Módulo 4
Desarrollo de interfaces interactivas con React

Manejo del DOM

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

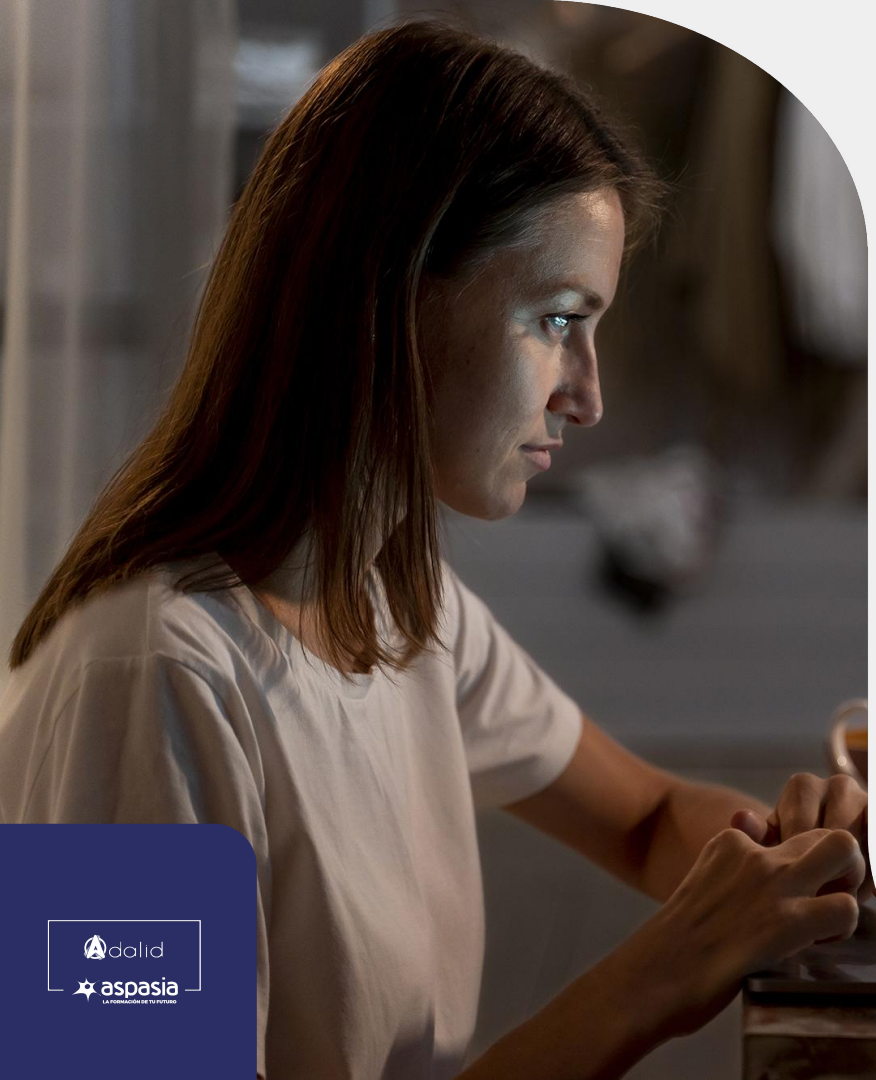
Módulo 4

AE 3.1

OBJETIVOS

Aprende a manejar el DOM en ReactJS: Virtual DOM, React Fiber, referencias, React-DOM (cliente/servidor), diferencias en atributos HTML y herramientas de pruebas.

