**React** is a **declarative, efficient, and flexible JavaScript library** **(not a framework)** for building user interfaces.

It’s ‘**V’** in **MVC**.

ReactJS is an **open-source, component-based front-end library** responsible only for the **view layer** of the application.

It is **maintained** by **Facebook**.

React is used to build **single-page applications (SPA).**

Facebook Software Engineer, **Jordan Walke**, created it in **2013.**

React Component  
A React application is made of multiple components, each responsible for rendering a small, reusable piece of HTML.

Components can be nested within other components to allow complex applications to be built out of simple building blocks. A component may also maintain an internal state.

# Why React

While building client-side apps, a team of Facebook developers realized interacting with the **DOM is slow**.

*(The Document Object Model (****DOM****) is an application programming interface (****API****)* ***for HTML and XML documents****. It defines the* ***logical structure of documents*** *and the way a document is* ***accessed*** *and* ***manipulated****.).*

React Working

To make DOM interaction faster, React implements a **virtual DOM** in **memory** that is basically a **DOM tree representation in JavaScript Object**. So, when it needs to read or write to the browser DOM, it will use the virtual representation of it. Then the virtual DOM will try to find the most **efficient** way to update the browser’s DOM.

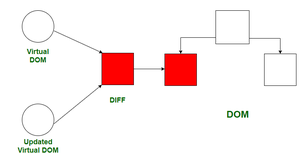
# ReactJS Reconciliation

**Reconciliation is the process through which React updates the Browser DOM.**

Important concepts behind the working of the Reconciliation process are:

1. Virtual DOM
2. Diffing Algorithm

* 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. The cool fact is all these comparisons take place in the memory and nothing is yet changed in the Browser.
* After comparing, React goes ahead and creates a new Virtual DOM having the changes. It is to note that as many as 200,000 virtual DOM nodes can be produced in a second.
* Then it updates the Browser DOM with the least number of changes possible without rendering the entire DOM again. This changes the efficiency of an application tremendously.



# Two ways to add react to project

1. Including CDN’s in HTML, adding script tags in HTML documents, 3 scripts (react script, react-dom script & Babel).
2. By setting up the react environment using npx and npm (using Node.js).

**Problem 1:** Browser don’t understand the JSX

**Solution:** JSX, ES6, ES5 => Babel => Pure JS/ Vanilla JS => Browser Understand

Babel transpile the code (Software that converts the source code of one language into the source code of another).

***Note*** Babel is commonly used for both front and back-end.

**Problem 2:** Browser don’t understand the import statements

**Solution:** webpack bundle all files (multiple js files, .png, .css files …) into single-single file (bundle.js, .png, .css …)

# React Execution flow

Src Folder

Public Folder

Index.html

Components

App.js

App.css

Index.js

Index.css

# Differences between npm and npx:

|  |  |
| --- | --- |
| **npm (node package manager)** | **npx(node package execute)** |
| If you wish to run package through npm then you have to specify that package in your package.json and installed it locally. | A package can be executable without installing the package, it is a npm package runner so if any packages that aren’t already installed it will be installed automatically. |
| To use create-react-app in npm the commands are ***npm install create-react-app*** then ***create-react-app myApp***(Installation required). | But in npx you can use that without installing like ***npx create-react-app myApp***, this command is required in every app’s life cycle only once. |
| Npm is a tool that use to install packages. | Npx is a tool that use to execute packages. |
| Packages used by npm are installed globally you have to care about pollution for the long term. | Packages used by npx are not installed globally so you have to carefree for the pollution for the long term. |

| **Named export** | **Default export** |
| --- | --- |
| export function User() {...} | export default function User() {...} |
| import {User} from ... | import User from ... |

# JSX (JavaScript XML)

JSX provide cleaner code of JavaScript in react (syntactic sugar).

JSX allows us to write HTML & JS in React.

**Rule: A component can only return a single element**

JSX rule-

* All HTML must include a parent/root element.
* All HTML attributes and CSS properties must be named using camelCase.
* JavaScript code must be wrapped in curly braces {} inside JSX.
* All HTML must close properly (including empty elements)

Use a "**fragment**" to wrap multiple lines.

This will prevent unnecessarily adding extra nodes to the DOM.

A fragment looks like an empty HTML tag: <></> | <React.Fragment></React.Fragment>

The class attribute is a much-used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

Use attribute className instead.

