Assignment 04 - Talk is Cheap, show me the code

Owner	Pankaj Kumar

1: Is `JSX` mandatory for React?

- JSX is not a requirement for using React.
- Using React without JSX is especially convenient when you don't want to set up compilation in your build environment.
- Each JSX element is just syntactic sugar for calling React.createElement(component, props, ...children).
- So, anything you can do with JSX can also be done with just plain JavaScript.

Example of `JSX`

```
const header = <h1>Hello World!</h1>;
```

2: Is `ES6` mandatory for React?

- ES6 is not mandatory for `React` but is highly recommendable.
- ES6 is the standardization of javascript for making code more readable and more accessible.
- 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.
- ES6 is the standardization of javascript for making code more readable and more accessible.

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

- `{TitleComponent}`: This value describes the `TitleComponent` as a javascript expression or a variable.
  The `{}` can embed a javascript expression or a variable 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.
  A 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>
```

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

JSX comments are written as follows:

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

Example

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

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

- `<React.Fragment> </React.Fragment>` is a A common pattern in React is for a component to return multiple elements.
- `<React.Fragment>` is a component exported by React.
- Fragments let you 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 />
            <Body />
            <Footer />
        </React.Fragment>
    );

------------ OR --------------

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

6: What is `Reconciliation` in React?

- Reconciliation is the process by which React updates the UI to reflect changes in the component state. The reconciliation algorithm is the set of rules that React uses to determine how to update the UI in the most efficient way possible.
- React uses a virtual DOM (Document Object Model) to update the UI. The virtual DOM is a lightweight in-memory representation of the real DOM, which allows React to make changes to the UI without manipulating the actual DOM. This makes updates faster, as changing the virtual DOM is less expensive than changing the real DOM.
- The reconciliation algorithm works by comparing the current virtual DOM tree to the updated virtual DOM tree, and making the minimum number of changes necessary to bring the virtual DOM in line with the updated state.
- The algorithm uses two main techniques to optimize updates:
    - *Tree diffing*: React compares the current virtual DOM tree with the updated virtual DOM tree, and identifies the minimum number of changes necessary to bring the virtual DOM in line with the updated state.
    - *Batching*: React batches multiple changes into a single update, reducing the number of updates to the virtual DOM and, in turn, the real DOM.
- The reconciliation algorithm is a critical part of React's performance and helps make React one of the fastest and most efficient JavaScript libraries for building user interfaces.
- After the reconciler compares the current and updated virtual DOM, it identifies the differences and makes the necessary changes to the virtual DOM to bring it in line with the updated state.

7: What is `React Fiber`?

- React Fiber is a concept of ReactJS that is used to render a system faster, smoother and smarter.
- React Fiber is a backwards compatible, complete rewrite of the React core. In other words, it is a reimplementation of older versions of the React reconciler.
- Introduced from React 16, Fiber Reconciler is the new reconciliation algorithm in React.

- Fiber brings in different levels of priority for updates in React. It breaks the computation of the component tree into nodes, or 'units' of work that it can commit at any time. This allows React to pause, resume or restart computation for various components.
- Fiber allows the reconciliation and rendering to the DOM to be split into two separate phases: Phase 1: Reconciliation and Phase 2: Commit.
- Since Fiber is asynchronous, React can:
    - Pause, resume, and restart rendering work on components whenever required.
    - Increases the suitability of the React library to create animations, layouts, and gestures.
    - Reuse previously completed work and even abort it if not needed.
    - Split work into chunks and prioritize tasks based on importance.

8: Why do we need `keys` in React? When do we need keys in React ?

- `Key` is a special attribute you need to include when creating lists of elements in React.
- Keys helps React identify which items in the list have changed, are added, or are removed.
- In other words, we can say that keys are unique Identifier used to give an identity to the elements in the lists.
- The best way to pick a key is to use a string that uniquely identifies a list item among its siblings.
- When you don't have stable IDs for rendered items, you may use the item index as a key as a last resort.

Example

Recursing On Children
By default, when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there's a difference.

For example, when adding an element at the end of the children, converting between these two trees works well:

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

React will match the two <li>first</li> trees, match the two <li>second</li> trees, and then insert the <li>third</li> tree.

If you implement it naively, inserting an element at the beginning has worse performance. For example, converting between these two trees works poorly:

```
<ul>
  <li>Duke</li>
  <li>Villanova</li>
</ul>

<ul>
  <li>Connecticut</li>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

React will mutate every child instead of realizing it can keep the <li>Duke</li> and <li>Villanova</li> subtrees intact. This inefficiency can be a problem.
In order to solve this issue, React supports a key attribute. When children have keys, React uses the key to match children in the original tree with children in the subsequent tree. For example, adding a key to our inefficient example above can make the tree conversion efficient:

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

```
<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

Now React knows that the element with key '2014' is the new one, and the elements with the keys '2015' and '2016' have just moved.

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

Yes, we can use the `index as keys`, but React Developers don't recommend using indexes for keys 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 however when you don't have stable IDs for rendered items, you may use the item index as a key as a last option.

`` `NO key << INDEX as key <<<<<< Unique id as key from data` ``

10: What is `props in React`? Ways to use props ?

- Props stand for "Properties." They are read-only components. It is an object which stores the value of attributes of a tag and work similar to the HTML attributes. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
- Props are immutable so we cannot modify the props from inside the component. Inside the components, we can add attributes called props. These attributes are available in the component as props and can be used to render dynamic data in our render method.
- Props are used to store data that can be accessed by the children of a React component. They are part of the concept of reusability. Props take the place of class attributes and allow you to create consistent interfaces across the component hierarchy.

- Props act as a channel for component communication. Props are passed from parent to child and help your child access properties that made it into the parent's tree.

Every parent component can pass some information to its child components by giving them props. Props are similar to HTML attributes, but you can pass any JavaScript value through them, including objects, arrays, and functions.

Types of Props :

- Familar Props ——> HTML attributes like className, src, width, height passed in HTML tag
- Passing Props to Component ——> props are the only argument to your component. React component functions accept a single argument, a props object.

Ways to pass props to component	Ways to receive the props in another component
1. Add props to the JSX, just like you would with HTML attributes	All props are sent into a single props object
`<Profile name = { "Hello"} age={28} />`	`const Profile = (props) => { let name = props.name; let age = props.age; }`
2. Similar to the way mentioned in 1.	Props object can be destructed using {} to receive only the required props
`<Profile name = { "Hello"} age={28} />`	`const Profile = ({name, age}) => { }`
3. Using spread syntax	And props objects destructed using {}
`<Profile {...props} />`	`const Profile = ({name, age}) => { }`

However, props are immutable which means unchangeable. When a component needs to change its props (for example, in response to a user interaction or new data), it will have to “ask” its parent component to pass it different props—a new object! Its old props will then be cast aside, and eventually the JavaScript engine will reclaim the memory taken by them.

11: What is `Config Driven UI`?

- Config-driven UI is one of the UI design pattern in which the UI is rendered based on the configuration parameter sent by the server (backend). This is one of the popular pattern used in the industry now.
- `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.`
- Config driven UI not only make your code cleaner but also easier to maintain.
- You can any time change/add the validations in the config (Like maxValue , or add a regex pattern, etc..) and change the renderer function to accept those keys.

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

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). Diff 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 quickly
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
It is not light-weight.	It is light-weight abstraction of DOM.