El hook **`useSelector`** se utiliza en React-Redux para extraer datos del store. Permite seleccionar parte del estado almacenado en el store de Redux y suscribirse a las actualizaciones de ese estado en el componente de React.

Los Hooks en React son funciones especiales que te permiten usar características de React en componentes funcionales. Los Hooks más comunes son **`useState`** para manejar el estado y **`useEffect`** para trabajar con el ciclo de vida en componentes funcionales.

El componente **`<Provider>`** de 'react-redux' permite que los componentes accedan al store. Este componente envuelve la aplicación React y proporciona el contexto necesario para que los componentes puedan acceder al store de Redux.

En 'styled-components', puedes utilizar la interpolación **`${}`** para incluir expresiones JavaScript dentro de la definición de estilos. En este caso, **`props.active`** se utiliza para aplicar el color de fondo verde si **`props.active`** es verdadero y rojo en caso contrario.

Las props (propiedades) en React son los mecanismos a través de los cuales los datos se pasan de un componente padre a un componente hijo. Permiten la comunicación y transferencia de información entre componentes en la jerarquía de React.

Usar un objeto para agrupar todos los valores de input.Al utilizar un objeto para agrupar los valores de los inputs, puedes mantener un solo estado y acceder a los diferentes valores mediante propiedades del objeto. Esto puede ser especialmente útil cuando tienes varios campos de entrada relacionados.

En un componente controlado en React, se suele utilizar la convención de nombrar la función que maneja el envío del formulario como **`handleSubmit()`**. Esto es solo una convención y el nombre exacto puede variar según la preferencia del desarrollador, pero **`handleSubmit()`** es comúnmente utilizado.

. La clase **`btn`** se utiliza para estilizar un elemento como un botón, y la clase **`btn-primary`** se utiliza para aplicar estilos específicos de Bootstrap para un botón primario.

Cuando estás utilizando **`styled-components`**, la forma correcta de definir un botón con un color de fondo rojo es utilizando el método **`styled.button`** y aplicar estilos dentro de los backticks (``). La opción E representa esta sintaxis de manera correcta.

En React Router, la notación **`/:id`** en una ruta indica un parámetro de ruta llamado "id", que puede ser capturado dinámicamente desde la URL. Por ejemplo, si tienes una ruta como "/users/:id", podrías acceder al valor dinámico de "id" en el componente correspondiente a esa ruta.

El hook **`useDispatch`** en React-Redux se utiliza para obtener una referencia a la función **`dispatch`** de la tienda Redux. Puedes usar **`dispatch`** para despachar acciones y así actualizar el estado en la tienda. En resumen, **`useDispatch`** te proporciona una manera de despachar acciones y modificar el estado del store.

El hook **`useDispatch`** en 'react-redux' se utiliza para obtener una referencia a la función **`dispatch`** de la tienda Redux. Puedes usar esta función **`dispatch`** para despachar acciones y así actualizar el estado en la tienda.

El componente **`<BrowserRouter>`** en 'react-router-dom' envuelve tu aplicación y proporciona la funcionalidad de enrutamiento para los componentes que están dentro de él.

Para un componente controlado en React, es común utilizar la convención de nombrar la función que maneja el envío del formulario como **`handleSubmit()`**

En Bootstrap, un "container" es una clase que se utiliza para envolver y contener el contenido de una página o sección. Este contenedor ayuda a centrar y alinear el contenido dentro de un diseño predefinido para mantener una apariencia consistente en toda la aplicació

En un evento de React, **`e.target`** se refiere al elemento DOM que ha desencadenado el evento. Cuando se maneja un evento en React, como un clic o un cambio, el objeto de evento (a menudo denotado como **`e`** o **`event`**) contiene información sobre el evento, y **`e.target`** proporciona una referencia al elemento DOM específico que fue el objetivo del evento.

Por ejemplo, en un evento de clic, **`e.target`** sería el elemento DOM que fue clicado. En un evento de cambio en un campo de entrada, **`e.target`** sería el elemento de entrada que ha cambiado.

