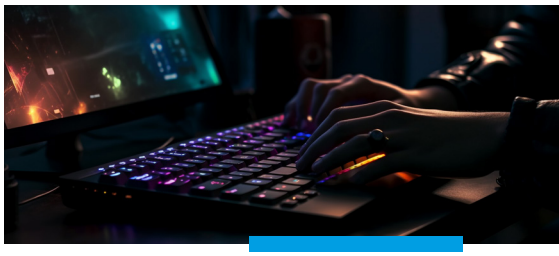


En este ejemplo, el componente Saludo recibe un prop llamado nombre y lo utiliza para personalizar el mensaje de saludo. A continuación, tienes el código que se utiliza:

```
import React from 'react';
function Saludo(props) {
  return <h1>Hola, {props.nombre}</h1>;
}
export default Saludo;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import Saludo from './Saludo';
ReactDOM.render(<Saludo nombre="Mundo" />, document.getElementById('root'));
```

Estado *(state)*



Es un objeto que representa la información local del componente. El estado puede cambiar con el tiempo y afectar la renderización del componente. En componentes funcionales, se maneja usando el *hook* useState.

En este ejemplo, contador es un componente funcional que utiliza el *hook* useState para manejar el estado local cuenta. Se inicializa con el valor 0. A continuación, tienes el código que se utiliza:

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

function Contador() {
  const [cuenta, setCuenta] = useState(0);

  return (
    <div>
      <p>Cuenta: {cuenta}</p>
      <button onClick={() => setCuenta(cuenta + 1)}>Incrementar</
button>
    </div>
  );
}

export default Contador;
```

Manejo de eventos

Los eventos en React se manejan de manera similar a los eventos en *HTML*, pero con algunas diferencias sintácticas.

En React, los eventos se nombran usando camelCase y se pasan como funciones.

En este ejemplo, el componente Botón cambia el mensaje mostrado cuando se hace clic en el botón. Se inicializa con el valor 0. A continuación, tienes el código que se utiliza:

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

function Boton() {
  const [mensaje, setMensaje] = useState('Haz clic en el botón');

  function manejarClic() {
    setMensaje('¡Botón clickeado!');
  }

  return (
    <div>
      <p>{mensaje}</p>
      <button onClick={manejarClic}>Clic aquí</button>
    </div>
  );
}

export default Boton;
```

Ciclo de vida de los componentes

El ciclo de vida de un componente se refiere a las fases por las que pasa desde su creación hasta su eliminación.

En componentes de clase, se manejan con métodos específicos como componentDidMount, componentDidUpdate y componentWillUnmount.

En componentes funcionales, se utiliza el hook useEffect para manejar los efectos secundarios y el ciclo de vida. En este ejemplo, useEffect actualiza el título del documento cada vez que cambia la cuenta. A continuación, tienes el código que se utiliza:

```
import React, { useState, useEffect } from 'react';

function ContadorConEfecto() {
  const [cuenta, setCuenta] = useState(0);

  useEffect(() => {
    document.title = `Cuenta: ${cuenta}`;
  }, [cuenta]);

  return (
    <div>
      <p>Cuenta: {cuenta}</p>
      <button onClick={() => setCuenta(cuenta + 1)}>Incrementar</
button>
    </div>
  );
}

export default ContadorConEfecto;
```

Recursos complementarios

A continuación, tienes algunos enlaces de referencia de material complementario sobre React:

- React. (s. f.). Componentes y propiedades. https://es.reactjs.org/docs/components-and-props.html
- React. (s. f.). Tutorial: tic, tac, toe. https://reactjs.org/tutorial/tutorial.html

Has profundizado en el desarrollo de componentes en React, abordando la creación de componentes funcionales, el uso de props y estado, el manejo de eventos y el ciclo de vida de los componentes. Estos conceptos son esenciales para construir aplicaciones React modulares y dinámicas.

Guía práctica: implementación de *CRUD* con local *storage* en JavaScript

Introducción

En este recurso, aprenderás a construir una aplicación CRUD (crear, leer, actualizar, eliminar) utilizando local storage en React. Local storage es una API de almacenamiento web que permite almacenar datos de manera persistente en el navegador del usuario.