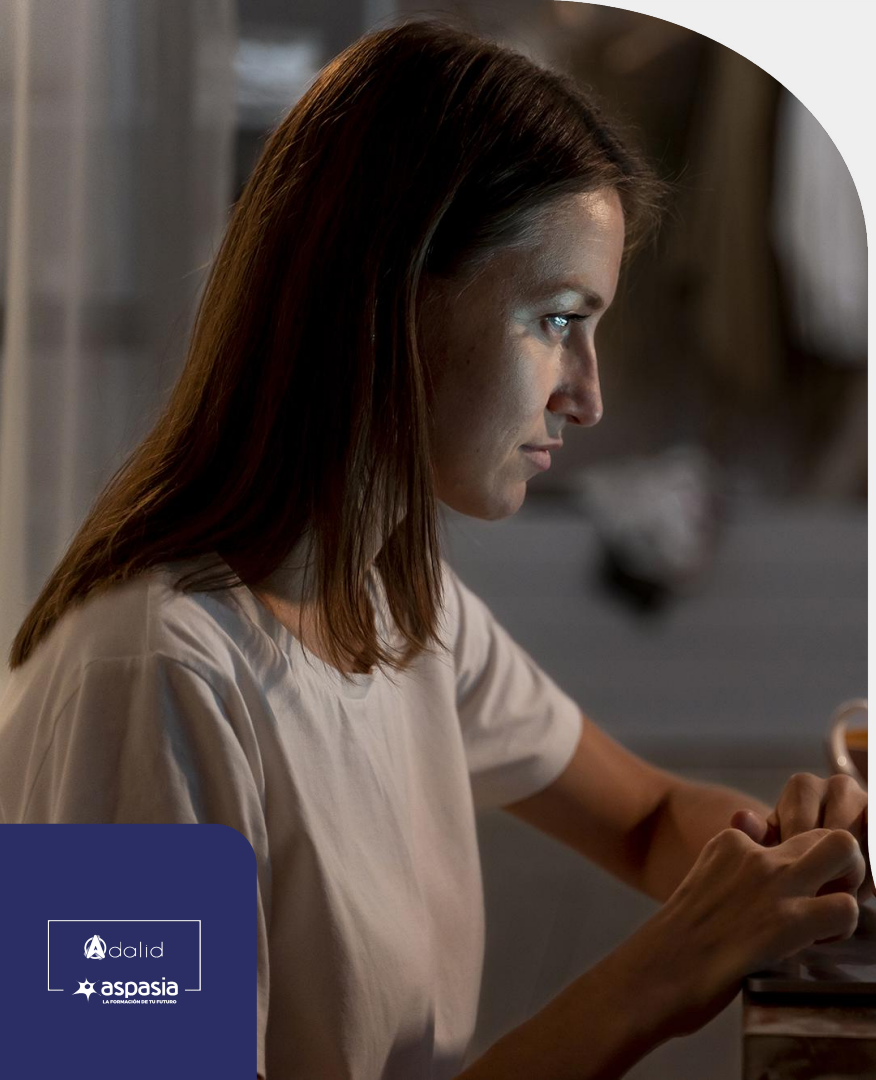
Chile
dalid
aspasia
LA FORMACIÓN DE TU FUTURO



¿QUÉ VAMOS A VER?

- Manejo del DOM.
- Elementos DOM en ReactJs.
- Acerca de React.Component.
- El DOM Virtual en ReactJs.
  - Qué es el DOM Virtual.
  - Qué es el Shadow DOM.
  - Qué es React Fiber.
- Uso de referencias.
  - Qué son las referencias.
  - Cuándo usar referencias.
  - Creación de referencias.
  - Accediendo a las referencias.
  - Agregando referencias al DOM.
  - Referencias mediante Callback.