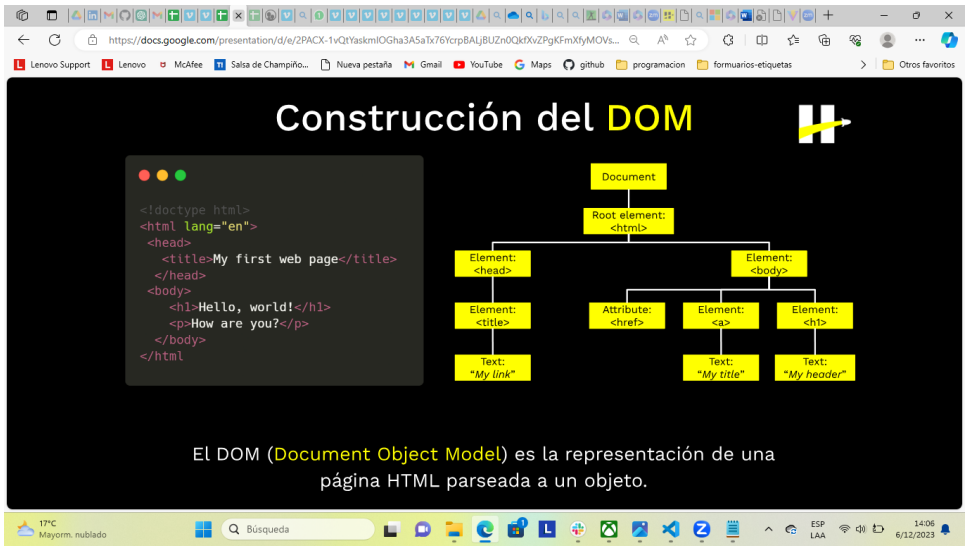
Esta propiedad es útil para realizar acciones específicas basadas en el elemento que causó el evento. Puedes acceder a sus propiedades, como **`value`** para un campo de entrada o **`innerText`** para un elemento de texto, para obtener o modificar su contenido.

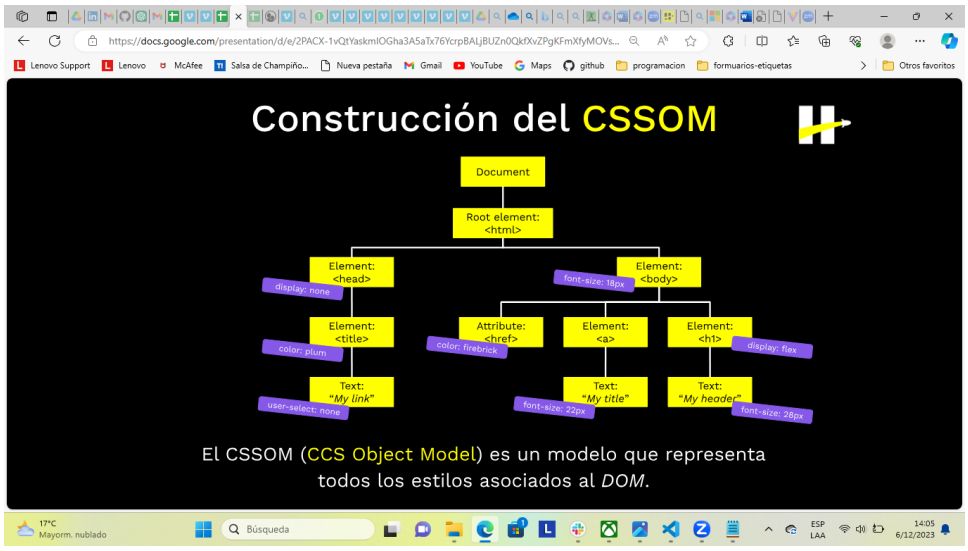
Un componente controlado en React se refiere a un componente cuyo estado es gestionado de manera explícita a través de los props y funciones proporcionadas por React. En un componente controlado, los datos del estado (como el valor de un campo de entrada) están vinculados al estado de React y se actualizan mediante funciones manejadoras.

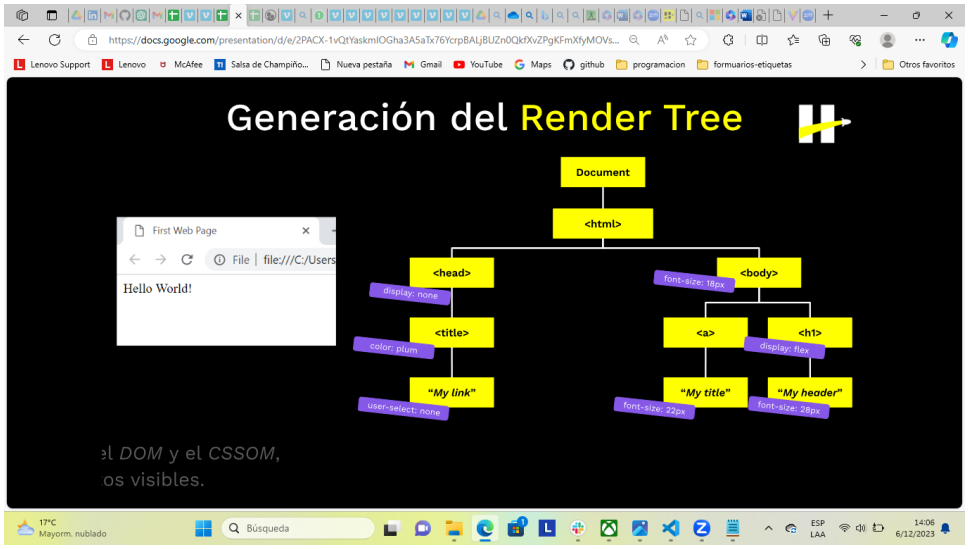
En un componente controlado, los elementos de formulario, como campos de entrada (**`<input>`**) o áreas de texto (**`<textarea>`**), no mantienen su propio estado interno. En cambio, su valor es controlado por el estado de React y se actualiza mediante una función de cambio (**`onChange`**) proporcionada.

Un componente controlado en React se refiere a un componente cuyo estado es gestionado de manera explícita a través de los props y funciones proporcionadas por React. En un componente controlado, los datos del estado (como el valor de un campo de entrada) están vinculados al estado de React y se actualizan mediante funciones manejadoras.

En un componente controlado, los elementos de formulario, como campos de entrada (**<input>**) o áreas de texto (**<textarea>**), no mantienen su propio estado interno. En cambio, su valor es controlado por el estado de React y se actualiza mediante una función de cambio (**onChange**) proporcionada.