Es ideal para aplicaciones que requieren almacenamiento de datos en el lado del cliente sin necesidad de una base de datos en el servidor.



¿Qué es local *storage*?

Es una API de almacenamiento web que permite almacenar datos en el navegador de manera persistente y sin fecha de expiración. Los datos almacenados en *local storage* están disponibles incluso después de cerrar el navegador.

Local storage es una interfaz de programación de aplicaciones *(API)* que proporciona la capacidad de:

Almacenar datos de forma persistente en el navegador web. Estos datos se guardan sin una fecha de expiración y permanecen accesibles incluso después de cerrar la ventana del navegador.

Gracias a *Local storage*, los desarrolladores pueden almacenar información importante de sus aplicaciones web para su posterior acceso y utilización.

Local storage tiene algunas características principales, las cuales conocerás a continuación:

- Almacenamiento clave-valor.
- Tamaño máximo de aproximadamente 5MB por dominio.
- Acceso sincrónico.

Ahora que ya conoces las características principales, aquí tienes las operaciones básicas:

localStorage.setItem('clave', 'valor'): Almacenar un valor.	localStorage.getItem('clave'): Recuperar un valor.
localStorage.removeItem('clave'): Eliminar un valor.	localStorage.clear(): Limpiar todo el almacenamiento.

Configuración del proyecto

Primero, necesitas configurar un nuevo proyecto de React utilizando Vite. A continuación, conocerás los pasas que necesitas.

Crear un nuevo proyecto

Abre tu terminal y ejecuta el siguiente comando:

```
npm create vite@latest mi-crud --template react
```

Navega al directorio del proyecto:

```
cd mi-crud
```

Instalar dependencias

Ejecuta el siguiente comando para instalar las dependencias necesarias:

```
npm install
```

Iniciar el servidor de desarrollo

Ejecuta el siguiente comando para iniciar el servidor de desarrollo:

```
npm run dev
```

La aplicación estará disponible en http://localhost:3000.

Estructura del proyecto

La estructura básica del proyecto será la siguiente:

```
mi-crud/
 ├──── node_modules/
 ├──── public/
 ├──── src/
 │       ├──── components/
 │       │       ├──── Form.js
 │       │       ├──── List.js
 │       │       └──── Item.js
 │       ├──── App.js
 │       ├──── index.css
 │       └──── main.jsx
 ├──── .gitignore
 ├──── index.html
 ├──── package.json
 └──── vite.config.js
```

Creación de componentes Form

El componente Form se utilizará para crear y actualizar elementos.

```
import React, { useState, useEffect } from 'react';

function Form({ addOrUpdateItem, itemToEdit }) {
  const [inputValue, setInputValue] = useState('');

  useEffect(() => {
    if (itemToEdit) {
      setInputValue(itemToEdit.value);
    } else {
      setInputValue('');
    }
  }, [itemToEdit]);
const handleSubmit = (e) => {
    e.preventDefault();
    if (inputValue.trim()) {
      addOrUpdateItem(inputValue);
      setInputValue('');
    }
  };
```

```
return (
   <form onSubmit={handleSubmit}>
     <input
       type=”text”
       value={inputValue}
       onChange={(e) => setInputValue(e.target.value)}
     />
     <button type=”submit”>{itemToEdit ? ‘Actualizar’ : ‘Agregar’}</
button>
   </form>
 );
}

export default Form;
```

Creación de componentes List

El componente List mostrará los elementos almacenados y permitirá eliminarlos o seleccionarlos para edición.

```
Import React from ‘react’;
import Item from ‘./Item’;

function List({ items, deleteItem, editItem }) {
 return (
   <ul>
     {items.map((item) => (
       <Item
         key={item.id}
         item={item}
         deleteItem={deleteItem}
         editItem={editItem}
       />
     ))}
   </ul>
 );
}

export default List;
```

Componentes Item

El componente Item representará un solo elemento de la lista.

```
import React from 'react';

function Item({ item, deleteItem, editItem }) {
  return (
    <li>
      {item.value}
      <button onClick={() => editItem(item)}>Editar</button>
      <button onClick={() => deleteItem(item.id)}>Eliminar</button>
    </li>
  );
}

export default Item;
```

