Day - 9 VDOM,FileStructure 04.12.2025

Babel :-

Babel is a JavaScript transpiler.
It converts **modern JavaScript (ES6+) into older JavaScript** that browsers can understand.
A **transpiler** converts **one high-level language to another high-level language** (e.g., ES6 → ES5,JSX to JS).

React Router DOM :-

React Router DOM is used to build Single Page Applications (SPAs).
It lets React update the **URL** and **render different components without reloading the page**.

Virtual DOM :-

- **React does not update the real DOM directly.**
- It creates a **Virtual DOM**, a lightweight copy of the real DOM.
- When state changes, React performs **Diffing**
  → compares old Virtual DOM with new Virtual DOM.
- Then React performs **Reconciliation**
  → updates **only the changed parts** of the real DOM.

This makes React **fast and efficient**.

What is the Real DOM?

Real DOM (Document Object Model) is the **actual browser HTML tree**.
It represents your webpage as **objects**, where each HTML element is a **node** in a tree structure.

Example:
HTML → Browser converts it into a **DOM Tree** (object representation).

React and Virtual DOM

React does **NOT** update the Real DOM directly because it is **slow** to update.

So React first:

1. **Creates a lightweight copy of the Real DOM in memory.**
2. This copy is the **Virtual DOM (VDOM)**.
3. When state changes, React updates the Virtual DOM.
4. React compares old VDOM and new VDOM (Diffing).
5. Only the changed nodes are updated in the **Real DOM** (Reconciliation).

Why React Uses Virtual DOM

Updating the **Real DOM is slow** because every change in the real DOM can trigger **expensive browser operations** like:

- **Reflow** → recalculating layout
- **Repaint** → redrawing pixels
- **Layout calculations** → recalculating positions, sizes, styles
- **Style recalculation** → computing CSS changes

These operations block the main thread and reduce performance.

Very Important for Interview

1. Diffing

When something in your UI changes (like state or props), React needs to update the screen. But instead of updating the real DOM immediately, React follows these steps:

1. **React creates a new Virtual DOM tree**
   – This is the updated version of how the UI should look.
2. **React compares this new Virtual DOM with the old Virtual DOM**
   – It checks what changed between the two versions.
3. **React finds the smallest possible differences**
   – This helps React avoid unnecessary updates.
4. **React updates only those changed parts in the real DOM**
   – Not the whole page, only the exact elements that changed.

This entire comparison process is called **Diffing**.

Reconciliation

Reconciliation is the **full process React uses to update the UI** when something changes.

Here is how it works:

1. **React receives a new UI**
   – This happens when state or props change.
2. **React creates a new Virtual DOM**
   – A fresh version of how the UI should look.
3. **React compares (diffs) the new Virtual DOM with the old one**
   – It checks what exactly changed.
4. **React decides which parts need updates**
   – Some nodes may need to be updated, some removed, some added.
5. **React applies only the minimal required changes to the Real DOM**
   – Instead of re-rendering the whole page, only the changed parts are updated.

This entire update process is called **Reconciliation**.

✅Step 1: JSX → Virtual DOM

React first **converts your JSX into a Virtual DOM tree**.
This is a lightweight JS object that represents your UI.

✅Step 2: Data Changes → Re-render (in memory)

Whenever **state or props change**, React **re-renders the component tree**,
but **only in the Virtual DOM** — *not the real DOM*.

This produces a **new Virtual DOM**.

✅Step 3: Diffing Algorithm

React compares the **old VDOM** with the **new VDOM** and determines:

- what stayed the same
- what changed
- what needs to be removed
- what needs to be added

✅Step 4: Patch Generation (List of DOM updates)

React prepares a **patch object**,
which is basically a **list of minimal DOM operations** required to update the real UI.

This ensures React updates **only what's necessary**.

✅Step 5: Commit Phase (Real DOM Update)

React applies the patch to the **Real DOM** using the **browser's DOM API**.
Only the changed nodes are updated.

CRA (Create React App) — Interview Answer

CRA uses Webpack under the hood. Webpack bundles the entire project into a single large bundle before running the app. This bundling process is slow, so startup time and rebuild time become very high, especially in large projects. That's why CRA feels slow.

