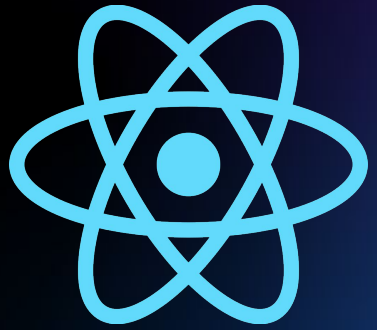
Clase 1 - React & Node



Crombie

Objetivos de la clase

- ➔ Entender qué es React, su historia y motivaciones detrás de su creación.
- ➔ Conocer las ventajas principales de React como los componentes reutilizables y el Virtual DOM.
- ➔ Comparar React con JavaScript vanilla.

Crombie

¿Qué es React?

- ➔ React es una biblioteca de JavaScript para construir interfaces de usuario interactivas.
- ➔ Desarrollado por Facebook en 2013.
- ➔ Uso principal: Crear componentes reutilizables que gestionen la interfaz de una aplicación.

Historia de React

- ➔ **Motivación**: React fue creado por Jordan Walke en Facebook para mejorar el manejo del creciente código en su plataforma.
- ➔ **Contexto**: Antes de React, las aplicaciones web tenían problemas con la manipulación directa del DOM, lo que causaba ineficiencia.
- ➔ **Lanzamiento**: En 2013 fue lanzado al público como código abierto.

Motivaciones detrás de React

- ➔ **Necesidad**: Facebook buscaba una forma más eficiente de actualizar y renderizar grandes volúmenes de datos.
- ➔ **Problema**: El DOM tradicional era lento y difícil de gestionar en aplicaciones dinámicas.
- ➔ **Solución**: Crear una biblioteca que fuera rápida, modular y eficiente para actualizar la UI.

Cuales son las ventajas de React?

- ➔ **Componentes Reutilizables**: Se pueden crear pequeños componentes modulares que se combinan para formar la interfaz.
- ➔ **Virtual DOM**: React utiliza un DOM virtual, que es mucho más rápido que actualizar el DOM real directamente.
- ➔ **Unidirectional Data Flow**: El flujo de datos en una sola dirección facilita el control de estados y la depuración.
- ➔ **Comunidad y Ecosistema**: Gran comunidad, herramientas poderosas como React Developer Tools y un ecosistema robusto (Next.js, React Router, etc.).

Componentes Reutilizables

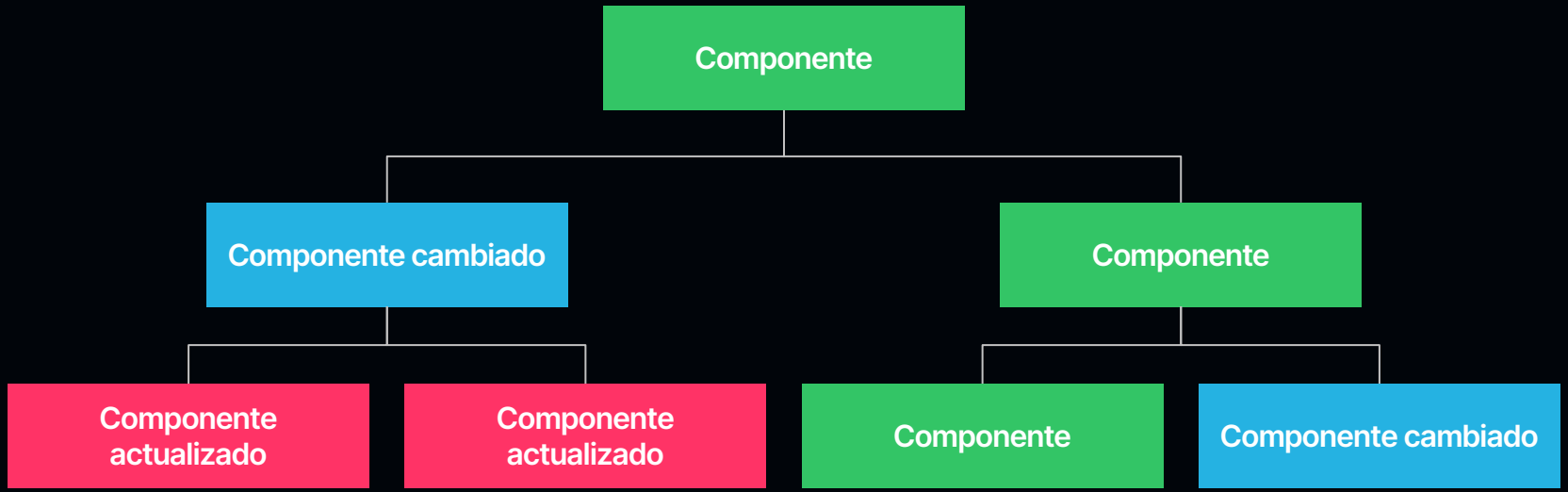
- ➔ En React, todo es un componente (botones, formularios, menús), lo que promueve la reutilización de código.
- ➔ Facilita la escalabilidad de aplicaciones grandes.