# Render Function

The ReactDOM.render() function takes two arguments, HTML code and an HTML element.

The purpose of the function is to display the specified HTML code inside the specified HTML element.

Conditional Rendering

* and operator [expression && expression]

(avoid using because it may cause problem (0 && something) => 0)

* ternary operator [expression? value: value;]

# Components

Components are **independent, isolated** and **reusable bits of code**. They serve the same purpose as JavaScript functions, but work in **isolation** and return HTML via a render or return function.

Class components were the only way to track state and lifecycle on a React component. Function components were considered "state-less".

With the addition of Hooks, Function components are now almost equivalent to Class components. The differences are so minor that you will probably never need to use a Class component in React.

### **Class Component**

A class component must include the extends React.Component statement. This statement creates an inheritance to React.Component, and gives your component access to React.Component's functions.

The component also requires a render() method, this method returns HTML.

class Car extends React.Component {

render() {

return <h2>Hi, I am a Car!</h2>;

}

}

### **Function Component**

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand

function Car() {

return <h2>Hi, I am a Car!</h2>;

}

**Note- ClassName should be upper camel case because in JSX it denotes component Instance and not an HTML element.**

***States are mutable and props are immutable***

# Props

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

React Props are like function arguments in JavaScript and attributes in HTML.

props stand for properties.

**Note:** React Props are read-only! You will get an error if you try to change their value.

# State

React components has a built-in state object.

The state object is where you store property values that belongs to the component.

When the state object changes, the component re-renders.

***React follows unidirectional data flow model | top to bottom | parent to child component***

# Hooks

Hooks allow function components to have access to state and other React features (lifecycle methods).

## **Hook Rules**

1. Hooks can only be called inside React function components. i.e., Hooks will not work in React class components.
2. Hooks can only be called at the top level of a component.
3. Hooks cannot be call inside if statements, loops, function etc. (No nesting nor conditional).
4. Hooks must be executed in exact same order.

**useState**

The React useState Hook allows us to **track state** in a function component.

State generally refers to data or properties that need to be tracking in an application.

Two return value = 1. Value (it denotes the current state of the component) 2. Function (to change the state)

Function should always take the previous value before altering the value.

Function overrides all previous values in useState().

While calling the function to update the state multiple time in a single call it only runs once because of batching as every time function get call it re render which cause the performance issue of the app to avoid it react override all the function calls with last call.

To overcome this always take the previous value setX(prevX => prevX+1)

If arrow function is passed inside useState() parameter it only runs once else if normal function is passed it runs each and every time thus leads in creating bulky software.

**useEffect**

The useEffect Hook allows you to perform **side effects** in your components whenever anything gets changes.

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>)

two parameters 1. function 2. array of arguments [arg1, arg2] on what value change we need to take care

[] => changes on mount

[parameter1, parameter2 …] => changes if changes occur in parameter value.

Function must contain the return for cleanup process and improve performance. return () => {}

**useMemo**

Using memo will cause React to skip rendering a component if its props have not changed.

Used for caching, memoization

**useContext**

React Context is a way to **manage state globally**.

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

Used to avoid prop drilling, to access a DOM element directly.

Flow createContext() => Provider => Consumer/ useContext.

**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 can be used to provide **reference to HTML element** (DOM element) as every html element consist of ref attribute.

Lifecycle Methods

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases. Mounting, Updating, and Unmounting.

**Mounting**

Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

The render() method is required and will always be called, the others are optional and will be called if you define them.

**Updating**

The next phase in the lifecycle is when a component is updated.

A component is updated whenever there is a change in the component's state or props.

React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

The render() method is required and will always be called, the others are optional and will be called if you define them.

**Unmounting**

The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.

React has only one built-in method that gets called when a component is unmounted:

1. componentWillUnmount()

constructor: to run before loading the page

Mounting: if loading/rendering for component for first time (componentDidMount(){})

useEffect(()=>{}, []) with no parameter

Updating: if updating component (componentDidUpdate(){by comparing previous state and new state })

useEffect(()=>{}, dependency) with parameter for dependency

Unmounting: if removing certain component (componentWillUnmount(){})

useEffect(()=>{return () => {}}, dependency)

Render: to render component

A screenshot of a computer

Description automatically generated with medium confidence

Lazy Loading

Code splitting functionality

The goal of lazy loading is to get faster initial load times.

Users only load the code they need for the features they are using. This leads to fast load times and more efficient use of resources.