Vite — Interview Answer

Vite uses Rollup for production builds and uses a lightning-fast dev server powered by ES modules. Instead of bundling everything upfront, Vite serves files on demand. This makes startup extremely fast and updates almost instant

React Project Folder Structure — Full Explanation

---

1. `node_modules/`

`node_modules` is a **library storage folder**.

- When you install packages (`npm install react`, `npm install axios`, etc.), all those libraries are downloaded into this folder.
- It contains **thousands of pre-defined functions** and dependencies used by your project.
- This folder is **auto-generated** and **never pushed to GitHub** (ignored using `.gitignore`).

☞ **Simple line:**
`node_modules` stores all the libraries and dependencies your project needs.

---

2. `public/`

`public` contains **static assets** that do NOT get processed by JavaScript bundlers.

Key points:

- Stores `index.html` → the **root HTML** where React injects the entire app.
- Holds images, icons, fonts that are referenced using **absolute paths**.
- Content here is directly copied to the final build without modification.

☞ **Simple line:**
`public` holds static files and the main HTML file used by the browser.

---

3. `src/`

`src` is the **most important folder** because this is where your entire React app lives.

It contains:

- `main.jsx` → Entry point (ReactDOM creates the app here)
- `App.jsx` → Root component
- Components (`/components`)

- Pages (`/pages`)
- Styles (`.css`)
- Hooks
- Context
- Assets

Everything inside `src/` gets processed and compiled by Vite.

☞ **Simple line:**
`src` contains all your React components, logic, styles, and main application code.

---

4. `.gitignore`

The `.gitignore` file tells Git **which files should not be uploaded to GitHub**.

Common ignored items:

- `node_modules/`
- `dist/` or `build/`
- `.env`
- Log files
- Temporary files

☞ **Simple line:**
`.gitignore` prevents unnecessary or sensitive files from going to GitHub.

---

5. `eslint.config.js`

This file controls **ESLint settings**.

- ESLint checks your code for errors, bad practices, unused variables, etc.
- Helps maintain **clean and consistent code quality**.
- Prevents bugs by enforcing rules.

☞ **Simple line:**
`eslint.config.js` manages code-quality rules and detects errors automatically.

6. `index.html`

This is the main HTML file where your entire React app is injected.
React does not create multiple pages — your whole application starts from the `<div id="root"></div>` inside this file.

☞ **Simple line:**
`index.html` is the file where React renders the whole UI.

---

7. `package.json`

This file stores the project dependencies, scripts, version info, and metadata.
It controls:

- Installed libraries (react, axios, tailwind, etc.)
- Project version
- Commands like `npm run dev`, `npm run build`
- Project configuration

☞ **Simple line:**
`package.json` manages dependencies and run commands for your project.

---

8. `package-lock.json`

This file locks the exact versions of every dependency installed.
It ensures:

- Same versions get installed on every machine
- Stability and predictability during testing

☞ **Simple line:**
`package-lock.json` ensures consistent dependency versions.

(Not only for testing — also for security, stability.)

---

9. `README.md`

This file describes your project: how to install, run, features, usage, etc.
It is shown on GitHub as your project documentation.

☞ **Simple line:**
`README.md` is the project description and documentation file.

---

10. `vite.config.js`

Important configuration file for Vite.
Controls:

- Dev server settings

- Path aliases (`@/components`)
- Plugins (like React plugin)
- Build configuration

☞ **Simple line:**
`vite.config.js` controls how Vite builds and runs your project.

✸ Component Files

`main.jsx`

Entry point of the app — React creates the Virtual DOM and mounts `<App />` into `index.html`.

Steps:

1. Imports React and ReactDOM
2. Creates the Virtual DOM root
3. Renders the `<App />` component

☞ **Simple line:**
`main.jsx` starts the entire app and mounts React into the DOM.

`App.jsx`

Main component — usually the first file where your UI code begins.
This is where:

- Routes start
- Global layout
- Major components are imported

☞ **Simple line:**
`App.jsx` is the root component where your actual work begins.