adalid aspasia LA FORMACIÓN DE TU FUTURO

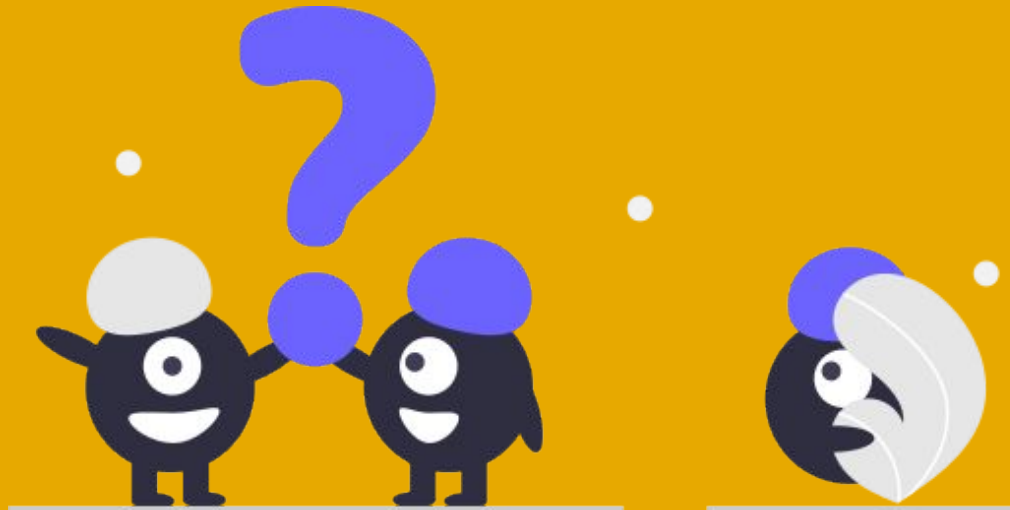


¿QUÉ VAMOS A VER?

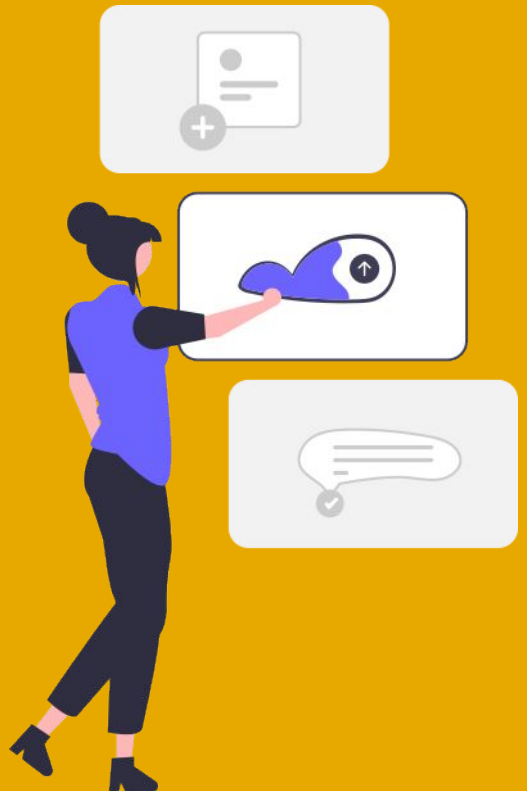
- El paquete React-DOM.
  - React DOM en el cliente.
  - React DOM en el servidor.
  - Soporte en navegadores.
- Diferencias en atributos HTML que funcionan diferente en ReactJs (checked, className, onChange, selected).
- Utilidades para pruebas.
- Renderizador de prueba.
- Requerimientos del entorno de JS.

Adalid aspasia LA FORMACIÓN DE TU FUTURO

¿Que es el DOM?



Adalid
aspasia
LA FORMACIÓN DE TU FUTURO



Manejo del DOM

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

Creación de Proyecto

Para este aprendizaje vamos a desarrollar una aplicación, cubriendo las temáticas mientras avanzamos en la construcción de nuestro proyecto.

Alista tu terminal 😁

Configuración Inicial del Proyecto

Busca tu carpeta de proyectos, crearemos una app de mantenimiento llamada **react-maintenance-app,** usaremos **Vite.**

Crear un proyecto React con Vite:

```
npm create vite@latest react-maintenance-app -- --template react
cd react-maintenance-app
npm install
npm run dev
```

npm run dev: Comando para iniciar el servidor de desarrollo.

```
react-maintenance-app/
├── node_modules/
├── public/
│   └── vite.svg
├── src/
│   ├── App.css
│   ├── App.jsx
│   ├── index.css
│   ├── main.jsx
│   └── assets/
│       └── react.svg
├── .gitignore
├── eslint.config.js
├── index.html
├── package-lock.json
├── README.md
└── vite.config.js
```

src/App.jsx: Archivo principal donde se define el componente raíz de la aplicación.

Vite.config.js: Configuración de Vite.

adalid aspasia LA FORMACIÓN DE TU FUTURO

Elementos DOM en ReactJS

En React, los elementos del DOM se manejan de **forma virtual** para optimizar actualizaciones. Esto significa que React no manipula directamente el DOM real, sino que **utiliza el DOM Virtual** como una representación ligera del DOM real.