Crombie

Virtual DOM

- ➔ **Virtual DOM**: React crea un árbol de componentes en memoria (Virtual DOM) y solo actualiza las partes que han cambiado.
- ➔ Mejora el rendimiento al reducir las operaciones costosas en el DOM real.

Virtual DOM



Crombie

React VS Vanilla JavaScript

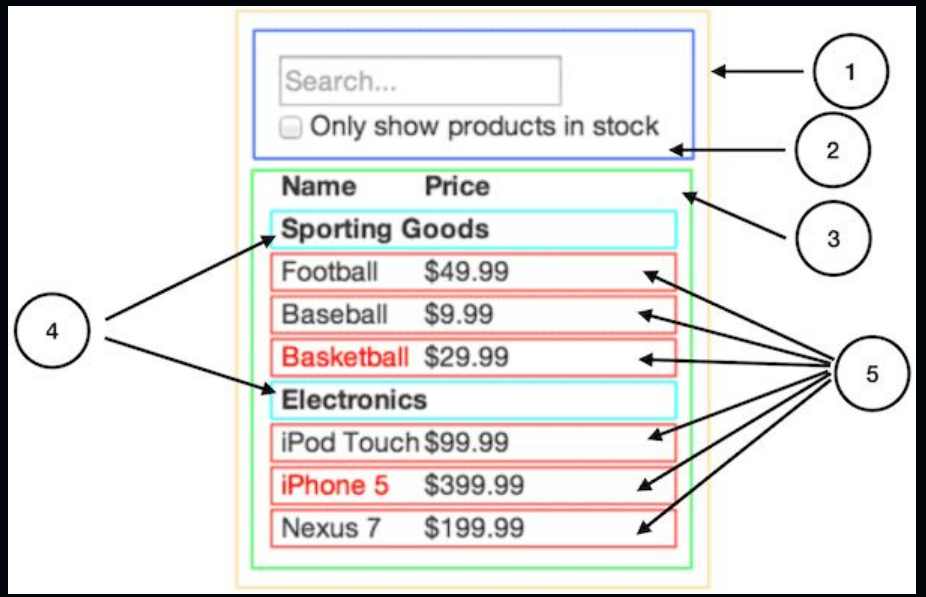
Característica	Vanilla JavaScript	React
Manipulación del DOM	Manual y propensa a errores	Automática con Virtual DOM
Reutilización del código	Limitada	Alta (componentes)
Complejidad en grandes apps	Alta	Más manejable

Comparación con otros frameworks

			
Tipo	Biblioteca enfocada en la UI	Framework completo	Framework progresivo
Rendimiento	Alto con Virtual DOM	Bueno, pero más pesado por ser completo	Alto con Virtual DOM
Arquitectura	Modular y flexible	Estructurada (MVC, mas rigida)	Punto intermedio entre React y Angular
Comunidad y ecosistema	Enorme, maduro, con herramientas como Next.js	Gran comunidad, muchas funcionalidades listas para usar	Comunidad en crecimiento, menos recursos pero buena adopción

Componentes en React

- Los componentes son bloques modulares de código que representan partes independientes de la interfaz de usuario en una aplicación React.
- Permiten la reutilización de código y mejoran la organización de las aplicaciones.



Crombie

Componentes funcionales

- ➔ Son funciones de **JavaScript** que devuelven **JSX**.
- ➔ Usan **Hooks** (como `useState`, `useEffect`).
- ➔ Sintaxis concisa.

Props y State

- Props:
  - Datos que se pasan de un componente a otro, inmutables dentro del componente. Padre → Hijo.
  - Permiten a los componentes recibir datos y personalizar su contenido.
  - Ej: myComponent({props}) ⇒ {....}

Props y State

- State:
  - Es un valor mutable que un componente funcional puede mantener y modificar usando el hook useState, por ejemplo.
  - Uso: Permite actualizar la UI de acuerdo a cambios de datos internos (ej: contador, checkbox).

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

function Counter() {
const [count, setCount] = useState(0);

const increment = () => setCount(count + 1);

return (
 <div>
  <p>Count: {count}</p>
  <button onClick={increment}>
   Increment
  </button>
 </div>
);

} export default Counter;
```

Crombie

¡Momento de práctica!

- Creemos un componente funcional “myCounter”.
- El componente debe constar de:
  - Un botón que el usuario pueda usar para sumar.
  - Una función que sea llamada por este botón.
  - Una invocación del hook useState para capturar y manipular el estado del contador.

Crombie

NodeJs

Crombie

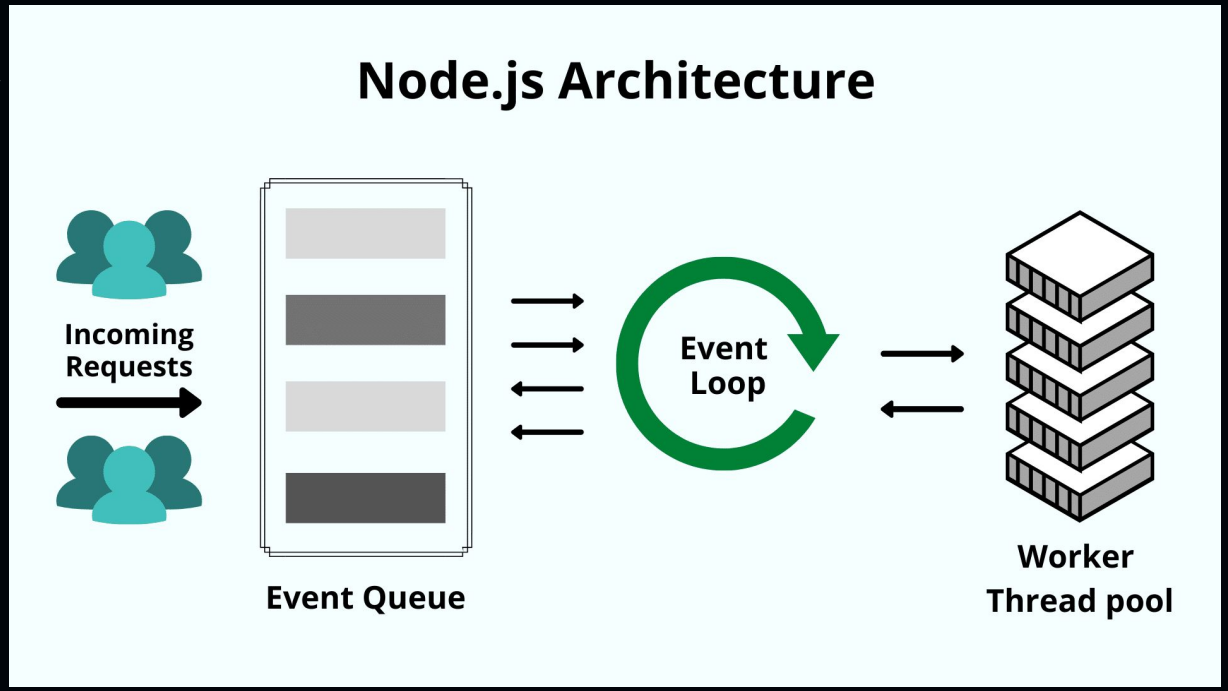
¿Qué es Node.js?

- "Node.js es un entorno de ejecución de JavaScript en el servidor."
- Desarrollado por Ryan Dahl en 2009.
- Permite ejecutar JavaScript fuera del navegador.

Características principales

- ➔ Creado para ejecutar JavaScript en el backend, aprovechando la velocidad del motor V8 de Chrome.
- ➔ Basado en el modelo de I/O no bloqueante.
- ➔ Usa un único hilo con un bucle de eventos (event loop).
- ➔ Gran rendimiento para tareas intensivas de I/O.