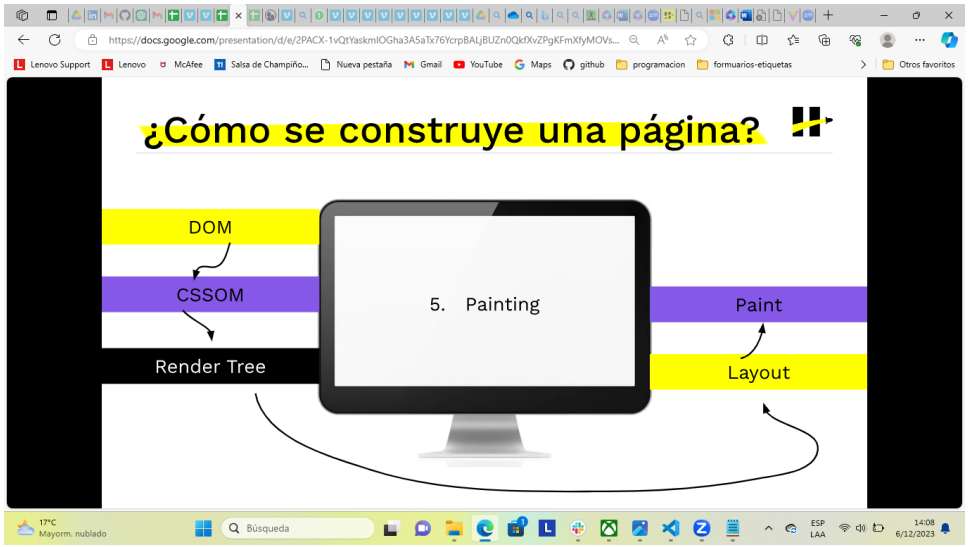




jsx

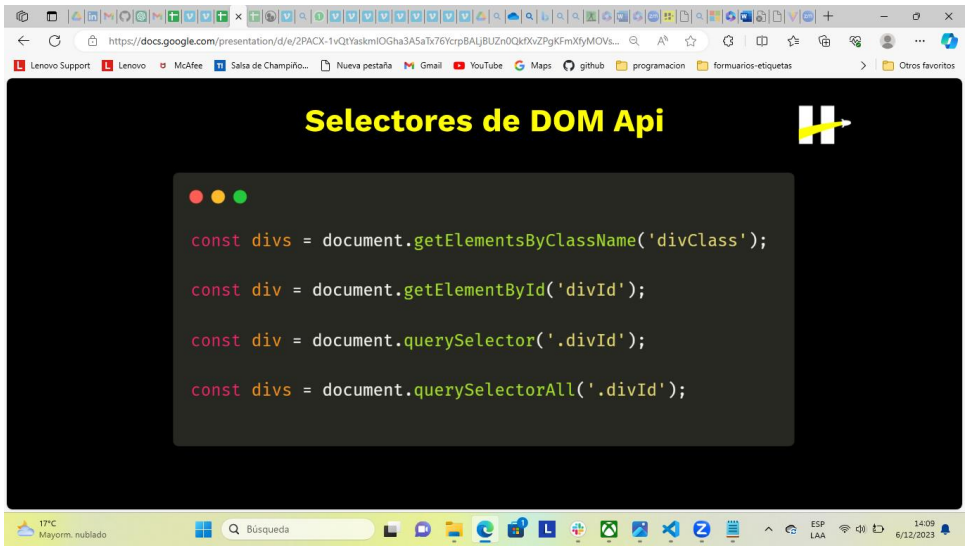
El término "layout" en el contexto del desarrollo web se refiere a la disposición y organización visual de los elementos en una página o interfaz. El diseño o layout determina cómo se distribuyen y se presentan los componentes, como texto, imágenes, formularios, etc., en el espacio disponible.

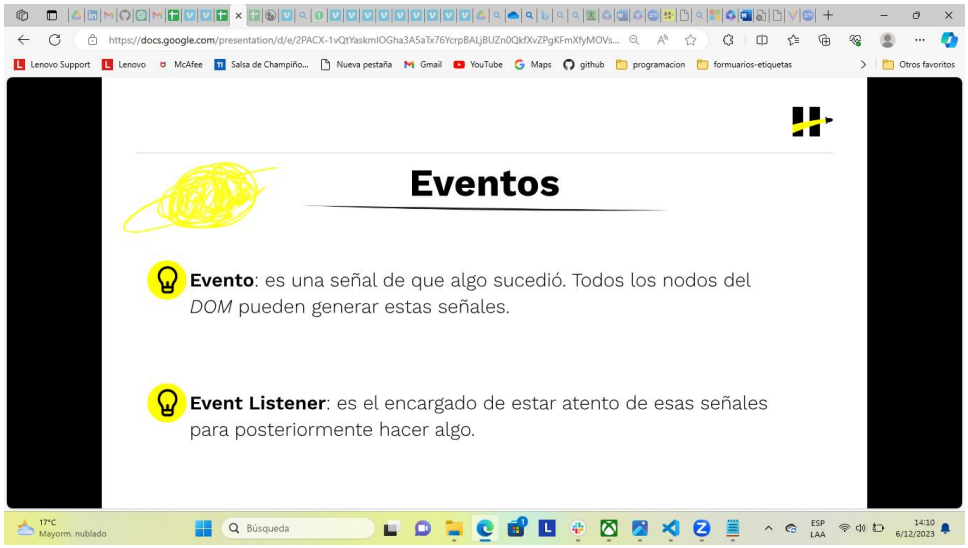
Un buen layout es esencial para proporcionar una experiencia de usuario agradable y efectiva

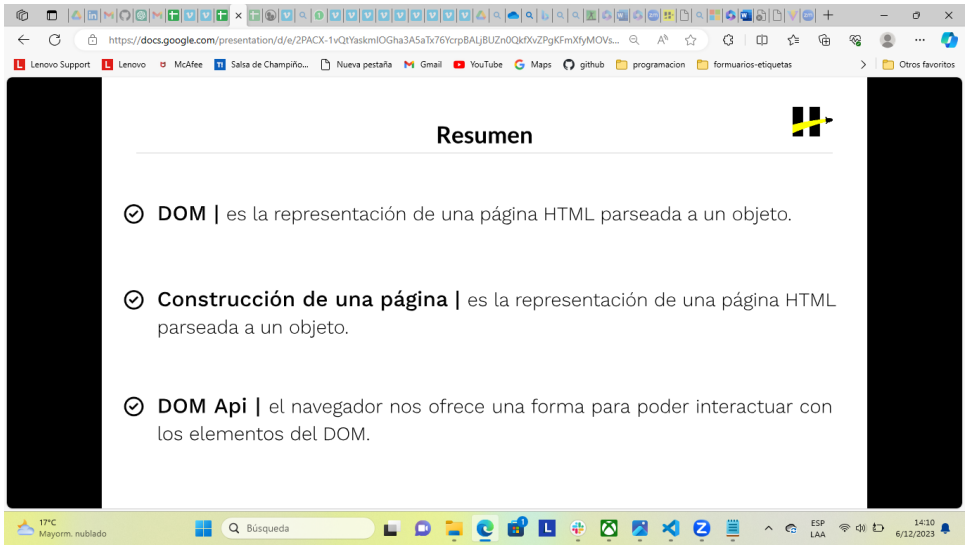


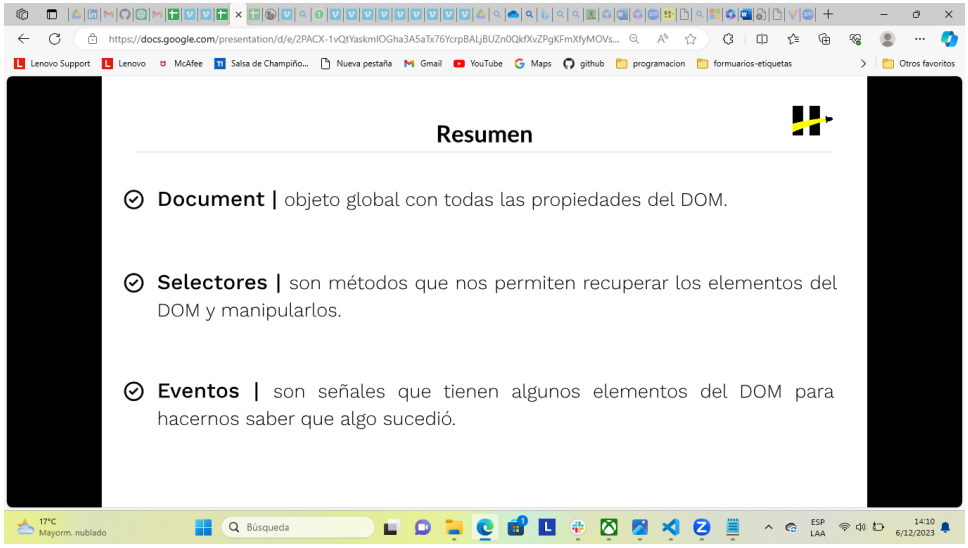
El DOM API, o Interfaz de Programación de Aplicaciones del Documento Object Model (Document Object Model Application Programming Interface), es una interfaz que permite a los programas y scripts acceder y manipular la estructura, contenido y estilo de documentos HTML y XML. El DOM representa el documento como una estructura de árbol de objetos, donde cada nodo del árbol representa una parte del documento, como elementos HTML, atributos, texto, etc.

La DOM API proporciona un conjunto de métodos y propiedades que permiten a los desarrolladores web interactuar con el contenido de un documento.









REACT

React es una biblioteca de JavaScript utilizada para construir interfaces de usuario (UI) interactivas y reutilizables. Aquí hay algunos conceptos y funcionamientos básicos de React:

- **Componentes:** En React, una aplicación se construye a partir de componentes. Un componente es una pieza reutilizable de código que encapsula la lógica y la interfaz de usuario de una parte específica de la aplicación. Los componentes pueden ser simples o complejos y se pueden anidar unos dentro de otros.

```
// Ejemplo de componente funcional

function MiComponente() {

  return <p>Hola, soy un componente de React.</p>;
```

} **JSX:** JSX (JavaScript XML) es una extensión de JavaScript que permite escribir código HTML dentro de archivos JavaScript. Facilita la creación de estructuras de interfaz de usuario de manera declarativa en lugar de imperativa.

```
const elemento = <h1>Hola, mundo!</h1>;
```

Estado (State): El estado es un objeto que contiene datos relevantes para un componente y determina su comportamiento y apariencia. Los componentes pueden tener un estado interno que se actualiza en respuesta a eventos o cambios en la aplicación

```
const [contador, setContador] = useState(0);
```

Props (Propiedades): Las props son datos que se pasan de un componente principal a un componente secundario. Permiten la comunicación entre componentes y la configuración de su comportamiento.

```
function Saludo(props) {
  return <p>Hola, {props.nombre}!</p>;
}
```

Renderización condicional: Puedes renderizar diferentes partes de la interfaz de usuario en función del estado o las props

```
function Saludo(props) {
  return props.usuario ? <p>Hola, {props.usuario}!</p> : <p>Hola, invitado!</p>
```

Ciclo de vida del componente: Los componentes de React tienen un ciclo de vida que comprende fases como montaje, actualización y desmontaje. Los métodos de ciclo de vida permiten ejecutar lógica en diferentes etapas del ciclo de vida del componente

```
class MiComponente extends React.Component {
  componentDidMount() {
    console.log('El componente se ha montado.');
  }
  componentWillUnmount() {
    console.log('El componente se va a desmontar.');
  }
  render() {
    return <p>Hola, soy un componente de clase.</p>;
  }
}
```

Hooks: Los hooks son funciones especiales que permiten usar el estado y otras características de React en componentes funcionales. **`useState`** y **`useEffect`** son ejemplos comunes de hooks.

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


function MiComponente() {
  const [contador, setContador] = useState(0);


  useEffect(() => {
    document.title = `Contador: ${contador}`;
  }, [contador])
  return (
    <div>
      <p>Contador: {contador}</p>
      <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
  );
}
```

las **hooks** son funciones especiales que te permiten usar el estado y otras características de React en componentes funcionales. Antes de la introducción de hooks, los componentes funcionales no podían tener su propio estado interno ni aprovechar características como el ciclo de vida del componente. Las hooks fueron introducidas en React 16.8 para abordar estas limitaciones y permitir un desarrollo más avanzado en componentes funcionales.

`react-redux` es una biblioteca que proporciona integración entre React y Redux, facilitando el uso de Redux en aplicaciones React. Redux es un contenedor de estado predecible para aplicaciones JavaScript, y **`react-redux`** ofrece herramientas para conectar los componentes de React al estado gestionado por Redux.

Las características principales de **`react-redux`** incluyen:

- **Provider:** **`Provider`** es un componente de **`react-redux`** que permite que la aplicación React acceda al store de Redux. Este componente debe envolver la aplicación en el nivel superior para que todos los componentes tengan acceso al store.

- **connect:** `connect` es una función de `react-redux` que conecta componentes de React al store de Redux. Permite que los componentes accedan al estado y despachen acciones sin necesidad de pasar estas propiedades manualmente.
- **useDispatch y useSelector:** Estos son hooks proporcionados por `react-redux` que permiten acceder a `dispatch` y seleccionar partes del estado sin necesidad de conectar componentes.
- **Hooks de React-Redux avanzados:** Además de `useDispatch` y `useSelector`, `react-redux` proporciona otros hooks avanzados como `useStore`, `useSelector` y `useDispatch`.