Elementos DOM en ReactJS

Crea un componente para registrar un vehículo y enviarlo al "taller" en la ruta **src/components/RegisterVehicle.jsx**.

- Creamos un componente funcional llamado **RegisterVehicle**.
- Incluye un formulario con campos básicos y un evento **onSubmit** para registrar un vehículo.

```
function RegisterVehicle() {
  const handleSubmit = (e) => {
    e.preventDefault();
    alert('Vehículo registrado exitosamente');
  };

  return (
    <form onSubmit={handleSubmit}>
      <h2>Registrar Vehículo</h2>
      <label htmlFor="vehicle">Modelo del Vehículo:</label>
      <input type="text" id="vehicle" name="vehicle" placeholder="Ejemplo: Toyota
Corolla" />

      <label htmlFor="owner">Propietario:</label>
      <input type="text" id="owner" name="owner" placeholder="Nombre del
propietario" />

      <button type="submit">Registrar</button>
    </form>
  );
}

export default RegisterVehicle;
```

dalid aspasia LA FORMACIÓN DE TU FUTURO

Elementos DOM en ReactJS

```
import RegisterVehicle from './components/RegisterVehicle';

function App() {
  return (
    <div>
      <h1>Mantenimiento de Vehículos</h1>
      <RegisterVehicle />
    </div>
  );
}

export default App;
```

Modifica tu archivo **App.jsx** para incluir el nuevo componente en el proyecto, primero elimina lo que no necesitas. Recuerda que el archivo se encuentra en **src/App.jsx**.

Acerca de React.Component

En React, un componente es una **unidad reutilizable** que puede ser de clase o funcional. React.Component es la base para crear componentes de clase, aunque los funcionales son más comunes.

Acerca de React.Component

Crea un componente de **clase** no es tan común pero nos servirá para experimentar y mostrar una lista estática de servicios disponibles. **src/components/ServiceList.jsx**.

- Usa un componente de clase llamado **ServiceList**.
- Renderizamos una lista estática de servicios usando un método **map**.

```
import { Component } from 'react';

class ServiceList extends Component {
  render() {
    const services = ['Cambio de Aceite', 'Revisión de Frenos',
'Cambio de Llantas'];

    return (
      <div>
        <h2>Servicios Disponibles</h2>
        <ul>
          {services.map((service, index) => (
            <li key={index}>{service}</li>
          ))}
        </ul>
      </div>
    );
  }
}

export default ServiceList;
```

Adalid aspasia LA FORMACIÓN DE TU FUTURO

Acerca de React.Component

Agrega al archivo **App.jsx** el componente componente **ServiceList**. Recuerda que el archivo se encuentra en **src/App.jsx**.

```
import RegisterVehicle from './components/RegisterVehicle';
import ServiceList from './components/ServiceList';

function App() {
  return (
    <div>
      <h1>Mantenimiento de Vehículos</h1>
      <RegisterVehicle />
      <ServiceList />
    </div>
  );
}

export default App;
```

El DOM Virtual

El DOM virtual en React es un concepto clave para mejorar la **eficiencia** y el **rendimiento** de las aplicaciones web. Básicamente, es una **representación ligera y virtual** del DOM real del navegador que React utiliza internamente para realizar actualizaciones rápidas y eficientes.

El DOM Virtual

Shadow DOM

El Shadow DOM es una tecnología de encapsulación que permite crear componentes con estilos independientes.

React Fiber

React Fiber es el motor de renderización que React utiliza para priorizar tareas de manera más eficiente.

El DOM Virtual

Crea el componente **StatusMessage** el cual actualizar dinámicamente el estado de un mensaje en pantalla.
src/components/StatusMessage.jsx.

- Crea un componente funcional **StatusMessage**.
- Utilizamos **useState** para manejar el estado del mensaje dinámicamente.
- React actualiza solo el nodo necesario en el DOM Virtual.

