Some basic stuff regarding HTML and DOM:

What is the relationship between DOM and HTML… How they are related to each other?

The Document Object Model (DOM) and an HTML file serve different purposes but are closely related in the context of web development. Let's clarify the distinctions between the DOM and an HTML file:

**HTML File:**

1. **Static Document:**
   * An HTML (Hypertext Markup Language) file is a plain text file that contains markup tags defining the structure and content of a web document.
2. **Human-Readable Code:**
   * HTML files are created and edited by developers using a text editor. They consist of tags (e.g., **<html>**, **<head>**, **<body>**, **<p>**) that describe the elements and content of a web page.
3. **Stored on Disk:**
   * HTML files are static and are stored on the server's file system or distributed to clients. When a user requests a webpage, the server sends the corresponding HTML file to the user's browser.
4. **Not Interactive:**
   * An HTML file alone does not have dynamic or interactive capabilities. It represents the initial, static structure of a webpage before any user interactions or modifications.

**Document Object Model (DOM):**

1. **Dynamic Representation:**
   * The DOM is a dynamic, in-memory representation of an HTML document that a browser creates when it loads an HTML file.
2. **JavaScript Interaction:**
   * The DOM is manipulated and interacted with using JavaScript. JavaScript code can modify the DOM to dynamically update the content, structure, and style of a webpage in response to user actions or other events.
3. **Tree-like Structure:**
   * The DOM represents the HTML document as a tree-like structure where each HTML element is a node, and relationships between elements are defined by parent-child relationships.
4. **Live and Responsive:**
   * Unlike an HTML file, the DOM is responsive to changes made by JavaScript. When JavaScript modifies the DOM, the changes are immediately reflected in the displayed webpage without requiring a full page reload.

**Relationship:**

* **HTML as Source Code:**
  + The HTML file serves as the source code for a webpage. It provides the initial structure and content, including the markup tags that define elements and their relationships.
* **DOM as Runtime Representation:**
  + The DOM is a runtime representation of the HTML document. When a browser loads an HTML file, it creates the DOM, which is then used to render the webpage and respond to user interactions.

In summary, an HTML file is a static document that provides the initial blueprint for a webpage, while the DOM is a dynamic, in-memory representation that the browser creates based on the HTML file. The DOM can be manipulated and updated in real-time using JavaScript, enabling interactive and responsive web applications.

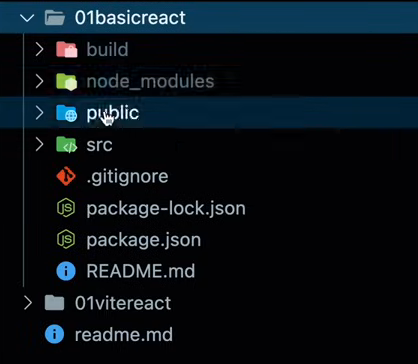
IS an HTML element converted into javascript object at the end of the day while rendering?

HTML elements themselves are not directly converted into JavaScript objects, JavaScript can interact with the DOM, treating DOM nodes as objects with properties and methods. This interaction allows developers to dynamically modify the content and structure of a web page, creating dynamic and interactive user interfaces.

**The Foundation**

Understanding the React flow and Structure :

**Understanding the Basic Folder structure of react jitha aapn kam karnar:**



Jevde pn dependencies mala package.json madhe distayt te download or install houn **node\_modules** madhe bastat

Most of the kam je asnar ahe te **source folder** madhe asnar kiva **public folder** madhe asnar

Talking in context of create-react-app wala react project

**In Public Folder:**

Public folder madhe important file ahe index.html which is where all the component code will be painted by react as and when needed …during runtime

This is the reason why it is called as single page application

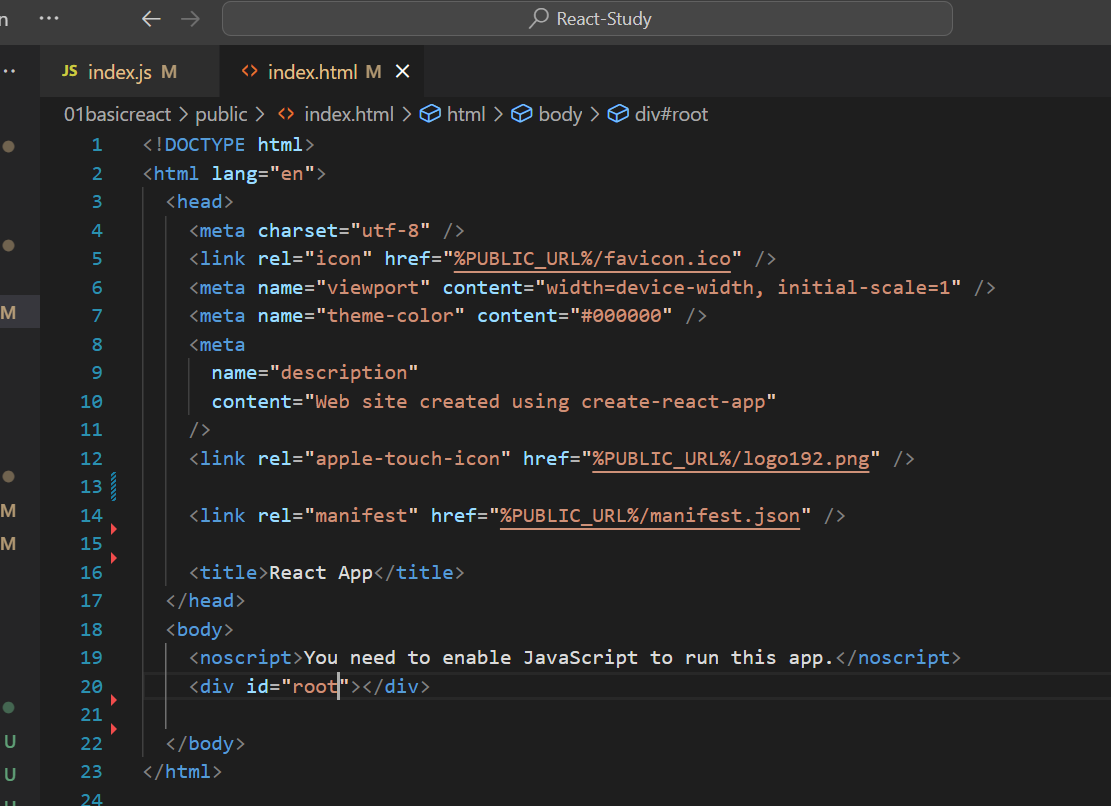
**In Source Folder:**

Index.js ani App.js are important files in this

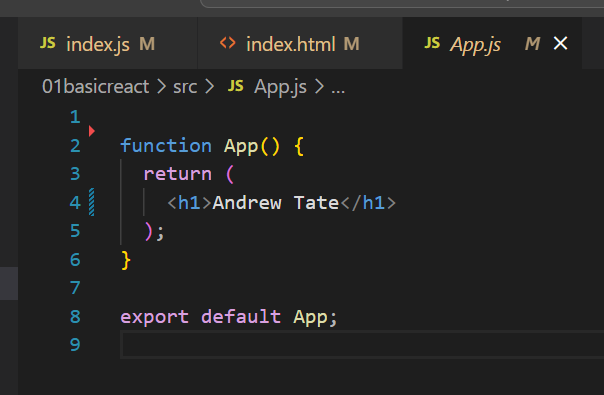
Index.js pasun ek virtual dom create hotay memory madhe and which maps whatever components are there in the browser dom.

App.js pasun aapn index.js madhe components pathvnar which will render them.

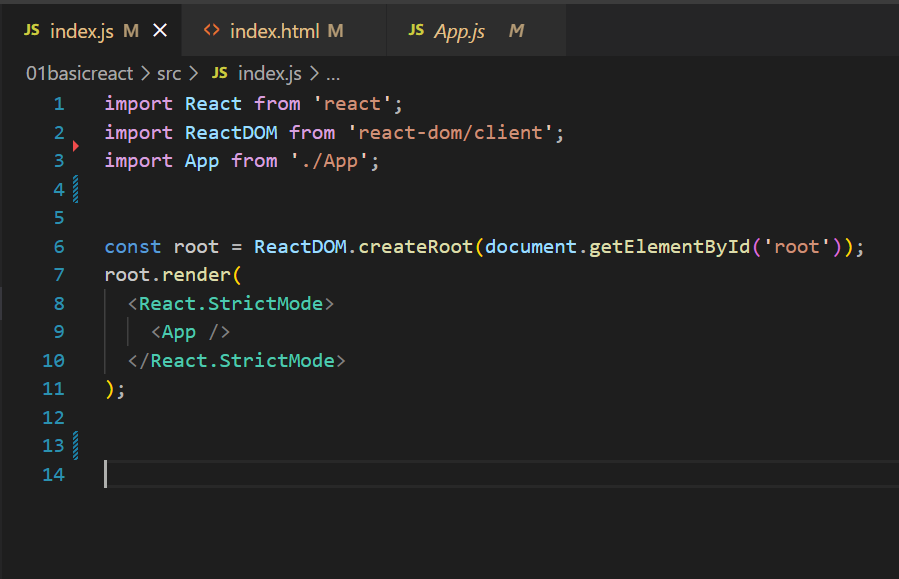
Index.html



App.js: Ithun aapn index.js la components export karnar…mhnje tithe index.js

Virtual dom madhe render karu shaknar

Index.js madhe virtual dom vrti render vhyla sodlay



**Virtual DOM cha reference aapn root variable madhe ghetoy and then aapn tyala render karyla sangtoy**

Jr webpage load karat astana ky langar asel tr react script te add karat jate necessary goshti (create-react-app chya context madhe)

**React madhe components banavna means what ?**

Eka jsx kiva js file madhe function banvun HTML cha code return karne is what creating components means

**Creating First React Component:**

**We will make projects in Vite+ react environment**

Steps:

1. Create a new jsx file which will denote our component
2. In that create a function which returns HTML … export default that
3. Import that in the App.jsx file and include that in an appropriate area in tag format
4. Run the application

Learnings:

It’s a good practice to keep the file name and component name starting with a capital letter

**Understanding JSX via creating our own react library**

**Through code we concluded the following**

practically saw whatever the JSX which our functional components are returning they are first transpiled into JS objects or **react elements** (where they are loaded with properties which might be required for the future optimizations or algorithms ) and then given for the virtual DOM to render it

when you return JSX from a functional component, React internally processes it, converts it into React elements, and uses these elements to build and update the virtual DOM, which eventually leads to the efficient rendering of the actual DOM.

Since at the end we have an intermediary state where babble converts the JSX into React elements … **It is possible to create our own react elements and then give for the virtual dom to render it.**

**Another point to be noted here is that we can inject JS variables in our JSX code. JSX supports that functionality as well… Note here that only evaluated expression can be written inside the JSX**

**Custom react element creation Syntax:**

const element = React.createElement(

tag name,

props which is an object used to set the properties of the tag,

innerHTML ,

variables which can be injected

)

How JSX and HTML are different?

JSX and HTML look very similar, but they are distinct in a few key ways:

1. **Syntax:**
   * **JSX:** JSX is a syntax extension for JavaScript. It allows you to write XML-like code (HTML-like) within your JavaScript files. JSX tags resemble HTML, but they are not exactly the same.

const element = <div>Hello, JSX!</div>;

* + **HTML:** HTML is a markup language used to structure content on the web. It is a standalone language with its own syntax.

<div>Hello, HTML!</div>

1. **Attribute Names:**
   * **JSX:** Attribute names in JSX use camelCase, similar to how you would write attribute names in JavaScript.

const element = <input type="text" onChange={handleInputChange} />;

* + **HTML:** Attribute names in HTML are case-insensitive and typically written in lowercase.

<input type="text" onchange="handleInputChange()" />

1. **Class vs. className:**
   * **JSX:** To specify CSS classes in JSX, you use the **className** attribute.

const element = <div className="myClass">Styled with JSX</div>;

* + **HTML:** In HTML, you use the **class** attribute to specify CSS classes.

<div class="myClass">Styled with HTML</div>

1. **Comments:**
   * **JSX:** Comments in JSX are written as curly-brace-enclosed JavaScript comments.

const element = ( <div> {/\* This is a JSX comment \*/} Hello, JSX! </div> );

* + **HTML:** HTML comments are written using **<!-- -->**.

<div> <!-- This is an HTML comment --> Hello, HTML! </div>

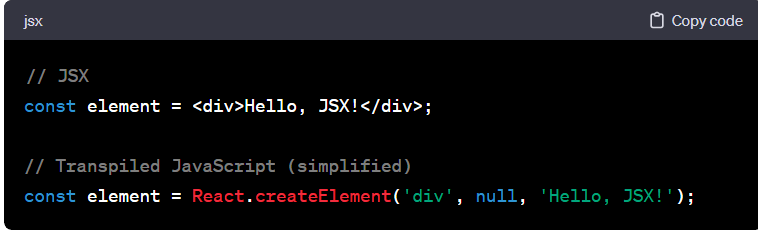
1. **Usage:**
   * **JSX:** JSX is primarily used in React applications to describe the structure of UI components within JavaScript files.
   * **HTML:** HTML is the standard markup language for web pages and is used in static HTML files.

Despite these differences, JSX is designed to be familiar to developers who are used to writing HTML. JSX gets transpiled to JavaScript by tools like Babel before being rendered by the browser. It allows developers to write UI components in a syntax that closely resembles HTML while leveraging the power and expressiveness of JavaScript.

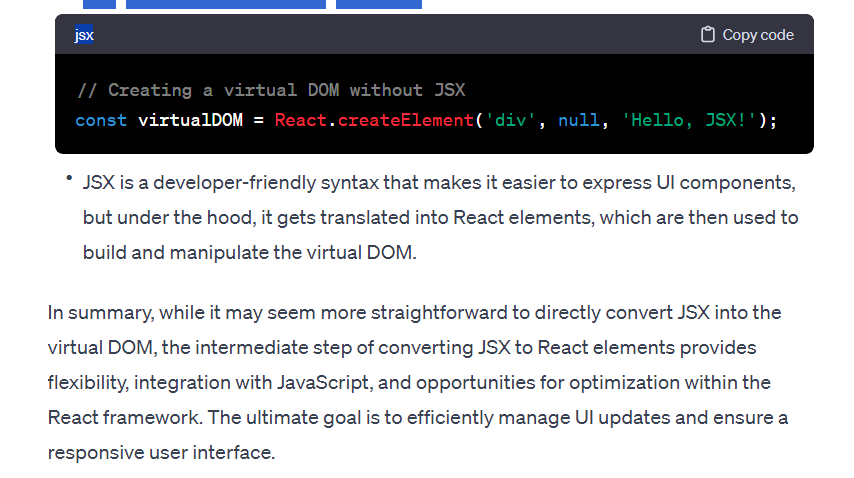
One question which i am having is that if HTML elements can easily be built into DOM tree ... then why JSX (which contains HTML tags) is firstly converted into javascript objects or react elements and then it is added to the virtual dom... why it is not directly converted into virtual dom through tags only .... Am i thinking in the right direction or is there any inconsistency in my understanding?

Your question is insightful, and there's no inconsistency in your understanding. Let me clarify the process and explain why JSX is first converted into React elements before being added to the virtual DOM.

1. **JSX to React Elements:**
   * JSX is a syntactic extension for JavaScript that looks similar to HTML. However, browsers don't understand JSX directly; they only understand HTML and JavaScript.
   * React, being a JavaScript library, provides a way to represent UI components as JavaScript objects called React elements.
   * JSX is processed by a tool like Babel, which transforms it into JavaScript code that creates React elements.

****

1. **Reasons for JSX Transformation:**
   * **Integration with JavaScript:** JSX allows seamless integration with JavaScript. By converting JSX into JavaScript objects (React elements), React can leverage the full power of JavaScript to manage and manipulate components.
   * **Optimizations and Diffing:** Converting JSX to React elements before reaching the virtual DOM allows React to perform optimizations during the diffing process. React can efficiently compare the new and old virtual DOMs, identifying which parts of the DOM need updating without directly manipulating the actual DOM for every change.
   * **Component Lifecycle and State Management:** Converting JSX to React elements enables React to manage component lifecycles, state, and other aspects of component behavior. It provides a consistent and predictable way to handle component updates and rendering.
2. **Direct Virtual DOM Creation:**
   * While React introduces an additional step in converting JSX to React elements, it's worth noting that you can create a virtual DOM directly without using JSX by using the **React.createElement** function.



a

**Creating Custom react elements**

We learnt, At the end JSX is converted into react elements / JS objects … before being rendered into virtual DOM … **You can even create your own react elements and give for the virtual DOM to render it…**