Event Loop



Crombie

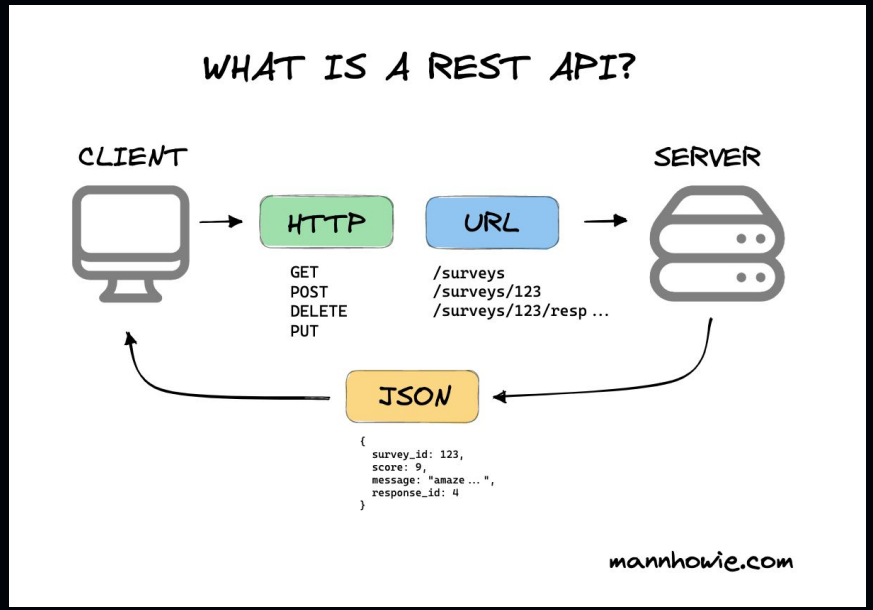
- ➔ Alto Rendimiento: Gracias a su arquitectura **asíncrona** y el motor V8.
- ➔ Escalabilidad: Fácil de escalar en aplicaciones distribuidas.
- ➔ Un solo lenguaje: Permite usar **JavaScript** tanto en frontend como en backend.
- ➔ Gran Ecosistema: npm facilita el acceso a miles de librerías y herramientas.

npm - Gestión de paquetes

- ➔ npm (Node Package Manager):
- ➔ Facilita la instalación, actualización y gestión de paquetes de Node.js.
- ➔ Ventaja: Acceso rápido a una gran comunidad de herramientas.

Creación de APIs con Node.js y Express

- Una API (Application Programming Interface) es una forma de comunicar aplicaciones, permitiendo el intercambio de datos y funcionalidades.
- Node.js se utiliza principalmente para crear APIs RESTful, facilitando la comunicación entre frontend y backend.



Crombie

Express

- ➔ Express es un framework minimalista para Node.js que simplifica la creación de rutas, manejo de middleware y gestión de respuestas.
- ➔ Ofrece una estructura clara para construir APIs y soporta middleware para ampliar sus funcionalidades (como body-parser para JSON).

```
import express from “express”
const app = express();
app.use(express.json());


app.get('/api/mensaje', (req, res) => {
  res.json({ mensaje: '¡Bienvenido a nuestra API!' });
});

app.post('/api/suma', (req, res) => {
  const { num1, num2 } = req.body;
  res.json({ resultado: num1 + num2 });
});


app.listen(3000, () => console.log("Servidor en
http://localhost:3000"));
```

¡Momento de práctica!

- Creemos una pequeña API con express.
- La API debe contar con:
  - Un endpoint que nos devuelva un nombre / mensaje.
  - Un endpoint que nos permita enviar dos números por el body y nos devuelva la suma.

Preguntas?