Manejo de *local storage*

Utiliza *Local Storage* para almacenar y recuperar los elementos de la lista.

```
import React, { useState, useEffect } from 'react';
import Form from './components/Form';
import List from './components/List';
import './App.css';

function App() {
  const [items, setItems] = useState([]);
  const [itemToEdit, setItemToEdit] = useState(null);

  useEffect(() => {
    const storedItems = JSON.parse(localStorage.getItem('items')) ||
[];
    setItems(storedItems);
  }, []);
useEffect(() => {
    localStorage.setItem('items', JSON.stringify(items));
  }, [items]);
const addOrUpdateItem = (value) => {
    if (itemToEdit) {
      setItems(items.map(item => item.id === itemToEdit.id ? { ...item,
value } : item));
      setItemToEdit(null);
    } else {
      setItems([...items, { id: Date.now(), value }]);
    }
  };

  const deleteItem = (id) => {
    setItems(items.filter(item => item.id !== id));
  };
```

```
 const editItem = (item) => {
   setItemToEdit(item);
 };
  return (
   <div className="App">
     <h1>CRUD con LocalStorage</h1>
     <Form addOrUpdateItem={addOrUpdateItem} itemToEdit={i-
temToEdit} />
     <List items={items} deleteItem={deleteItem} editItem={editItem}
/>
   </div>
 );
}

export default App;
```

Ejecución de pruebas

Inicia la aplicación y prueba las operaciones *CRUD*:

Crear: agrega nuevos elementos utilizando el formulario.
Leer: visualiza los elementos en la lista.
Actualizar: edita un elemento seleccionándolo y actualízalo en el formulario.
Eliminar: elimina elementos de la lista.

Recursos complementarios

A continuación, tienes un ejemplo de ejercicio completo en React y la aplicacion de listado de cosas por hacer.

- Beltrand, P. (2022, 20 de junio). FRONTEND - Aplicación TODO List con React. https://www.youtube.com/watch?v=1A0fpwrSDAo

De igual manera tienes referencias complementarias para profundizar en tu conocimiento:

- mdn web docs. (s. f.). Window: localStorage property. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- React. (s. f.). File structure. https://reactjs.org/docs/faq-structure.html
- React. (s. f.). Getting started. https://reactjs.org/docs/getting-started.html

Has construido una aplicación CRUD completa utilizando React y LocalStorage, además, has aprendido a manejar el almacenamiento en el lado del cliente y a crear componentes reutilizables para realizar operaciones CRUD. Estos conceptos son esenciales para el desarrollo de aplicaciones web modernas y persistentes.

Guía práctica: acceso a datos externos con *fetch* y *axios* en JavaScript

En este recurso, explorarás los elementos esenciales de los *frameworks* de JavaScript a través de una guía interactiva. Te enfocarás en los **componentes clave, las características y los ejemplos prácticos** de algunos de los *frameworks* más populares, como React, Vue.js y Angular. Esta guía te ayudará a comprender cómo estos f*rameworks* facilitan el desarrollo de aplicaciones web modernas.

En esta semana, aprenderás a **acceder a datos externos utilizando las *API* Fetch y Axios en React**. Las *API* (interfaz de programación de aplicaciones) permiten que diferentes aplicaciones se comuniquen entre sí, lo que es esencial para **integrar datos externos** en nuestras aplicaciones React. Esta capacidad es crucial para **construir aplicaciones dinámicas** y con **datos en tiempo real, mejorando la interacción del usuario** y la funcionalidad general de la aplicación.

Además, profundizarás en la a**rquitectura basada en componentes de React**, la reactividad, el sistema de plantillas de Vue.js, la inyección de dependencias y el enrutamiento avanzado en Angular.

Al final, serás capaz de **implementar componentes reutilizables, manejar estados y eventos, y conectar tus aplicaciones** a **fuentes de datos externas**, utilizando las **mejores prácticas y patrones de diseño** en cada uno de estos *frameworks*.

¿Qué es una *API* (interfaz de programación de aplicaciones)?