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

function StatusMessage() {
  const [status, setStatus] = useState('Esperando
acción...');

  return (
    <div>
      <h2>Estado del Sistema</h2>
      <p>{status}</p>
      <button onClick={() => setStatus('Vehículo en proceso
de mantenimiento')}>
        Actualizar Estado
      </button>
    </div>
  );
}

export default StatusMessage;
```

Adalid aspasia LA FORMACIÓN DE TU FUTURO

El DOM Virtual

Agrega al archivo **App.jsx** el componente componente **StatusMessage**. Recuerda que el archivo se encuentra en **src/App.jsx**.

```
import RegisterVehicle from './components/RegisterVehicle';
import ServiceList from './components/ServiceList';
import StatusMessage from './components/StatusMessage';

function App() {
  return (
    <div>
      <h1>Mantenimiento de Vehículos</h1>
      <RegisterVehicle />
      <ServiceList />
      <StatusMessage />
    </div>
  );
}

export default App;
```

Uso de Referencias

En React, las referencias (Refs) permiten **acceder directamente a elementos del DOM** o a **instancias de componentes** creadas.

Son útiles para:

- Acceder a un elemento para enfocarlo.
- Manipular el DOM directamente.

Uso de Referencias

Crea el componente **FocusInput** que use referencias para enfocar un campo de texto automáticamente.
src/components/FocusInput.jsx.

- Usamos **useRef** para crear una referencia al input.
- Al hacer clic en el botón, llamamos a **inputRef.current.focus()**.

```
import { useRef } from 'react';

function FocusInput() {
  const inputRef = useRef(null);

  const handleFocus = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <h2>Enfocar Campo de Texto</h2>
      <input ref={inputRef} type="text" placeholder="Escribe algo aquí..." />
      <button onClick={handleFocus}>Enfocar</button>
    </div>
  );
}

export default FocusInput;
```

adalid aspasia LA FORMACIÓN DE TU FUTURO

Uso de Referencias

Agrega al archivo **App.jsx** el componente componente **FocusInput**. Recuerda que el archivo se encuentra en **src/App.jsx**.

```
import RegisterVehicle from './components/RegisterVehicle';
import ServiceList from './components/ServiceList';
import StatusMessage from './components/StatusMessage';
import FocusInput from './components/FocusInput';

function App() {
  return (
    <div>
      <h1>Mantenimiento de Vehículos</h1>
      <RegisterVehicle />
      <ServiceList />
      <StatusMessage />
      <FocusInput />
    </div>
  );
}

export default App;
```

adalid aspasia LA FORMACIÓN DE TU FUTURO

El Paquete React-DOM

El paquete react-dom es una biblioteca complementaria a React que se encarga de **interactuar con el DOM real del navegador** y **conectar los componentes de React a las interfaces del usuario**. Proporciona métodos para renderizar componentes en el cliente o en el servido

El Paquete React-DOM

React DOM en el cliente

En aplicaciones que se ejecutan en el navegador, React DOM en el cliente permite renderizar componentes React en elementos del DOM real.

Código de ejemplo

```
import ReactDOM from 'react-dom/client'; // Importar desde react-dom

import App from './App'; // Componente principal de tu aplicación

// Crear un "root" en el DOM real
const root = ReactDOM.createRoot(document.getElementById('root'));

// Renderizar la aplicación en el DOM real
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Adalid aspasia LA FORMACIÓN DE TU FUTURO

El Paquete React-DOM

React DOM en el servidor

React DOM en el servidor permite renderizar componentes React en el servidor y generar un HTML estático que se envía al cliente.

Código de ejemplo

```
import { renderToString } from 'react-dom/server';

import App from './App'; // Componente principal

const html = renderToString(<App />);
console.log(html); // Devuelve un HTML estático que
representa tu componente
```

El Paquete React-DOM

Usa el paquete **react-dom** para renderizar un nuevo componente independiente en un contenedor diferente al principal. Crea el componente **ExtraInfo.**

src/components/ExtraInfo.jsx

```
function ExtraInfo() {
  return (
    <div>
      <h2>Información Adicional</h2>
      <p>React-DOM permite renderizar componentes tanto en
cliente como en servidor.</p>
    </div>
  );
}

export default ExtraInfo;
```

Adalid aspasia LA FORMACIÓN DE TU FUTURO

El Paquete React-DOM

Crear un contenedor adicional en el body del archivo **index.html** justo debajo de la línea ** **:

```
<div id="extra-info"></div>
```

