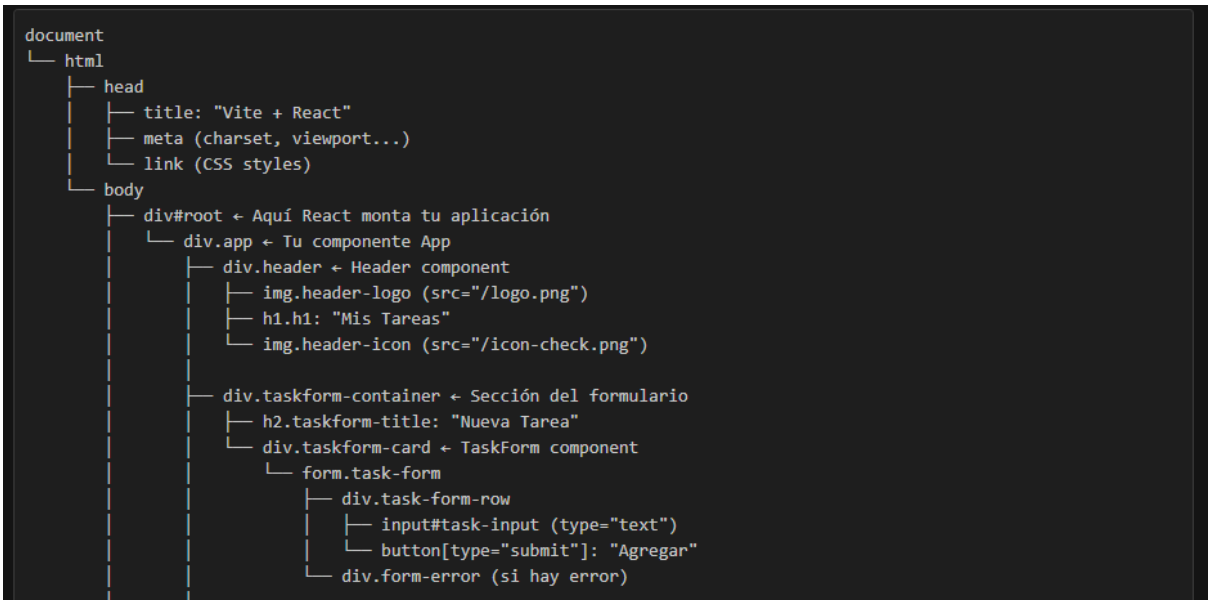
1. ¿Qué es el Virtual DOM y por qué React lo utiliza?

Para explicar que es el Virtual DOM comenzaré explicando que es el DOM. DOM significa Document Object Model y es una representación estructurada de la página web que el navegador crea automáticamente. Es una representación real de la página web en el navegador y vive en la memoria del navegador. Es como un “árbol” donde cada elemento HTML es una ”rama”.

Fragmento del DOM de To-DoApp:



El DOM permite darle dinamismo y vida a la página, y lo manipulamos mediante JavaScript. Pero esto es lento y costoso, por eso usamos el Virtual DOM.

El Virtual DOM es una representación en memoria del DOM real, hecha en JavaScript. Es como una “copia ligera” del DOM que React mantiene en JavaScript.

Aquí hago un paréntesis para explicar que React es una librería que trabaja sobre JavaScript que solo se encarga de la interfaz de usuario, es flexible ya que se puede combinar con otras herramientas. En modo desarrollo(npm run dev) hace una transformación en tiempo real de JSX a JavaScript y en modo producción(npm run build) compilamos completamente todo el código y lo transformamos a JS, HTML y CSS.

Volviendo al Virtual DOM, la idea de tener una representación virtual del DOM ya existía, se usaba en algunas librerías y frameworks antes de React. El virtual DOM viene de conceptos de programación funcional. En 2013, Facebook implementó su propia versión del Virtual DOM en React, lo hizo mainstream y conocido por todos. Además lo mejoró y optimizó significativamente y creó el algoritmo de Diffing eficiente.

El algoritmo Diffing compara dos árboles del Virtual DOM y encuentra diferencias entre el anterior y el nuevo. Luego el algoritmo de Reconciliación aplica los cambios al DOM real, después del Diffing . Es decir que React está manipulando el Virtual DOM con esos dos

algoritmos escritos dentro de la librería de React.

2. ¿Cuál es la diferencia entre estado (state) y props en React?

El userstate es un hook(función especial de React) que te permite crear una variable para luego manipularla mediante una función a la que le das nombre inmediatamente luego de la variable creada, y dentro del paréntesis declaramos el valor inicial e indicamos que deberá recibir si un arreglo una cadena etc para luego modificar esa variable. Es mutable, es interno al componente, se modifica mediante setState(la función que siempre contiene set + el nombre de la variable que creamos) y causa re-render cuando cambia.

Los props son datos que pasamos desde un componente padre a un componente hijo. Lo usamos para enviar funciones de useState, sus variables, pero también podemos enviar funciones creadas por nosotros mismos, arreglos, objetos, variables tipo String o Numbers. A diferencia de los argumentos, props es lo que recibimos del padre, en cambio si enviamos una variable cuando llamamos a una función que enviamos como prop, aqui estariamos enviando un argumento.

3. Explica en qué casos usarías useEffect.

El hook useEffect se usa para efectos secundarios(acciones extras). No maneja estado ni está directamente relacionado con renderizar la UI. Las acciones afectan a cosas fuera de los componentes, por ejemplo para el manejo de Timers, APIs, localStorage.

Por ejemplo, en la To-DoApp tenemos esta función:

```
const [tasks, setTasks] = useState(() => {
  try {
    const raw = localStorage.getItem('tasks');
    return raw ? JSON.parse(raw) : [];
  } catch (e) {
    console.warn('Failed to parse tasks from localStorage', e);
    return [];
  }
});

useEffect(() => {
  try {
    localStorage.setItem('tasks', JSON.stringify(tasks));
  } catch (e) {
    console.warn('Failed to persist tasks to localStorage', e);
  }
}, [tasks]);
```

Al iniciar, useState lee una vez las tareas guardadas en el navegador usando localStorage(API nativa del navegador que tiene métodos para guardar y acceder a datos guardados en el navegador). Con este obtenemos las tareas(datos) que guardamos antes en el navegador.

Luego se ejecuta useEffect. Mediante el metodo setItem le decimos que guarde las tareas, pero como localStorage solo acepta string usamos JSON.stringify para convertir el objeto tareas.

Aqui es donde usamos useEffect, para que cada vez que una tarea cambie, se ejecute localStorage.setItem y guarde las tareas en el navegador. Se ejecuta despues de cada render y re-ejecuta cada vez que cambian las dependencias.

Flujo completo:

1. App se monta
   ↓
2. useState se ejecuta (LEE localStorage)
   ↓
3. Componente se renderiza por primera vez
   ↓
4. useEffect se ejecuta (GUARDA en localStorage)
   ↓
5. Usuario interactúa (agrega/elimina tarea)
   ↓
6. useState cambia (setTasks)
   ↓
7. Componente se re-renderiza
   ↓
8. useEffect se ejecuta de nuevo (GUARDA cambios)