Una *API (application programming interface)* es esencialmente un **puente de comunicación entre dos aplicaciones**, proporcionando un **conjunto de definiciones y protocolos** que facilitan la interacción entre ellas.

Gracias a las *API*, es posible **acceder a datos y funcionalidades de distintos servicios y aplicaciones** de manera eficiente y segura.

Las *API* permiten que **diferentes aplicaciones interactúen entre sí** de manera fluida, sin necesidad de que los desarrolladores tengan que **conocer los detalles internos** de cada una de ellas. Esto permite a los programadores:

Crear nuevas funcionalidades combinando servicios existentes, lo que ahorra tiempo y recursos en el desarrollo de *software*.

Además, las *API* también facilitan la integración de diferentes plataformas y sistemas, lo que resulta fundamental en el mundo actual, donde la interoperabilidad entre aplicaciones es clave. En resumen, las *API* son herramientas fundamentales en el desarrollo de *software* moderno, ya que simplifican la comunicación entre aplicaciones y permiten la creación de soluciones más completas y eficientes.

Las *API*, o *interfaces de programación de aplicaciones*, son un **conjunto de protocolos y herramientas** que permiten que **diferentes aplicaciones se comuniquen entre sí** de manera eficiente. Existen **varios tipos de *API***, cada una diseñada para **cumplir funciones específicas y facilitar la integración de servicios** y datos externos en nuestras aplicaciones. Entre las más comunes se encuentran:

REST APIs: se utilizan HTTP requests para acceder y manipular datos. Los métodos comunes incluyen GET, POST, PUT, DELETE.

SOAP APIs: se utilizan XML para transferir datos y son más estrictas en términos de estructura.

Ahora que ya conoces los tipos de API, mira un ejemplo de *API* REST:

JSONPlaceholder

Es un servicio gratuito que proporciona datos de prueba para desarrollar y probar aplicaciones.

Este servicio ofrece endpoints que devuelven datos ficticios de usuarios, posts, comentarios, álbumes, fotos, y más.

Es una herramienta muy útil para aquellos desarrolladores que necesitan datos de ejemplo para su trabajo, ya que pueden hacer solicitudes a la API y recibir respuestas con información simulada.

JSONPlaceholder es una excelente opción para realizar pruebas y experimentar con diferentes tipos de datos sin tener que preocuparse por la integridad de la información.

Además, al ser gratuito, es accesible para cualquier persona que quiera utilizarlo en sus proyectos de desarrollo.

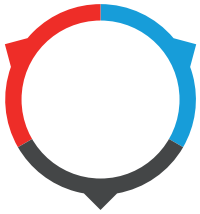
Introducción a fetch

Fetch es una ***API* nativa de JavaScript para realizar solicitudes *HTTP***.

Es una **alternativa moderna a XMLHttpRequest**.

Fetch es una *API* nativa de JavaScript que proporciona una manera sencilla y potente de realizar solicitudes *HTTP* asíncronas desde el navegador. Sus principales características incluyen:

Proporciona una sintaxis más limpia y legible.



Está basada en promesas.

Simplifica la realización de solicitudes HTTP.

Ahora que ya conoces las características, te suministraré la sintaxis básica de código:

Instalación de *axios*

```
npm install axios
```

Sintaxis básica de *axios*

```
import axios from 'axios';

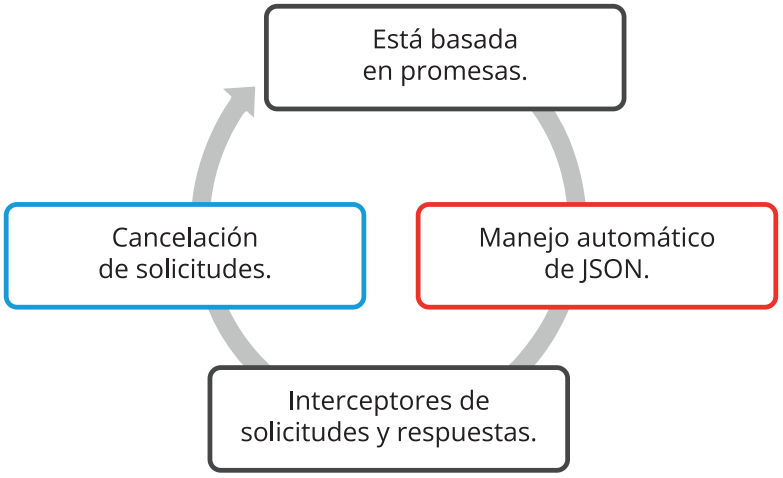
axios.get('https://api.example.com/data')
  .then(response => console.log(response.data))
  .catch(error => console.error('Error:', error));
```