Agregar el código de renderización en **src/main.jsx** debe quedar así**:**

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./index.css";
import App from "./App.jsx";
import ExtraInfo from "./components/ExtraInfo.jsx";

createRoot(document.getElementById("root")).render(
  <StrictMode>
    <App />
  </StrictMode>
);

createRoot(document.getElementById("extra-info")).render(
  <StrictMode>
    <ExtraInfo />
  </StrictMode>
);
```

Diferencias en Atributos HTML

Los atributos en HTML y los atributos en JSX (React) tienen algunas diferencias clave. Aunque **JSX se basa en la sintaxis de HTML**, hay ajustes necesarios para que sea compatible con las reglas de JavaScript y React.

Característica	HTML	JSX
Nombres de atributos	Minúsculas (`class`, `for`)	camelCase (`className`, `htmlFor`)
Booleanos	Sin valor (`disabled`)	Valor explícito (`disabled={true}`)
Estilos en línea	Cadena de texto	Objeto (`{ color: 'red' }`)
Eventos	Minúsculas (`onclick`)	camelCase (`onClick`)
Atributos personalizados	Cualquier nombre permitido	Prefijo `data-` necesario
Valores dinámicos	No permitido	`{}` para expresiones

Diferencias en Atributos HTML

Crea el componente **AttributeExample** manejara atributos como **checked**, **className**, **onChange** y **selected**. **src/components/AttributeExample.jsx.**

- checked y value en React son propiedades controladas, lo que significa que requieren un estado para su manejo.
- El evento onChange permite actualizar el estado del componente con los valores introducidos.

```jsx
import { useState } from "react";

function AttributeExample() {
  const [selected, setSelected] = useState(false);
  const [inputValue, setInputValue] = useState("");

  return (
    <div>
      <h2>Ejemplo de Atributos</h2>
      <label>
        <input
          type="checkbox"
          checked={selected}
          onChange={() => setSelected(!selected)}
        />
        ¿Seleccionado?
      </label>
      <br />
      <label>
        Texto:
        <input
          type="text"
          value={inputValue}
          onChange={(e) => setInputValue(e.target.value)}
        />
      </label>
      <p>Texto ingresado: {inputValue}</p>
      <p>Checkbox está {selected ? "marcado" : "desmarcado"}</p>
    </div>
  );
}

export default AttributeExample;
```

dalid aspasia LA FORMACIÓN DE TU FUTURO

Diferencias en Atributos HTML

Agrega al archivo **App.jsx** el componente componente **AttributeExample**. Recuerda que el archivo se encuentra en **src/App.jsx**.

```
import RegisterVehicle from './components/RegisterVehicle';
import ServiceList from './components/ServiceList';
import StatusMessage from './components/StatusMessage';
import FocusInput from './components/FocusInput';
import AttributeExample from './components/AttributeExample';

function App() {
  return (
    <div>
      <h1>Mantenimiento de Vehículos</h1>
      <RegisterVehicle />
      <ServiceList />
      <StatusMessage />
      <FocusInput />
      <AttributeExample />
    </div>
  );
}

export default App;
```

Adalid aspasia LA FORMACIÓN DE TU FUTURO

Utilidades para pruebas en React

React ofrece diversas utilidades para realizar pruebas de componentes y asegurarse de que funcionan como se espera. Estas utilidades se encuentran en bibliotecas como **react-dom**, **react-testing-library**, y otras. Una de las herramientas clave para realizar pruebas es el renderizador de prueba.

Renderizador de prueba en React

Para su implementación se pueden hacer usos de librerías como:

```
npm install --save-dev jest @testing-library/react @testing-library/jest-dom
```

El siguiente es un ejemplo de testing del componente **RegisterVehicle**

