Namaste React Course by Akshay Saini

Chapter 04 - Talk is Cheap, show me the code

Q: Is JSX mandatory for React?

A: JSX is an Extension Syntax that allows writing HTML and Javascript together easily in React and is used to create React elements. These elements are then rendered to the React DOM. Each JSX element is just to make use of React easy and for calling React.createElement(component, props, …children) with less work. So, anything that is done with JSX can also be done with just plain JavaScript. So JSX is not mandatory but is used for writing better and clean code instead of writing code using `React.CreateElement`.

Example of JSX

```
const sample = <h2>Greetings</h2>;
```

Q: Is ES6 mandatory for React?

A: ES6 is not mandatory for `React` but is highly recommendable. The latest projects created on React rely a lot on ES6. React uses ES6, and you should be familiar with some of the new features like: Classes, Arrow Functions, Variables(let, const).
ES6 stands for ECMAScript 6. ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015.

Q: `{TitleComponent}` vs `{<TitleComponent/>}` vs `{<TitleComponent></TitleComponent>}` in JSX.

A: The Difference is stated below:

- `{TitleComponent}`: This value describes the `TitleComponent` as a javascript expression or a variable or React element.
  The `{}` can embed a javascript expression or a variable or React element inside it.
- `<TitleComponent/>` : This value represents a Component that is basically returning Some JSX value. In simple terms `TitleComponent` a function that is returning a JSX value. If component is written inside the `{< />}` expression.
- `<TitleComponent></TitleComponent>` : `<TitleComponent />` and `<TitleComponent></TitleComponent>` are equivalent only when `< TitleComponent />` has no child components. The opening and closing tags are created to include the child components.

Example

```
<TitleComponent>
    <FirstChildComponent />
    <SecondChildComponent />
    <ThirdChildComponent />
</TitleComponent>
```

Q: How can I write `comments` in JSX?

A: JSX comments are written as follows:

- `{/* */}` - for single or multiline comments

Example

```
{/* A JSX comment */}
{/*
  Multi
  line
  JSX
  comment
*/}
```

Q: What is `<React.Fragment></React.Fragment>` and `<></>`?

A: `<React.Fragment></React.Fragment>` is a feature in React that allows you to return multiple elements from a React component by allowing you to group a list of children without adding extra nodes to the DOM.
`<></>` is the shorthand tag for `React.Fragment`. The only difference between them is that the shorthand version does not support the key attribute.

Example

```
return (
        <React.Fragment>
            <Header />
            <Navigation />
            <Main />
            <Footer />
        </React.Fragment>
    );

return (
        <>
            <Header />
            <Navigation />
            <Main />
            <Footer />
        </>
    );
```

Q: What is `Reconciliation` in React?

A: `Reconciliation` is the process through which React updates the Browser DOM and makes React work faster. React use a `diffing algorithm` so that component updates are predictable and faster. React would first calculate the difference between the real DOM and the copy of DOM (Virtual DOM) when there's an update of components.
React stores a copy of Browser DOM which is called `Virtual DOM`. When we make changes or add data, React creates a new Virtual DOM and compares it with the previous one. Comparison is done by `Diffing Algorithm`.
React compares the Virtual DOM with Real DOM. It finds out the changed nodes and updates only the changed nodes in Real DOM leaving the rest nodes as it is. This process is called Reconciliation.

Q: What is `React Fiber`?

A: React Fiber is a concept of ReactJS that is used to render a system faster, smoother and smarter.
The Fiber reconciler, which became the default reconciler for React 16 and above, is a complete rewrite of React’s reconciliation algorithm to solve some long-standing issues in React.
Because Fiber is asynchronous, React can:

- Pause, resume, and restart rendering work on components as new updates come in
- Reuse previously completed work and even abort it if not needed
- Split work into chunks and prioritize tasks based on importance

Q: Why do we need keys in React?

A: A key is a special attribute you need to include when creating lists of elements in React. Keys are used in React to identify which items in the list are changed, updated, or deleted. In other words, we can say that keys are unique Identifier used to give an identity to the elements in the lists.
Keys should be given to the elements within the array to give the elements a stable identity.

Example

```
<li key={0}>1</li>
<li key={1}>2</li>
<li key={2}>3</li>
```

Q: Can we use `index as keys` in React?

A: Yes, we can use the `index as keys`, but it is not considered as a good practice to use them because if the order of items may change. This can negatively impact performance and may cause issues with component state.
Keys are taken from each object which is being rendered. There might be a possibility that if we modify the incoming data react may render them in unusual order.

Q: What is `props in React`? Ways to.

A: props stands for properties. Props are arguments passed into React components. props are used in React to pass data from one component to another (from a parent component to a child component(s)). They are useful when you want the flow of data in your app to be dynamic.

Example

```
function App() {
  return (
    <div className="App">
      <Tool name="Chetan Nada" tool="Figma"/> // name and tool are props
    </div>
  )
}
```

Q: What is `Config Driven UI`?

A: `Config Driven UI` are based on the configurations of the data application receives. It is rather a good practice to use config driven UIs to make application for dynamic.
It is a very common & basic approach to interact with the User. It provides a generic interface to develop things which help your project scale well. It saves a lot of development time and effort.
A typical login form, common in most of the Apps. Most of these forms also get frequent updates as the requirements increase in terms of Form Validations, dropdown options,.. or design changes.

Q: Difference between `Virtual DOM` and `Real DOM`?

A: DOM stands for `Document Object Model`, which represents your application UI and whenever the changes are made in the application, this DOM gets updated and the user is able to visualize the changes. DOM is an interface that allows scripts to update the content, style, and structure of the document.

- `Virtual DOM`
    - The Virtual DOM is a light-weight abstraction of the DOM. You can think of it as a copy of the DOM, that can be updated without affecting the actual DOM. It has all the same properties as the real DOM object, but doesn’t have the ability to write to the screen like the real DOM.
    - Virtual DOM is just like a blueprint of a machine, can do the changes in the blueprint but those changes will not directly apply to the machine.
    - Reconciliation is a process to compare and keep in sync the two files (Real and Virtual DOM). Diffing algorithm is a technique of reconciliation which is used by React.
- `Real DOM`
    - The DOM represents the web page often called a document with a logical tree and each branch of the tree ends in a node and each node contains object programmers can modify the content of the document using a scripting language like javascript and the changes and updates to the dom are fast because of its tree-like structure but after changes, the updated element and its children have to be re-rendered to update the application UI so the re-

rendering of the UI which make the dom slow all the UI components you need to be rendered for every dom update so real dom would render the entire list and not only those item that receives the update .

Real DOM	**Virtual DOM**
DOM manipulation is very expensive	DOM manipulation is very easy
There is too much memory wastage	No memory wastage
It updates Slow	It updates fast
It can directly update HTML	It can’t update HTML directly
Creates a new DOM if the element updates.	Update the JSX if the element update
It allows us to directly target any specific node (HTML element)	It can produce about 200,000 Virtual DOM Nodes / Second.
It represents the UI of your application	It is only a virtual representation of the DOM