Introducción a *axios*

Axios es una **biblioteca de terceros que simplifica las solicitudes *HTTP***.

Es muy popular debido a su **simplicidad y capacidad para manejar solicitudes más complejas**.

Axios es una biblioteca de JavaScript ampliamente utilizada para realizar solicitudes *HTTP*, que destaca por su facilidad de uso y características avanzadas. Sus principales características incluyen:



Ahora que ya conoces las características, te suministraré el código de la instalación y la sintaxis básica de código:

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

Comparación entre *fetch* y *axios*, incluyendo un ejemplo

Al comparar *Fetch* y *Axios* es importante destacar las fortalezas y diferencias clave de cada

Comparación entre fetch y axios		
Característica	***Fetch***	***Axios***
Soporte nativo	Si	No
Basado en promesas	Si	Si
Manejo de JSON	Necesita response. json()	Automático
Intercepción de solicitudes	No	Si
Cancelación de solicitudes	No	Si
Compatibilidad	Todos los navegadores modernos	Todos los navegadores modernos

herramienta, para comprender mejor cuándo utilizar una sobre la otra. En la siguiente tabla encontrarás sus características:

En el siguiente ejemplo, construirás una aplicación React que obtiene datos de una *API* externa utilizando *fetch*

Configurar el proyecto

Crea un nuevo proyecto de React con Vite, para ello requieres el siguiente código:

```
npm create vite@latest fetch-example --template react
cd fetch-example
npm install
npm run dev
```

Crear componentes

Crea el componente App.js, para ello requieres el siguiente código:

```
import React, { useState, useEffect } from 'react';

function App() {
  const [data, setData] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch('https://jsonplaceholder.typicode.com/posts')
      .then(response => response.json())
      .then(data => {
        setData(data);
        setLoading(false);
      })
      .catch(error => {
        console.error('Error fetching data:', error);
        setError(error);
        setLoading(false);
      });
  }, []);
if (loading) return <p>Loading...</p>;
  if (error) return <p>Error fetching data.</p>;

  return (
    <div>
      <h1>Posts</h1>
      <ul>
        {data.map(post => (
          <li key={post.id}>{post.title}</li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

Crea el componente main.jsx, para ello requieres el siguiente código:

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

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

Ejecutar la aplicación

Inicia el servidor de desarrollo y abre http://localhost:3000 en tu navegador, para ver la lista de *posts* obtenidos de la *API*. Si necesitas más detalles o ajustes en este contenido, por favor realiza preguntas en la sesión semanal.

Recursos complementarios

A continuación, tienes referencias complementarias para profundizar en tu conocimiento:

- Beltrand, P. (2022, 12 de julio). Como consumir una API utilizando React [video de Youtube]. https://www.youtube.com/watch?-v=lJbbqXq1AiU

Listado de *APIs* públicas

- Guthub. (s. f.). public-apis. https://github.com/public-apis/public-apis

Has aprendido a acceder a datos externos utilizando fetch y axios en React. También has explorado las características de las API, comparado fetch y axios, y construido una aplicación práctica para obtener datos de una API utilizando fetch.

Estos conocimientos son esenciales para integrar datos externos en las aplicaciones web y mejorar la funcionalidad de las mismas.

Bibliografía

Angular. (s. f.). *Introduction to the Angular docs*. https://angular.io/docs

Axios. (s. f.). *Axios*. https://axios-http.com/

React. (s. f.). *Getting started*. https://reactjs.org/docs/getting-started.html

React. (s. f.). *React*. https://reactjs.org/

Vue.js. (2023). *Introduction*. https://vuejs.org/v2/guide/