```
import { render, screen } from '@testing-library/react';
import RegisterVehicle from '../components/RegisterVehicle';

test('Muestra el formulario de registro de vehículo', () => {
  render(<RegisterVehicle />);
  expect(screen.getByText(/Registrar
Vehículo/i)).toBeInTheDocument();
  expect(screen.getByPlaceholderText(/Toyota
Corolla/i)).toBeInTheDocument();
});
```

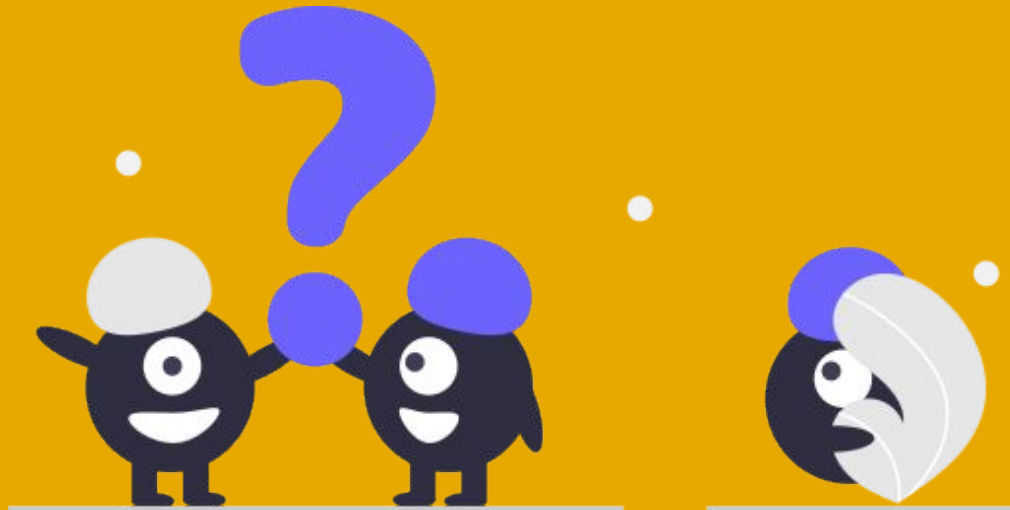
adalid aspasia LA FORMACIÓN DE TU FUTURO

Requerimientos del Entorno de JS

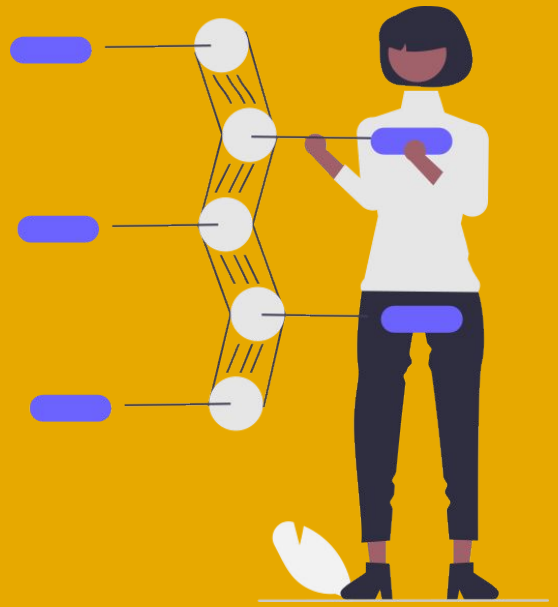
Para ejecutar y desarrollar aplicaciones con JavaScript en diferentes entornos, debes cumplir con ciertos requisitos de entorno según el caso.

Componente	Herramienta	Propósito
Motor de JS	Node.js	Ejecutar JS fuera del navegador.
Gestor de paquetes	npm o Yarn	Instalar y gestionar dependencias.
Transpilador	Babel	Convertir JS moderno para compatibilidad.
Bundler	Webpack, Parcel, o Vite	Empaquetar código para producción.
Testing	Jest, Mocha, Cypress	Asegurar calidad del código.
Linting	ESLint	Identificar errores de código.
Polyfills	core-js	Soportar navegadores antiguos.

¿Tienes alguna duda de la temática?



Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- **DOM en ReactJS**: Uso del Virtual DOM, React Fiber y su impacto en la eficiencia.
- **Referencias**: Creación, uso y acceso directo al DOM mediante referencias.
- **React-DOM**: Renderizado en cliente y servidor, con compatibilidad en navegadores.
- **Diferencias y pruebas**: Atributos HTML únicos en React y herramientas para pruebas. 🚀

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO