React

- **React** is a JavaScript library for building user interfaces or UI components.

Why React

- React is Simple to Read and Easy to Use
- React also allows us to create reusable UI components
- React allows developers to create large web applications that can change data, without reloading the page.

Features of React

- It uses VirtualDOM instead of RealDOM considering that RealDOM manipulations are expensive.
- Supports server-side rendering.
- Follows Unidirectional data flow or data binding.
- Uses reusable/composable UI components to develop the view.

render()

- The ReactDOM.render() function takes two arguments, HTML code and an HTML element.
- Example: - ReactDOM.render(<h1> Hello</h1>,document.getElementById('root'))
- Here < h1>Hello</h1> is the component which we want to append in our root component

JSX

- JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement() and/or appendChild() methods.
- JSX converts HTML tags into react elements.
- By using JSX, we can write HTML structures in the same file that

contains JavaScript code.

- it just provides syntactic sugar for the React.createElement()

DOM

- **DOM** stands for "**Document Object Model**". The DOM in simple words represents the UI of your application. Every time there is a change in the state of your application UI, the DOM gets updated to represent that change.

virtual DOM

- Virtual DOM is a Memory representation of Actual DOM.
- React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, virtual DOM changes only that object in the real DOM, rather than updating all the objects.
- React maintains **two Virtual DOM** at each time, one contains the updated Virtual DOM and second one which is just the pre-update version of this updated Virtual DOM.

Rendering

- Rendering is the process of react which asking the component to what the changes in UI will look like based on current combination of props and state.

https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-interview-questions#reactjs_interview_questions_on_components

HTML event V/S React event

- In HTML the name of events should be in lowercase, whereas react follow camel Casing.
- In HTML, you need to invoke the function by appending () Whereas in react you should not append () with the function name.
- In HTML, you can return false to prevent default behaviour, Whereas in React you must call preventDefault() explicitly:

Components

- Components are the building blocks of any React application, and a single app usually consists of multiple components. It splits the user interface into independent, reusable parts that can be processed separately.
- **components** are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

Types of React Components

- Functional Components
- Class Components
- Pure Components
- Higher-Order Components

functional components

- A **Functional component** is a way to define a **React component**. It is just a plain JavaScript function that accepts props as an argument and returns a React element

class components

- A **class component** is a way to define a **React component**. a class component

requires you to extend from React. **Class component** create a render function which returns a React element.

pure components

- **Pure Components** in React are the **components** which do not re-renders when the value of state and props has been updated with the same values. If the value of the previous state or props and the new state or props is the same, the **component** is not re-rendered.
- React.PureComponent is exactly the same as React.Component except that it handles the shouldComponentUpdate() method for you.

Higher Order Components

- The purpose of a HOC is **to enhance a component with extra functionality**. A HOC allows for reusability of components.
- A higher-order component acts as a container for other components. This helps to keep components simple and enables re-usability. They are generally used when multiple components have to use a common logic. (***)

Controlled Component

- A component that controls the input elements within the forms on subsequent user input is called Controlled Component, i.e. every state mutation will have an associated handler function.

Uncontrolled Components

- The Uncontrolled Components are the ones that store their own state internally, and you query the DOM using a ref to find its current value when you need it.

difference between pure component and component

- The **difference between** them is that React. **Component** doesn't implement

shouldComponentUpdate(), but react. **PureComponent** implements it **with a** shallow prop and state **comparison**.

Difference between Class & functional Components

- Class Components Can hold or manage state, on the other hand functional components are not holding any state they use useState Hook to manage states in their components.

- document.getElementById('load-posts').addEventListener('click', loadPosts);

-

- function loadPosts() {

- // Create a new XMLHttpRequest object

- var xhr = new XMLHttpRequest();

-

- // Configure it: GET-request for the URL

- xhr.open('GET', 'https://jsonplaceholder.typicode.com/posts', true);

-

- // Set up a function to handle the response data

- xhr.onload = function() {

- if (xhr.status >= 200 && xhr.status < 300) {

- // Parse JSON response

- var posts = JSON.parse(xhr.responseText);

- var output = '<h2>Posts</h2>';

-

- // Iterate through the posts and create HTML for each

```
            posts.forEach(function(post) {
                output += `
                    <div>
                        <h3>${post.title}</h3>
                        <p>${post.body}</p>
                    </div>
                `;
            });

            // Insert the HTML into the DOM
            document.getElementById('posts').innerHTML = output;
        } else {
            console.error('Request failed. Returned status of ' + xhr.status);
        }
    };

    // Handle network errors
    xhr.onerror = function() {
        console.error('Request failed');
    };

    // Send the request
```

- xhr.send();

- }Class Components are complex as compared with stateless component, while functional components are Simple and easy to understand.

- Class components Can work with all lifecycle methods. While functional components can't use lifecycle method, we can use useEffect Hook to perform lifecycle method in functional components

React Components are usually re-rendered when:

- setState() is called
- props values are updated
- forceUpdate() is called

Stateful & Stateless Component

- In React, a stateful component is **a component that holds some state**.

- Stateless components are **those components which don't have any state at all**, which means you can't use setState inside these components. It is like a normal function with no render method. It has no lifecycle, so it is not possible to use lifecycle methods such as **componentDidMount** and other hooks.

- Note that both types of components can use props.

Props

- **Props**" is a special keyword in **React**, which stands for properties and is being used for passing data from one component to another. But the important part here is that data with **props** are being passed in a uni-directional flow that means we can only pass the props through parents' component to its immediate child components.

State

- **State** is an object that holds some information of a component that may change over the lifetime of the component. The state can be modified but using only setState()

method.

- State is immutable is required setState method to update its value.

Why not update state directly?

- If you try to update state directly then it won't re-render the component.

difference State & Props

- **Props** are used to pass data, whereas **state** is for managing data.
- **State** data can be modified by its own component, whereas **Props** modify by it's on components.

Props drilling

- **Prop Drilling** is the process by which you pass data from one part of the React Component tree to another by going through the child component that do not need the data but only help in passing it around.

Memo

- React. memo is a higher order component.
- Using memo will cause React to skip rendering a component if its props have not changed.
- This can improve performance.

Difference between useMemo & memo

useMemo: - This is used when you want to memorize your variable/state

memo: - This is used when you want to memorize your component

export default memo(your component name)

Hooks

- Hooks are the new feature introduced in the React 16.8 version. It **allows you to use state and other React features without writing a class**.
- useState, useEffect, useContext, useRef, useReducer, useCallback, useMemo, Custom Hooks
    - useState: - useState is a Hook that allows you to have state variables in functional components.
    - useEffect: -
        - The useEffect Hook allows you to perform side effects in your components.

- Some examples of side effects are: fetching data, directly updating the DOM, and timers.

- useEffect accepts two arguments. The second argument is optional. useEffect(<function>, <dependency>)

- useContaxt: - React Context is a way to manage state globally. React's **useContext** hook makes it easy to pass data throughout your app without manually passing props down the tree.

- useRef: - The useRef Hook **allows you to persist values between renders**. It can be used to store a mutable value that does not cause a re-render when updated. It an be used to access a DOM element directly.

- useReducer: - The useReducer hook should be used in components that have complex logic behind it.

  The reducer function contains your custom state logic and the initialState and returns the current state and a dispatch method.

Side Effect

- Side effects are **all the operations that affect your component and can't be done during rendering**. Things like fetching data, subscriptions or manually changing the DOM are all examples of side effects

Lifecycle Methods

Lifecycle methods are **special methods built into React, used to operate on components throughout their duration (lifecycle/ until they get unmount from DOM) in the DOM**. For example, when the component mounts, renders, updates, or unmounts.

- getInitialState(): This is executed before the creation of the component.

- componentDidMount(): Is executed when the component gets

rendered and placed on the DOM.

- shouldComponentUpdate(): Is invoked when a component determines changes to the DOM and returns a “true” or “false” value based on certain conditions.

- componentDidUpdate(): Is invoked immediately after rendering takes place.

- componentWillUnmount(): Is invoked immediately before a component is destroyed and unmounted permanently.

Debouncing implementation

```
useEffect(() => {
        let myValue = setTimeout(runFunction, 1000)
        return () => {
        clearTimeout(myValue)
        }
},[])
```

key

- A key is a special string attribute you should include when creating arrays of elements. Key prop helps React identify which items have changed, are added, or are removed.

- Using indexes for keys is not recommended if the order of items may change. This can negatively impact performance and may cause issues with component state.

Ref

- The ref is used to return a reference to the element. They can be useful when you need a direct access to the DOM element or an instance of a component.

className/class

- class is a keyword in JavaScript, and JSX is an extension of JavaScript. That's the principal reason why React uses className instead of class.

Profiling

- FCP (first content-ful paint)
- LCP
- Network throttling

Stack trace

Phases in react

- Render phase
    - The virtual DOM is constructed
    - Find differences between virtual DOM
    - Browser DOM is not modified
- Commit phase
    - Apply the differences
    - Reconciliation is a part of commit phase
    - Browser DOM is modified

Reconciliation

Profiler API

https://reactjs.org/docs/profiler.html

Synthetic event

In order to work as a cross-browser application, React has created a wrapper same as the native browser in order to avoid creating multiple implementations for multiple methods for multiple browsers, creating common names for all events across browsers. Another benefit is that it increases the performance of the application as React reuses the event object.

It pools the event already done hence improving the performance.

Syntax:

- e. preventDefault() prevents all the default behaviour by the browser.
- e.stopPropagation() prevents the call to the parent component whenever a child component gets called.

- **bubbles:** Return true or false indicates event is a bubbling event or not.
- **cancelable:** Return true or false indicates if the event can be canceled or not.
- **currentTarget:** Indicates the element to which the handler is attached
- **defaultPrevented:** Return true or false, indicates whether the event has been canceled by preventDefault().
- **eventPhase:** Returns number, indicates the phase
- **isTrusted:** Return true when the event is generated by the user and false when by the browser/script
- **type:** Returns string, it indicates the type of the event

References

Optimization: - (https://www.codementor.io/blog/react-optimization-5wiwjnf9hj)

interview questions: - (https://iq.js.org/questions/react/what-is-react)

https://www.notion.so/arfat/16th-July-2022-390e347e207f4d7e86defefc9dce68f3

https://www.notion.so/arfat/17th-July-2022-fcb48834ac58420a88978812ee8fac73