`App.css`

Simple CSS file used only for styling the App component or global elements.

☞ **Simple line:**
`App.css` contains regular CSS for your app's layout.

`index.css`

Common place to import global styles like Tailwind, Bootstrap, or global resets.

Example:

```
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Bootstrap import if needed */
```

☞ **Simple line:**
`index.css` is for global styles and framework imports (Tailwind, Bootstrap).

3 Ways to Return JSX in React

When a React component returns UI, it must return **one parent element**.
Here are the 3 methods:

---

1. Return inside parentheses `()`

Used for clean formatting.

```
return (
  <h1>Hello</h1>
);
```

---

2. Return using Fragment `<>...</>`

A Fragment lets you return **multiple elements without adding an extra HTML tag**.

```
return (
  <>
    <h1>Hello</h1>
    <p>World</p>
  </>
);
```

---

3. Return using a Wrapper Element like `<div>...</div>`

```
return (
  <div>
    <h1>Hello</h1>
    <p>World</p>
  </div>
);
```

⋆ What Is a Fragment? (Very Important for Interview)

Fragment = Empty wrapper (<> </>) that does NOT create extra nodes in the real DOM.
React requires components to return **one parent element**.
Fragments allow you to group multiple child elements **without adding a div**.

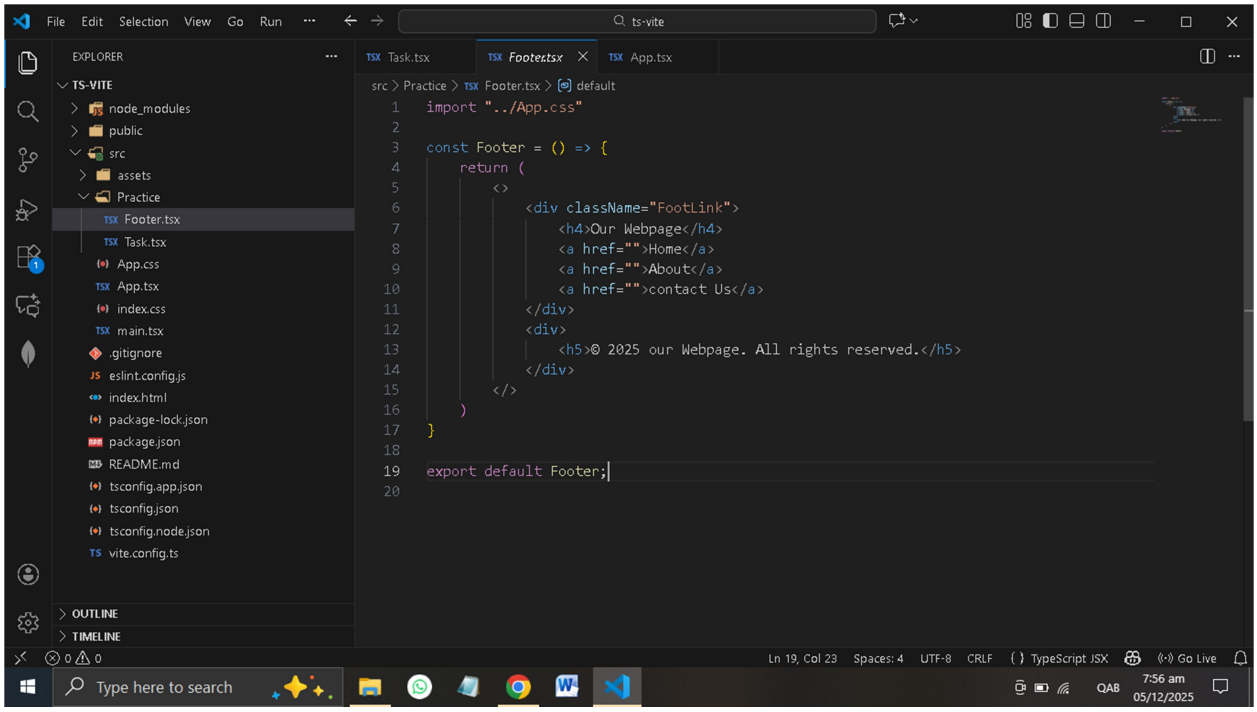
Example:

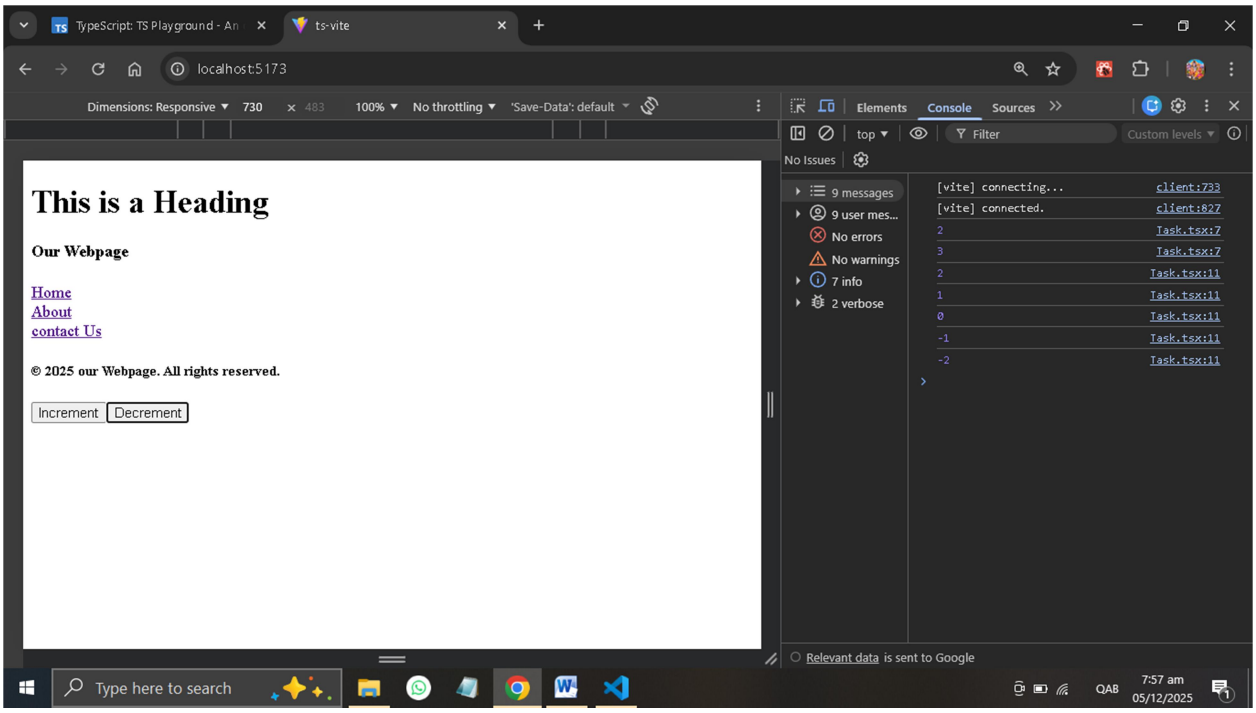
✕Using div

```
<div>
  <h1>Title</h1>
  <p>Description</p>
</div>
```

✔ Using Fragment

```
<>
  <h1>Title</h1>
  <p>Description</p>
</>
```




TypeScript: TS Playground - An
ts-vite
localhost:5173
Dimensions: Responsive 730 × 483 100% No throttling 'Save-Data': default
This is a Heading
Our Webpage
Home
About
contact Us
© 2025 our Webpage. All rights reserved.
Increment
Decrement
Elements
Console
Sources
top
Filter
Custom levels
No Issues
9 messages
9 user mes...
No errors
No warnings
7 info
2 verbose
[vite] connecting... client:733
[vite] connected. client:827
2 Task.tsx:7
3 Task.tsx:7
2 Task.tsx:11
1 Task.tsx:11
0 Task.tsx:11
-1 Task.tsx:11
-2 Task.tsx:11
Relevant data is sent to Google
Type here to search
QAB
7:57 am
05/12/2025