What is ReactJs ?

ReactJS is a popular JavaScript library used for building user interfaces, particularly for single-page applications where data can change over time without reloading the page.

ReactJS History

When compared to other technologies on the market, React is a new technology. Jordan Walke, a software engineer at Facebook, founded the library in 2011, giving it life.

## Why React?

* Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it requires less coding and offers more functionality, as opposed to JavaScript, where coding often gets complex very quickly.
* Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components.
* Small learning curve: React is easy to learn, as it mostly combines [basic HTML](https://www.simplilearn.com/tutorials/html-tutorial/what-is-html" \t "_blank" \o "basic HTML) and JavaScript concepts with some beneficial additions. Still, as is the case with other tools and frameworks, you have to spend some time to get a proper understanding of React’s library.
* It can be used for the development of both web and mobile apps.

## Features of React

React offers some outstanding features that make it the most widely adopted library for frontend app development. Here is the list of those salient features.

### JSX



JSX is a JavaScript syntactic extension. It's a term used in React to describe how the user interface should seem. You can write HTML structures in the same file as JavaScript code by utilizing JSX.

### Virtual Document Object Model (DOM)

The Virtual DOM is React's lightweight version of the Real DOM. Real DOM manipulation is substantially slower than virtual DOM manipulation. When an object's state changes, Virtual DOM updates only that object in the real DOM rather than all of them.

### Install Node.js and npm

Before you start, ensure that Node.js and npm (Node Package Manager) are installed on your machine. You can download and install them from [nodejs.org](https://nodejs.org/" \t "_new).

npm stands for Node Package Manager. It is a package manager for JavaScript and is the default package manager for Node.js, a runtime environment for executing JavaScript code outside of a browser.

### What is npm used for?

1. **Package Management**: npm is primarily used for installing, managing, and updating packages or libraries of JavaScript code. These packages can range from small utility libraries to large frameworks like React or Express.
2. **Dependency Management**: In JavaScript projects, especially those using Node.js, dependencies are managed through npm. When you create a new project or work on an existing one, you can define which external libraries your project relies on by listing them in a package.json file.
3. **Version Control**: npm helps in managing versions of installed packages. Each package in npm has a version number, and you can specify which versions of packages your project requires to ensure compatibility and stability.
4. **Command-Line Interface**: npm provides a command-line interface (CLI) that allows developers to interact with npm from the terminal or command prompt. This CLI lets you install packages, update them, uninstall them, and perform various other tasks related to package management.

### Basic npm Commands:

* **npm install <package-name>**: Installs a package locally in your current project.
* **npm install -g <package-name>**: Installs a package globally on your machine.
* **npm uninstall <package-name>**: Removes a package from your project.
* **npm update <package-name>**: Updates a package to its latest version.
* **npm init**: Initializes a new package.json file interactively.
* **npm start**: Runs the start script defined in package.json.
* **npm run <script-name>**: Runs a custom script defined in package.json.

### package.json File:

The package.json file is a manifest file for Node.js projects and is essential when using npm. It contains metadata about the project and configuration details, including:

* Project name, description, and version.
* List of dependencies (packages required by your project) and their versions.
* Scripts to run (e.g., start, test) and other project-specific configurations.

### node\_modules Directory:

When you install packages using npm, they are stored in a directory called node\_modules within your project. This directory is managed by npm and should not be manually modified. It contains all the dependencies required by your project, organized in nested directories based on the package structure.

### Usage in Frontend and Backend Development:

* **Frontend (Browser)**: npm is commonly used in frontend development to manage JavaScript libraries and frameworks (e.g., React, Vue.js) and build tools (e.g., webpack, Babel).
* **Backend (Node.js)**: In backend development using Node.js, npm manages server-side libraries and frameworks (e.g., Express, Sequelize) and utilities for handling database operations, HTTP requests, and more.

npx is a command-line tool that comes with npm version 5.2.0 and higher. It stands for Node Package Runner and is used to execute Node.js packages without installing them globally. Here's a breakdown of what npx does and why it's useful:

### Purpose of npx:

1. **Run Node.js Packages**: npx allows you to run Node.js packages (or any executable available via npm) directly without having to install them globally. This is particularly useful for running packages that you use occasionally or for one-off tasks.
2. **Execute Local Commands**: Instead of manually managing and updating globally installed packages, npx lets you execute commands from locally installed packages. This keeps your project dependencies isolated and prevents version conflicts.
3. **Temporary Usage**: npx executes the command in a temporary environment, ensuring that it doesn't interfere with your global npm packages or project dependencies.

### Key Features and Usage:

* **Command Syntax**: You can use npx with a command followed by a package name
*  **Latest Version**: npx by default installs the latest version of the package specified, unless a specific version is specified.
*  **Local Packages**: If a package is already installed locally in your project (in the node\_modules/.bin directory), npx will use that version instead of downloading it again.
*  **Running Scripts**: npx is often used to run scripts defined in package.json files.

### Advantages of npx:

* **Version Management**: npx simplifies package management by automatically handling the installation and execution of packages, ensuring you're always using the latest version or the version specified in your project's package.json.
* **Developer Workflow**: It promotes a cleaner and more controlled development environment by reducing global installations and ensuring dependencies are managed within project boundaries.

What is State ?

**State** refers to an object that holds information about the components' current state or data. It is managed within the component and can be updated over time in response to user actions, network responses, or any other events. State allows React components to create dynamic and interactive user interfaces.

### Key Points about State in ReactJS:

1. **Component-Specific**: Each component in React can maintain its own state as an object. This state is local and cannot be accessed or modified by other components unless explicitly passed down as props.
2. **Mutable**: Unlike props, which are immutable and passed down from parent to child components, state can be updated using the setState() method provided by React.
3. **Initialization**: State is typically initialized in the constructor of a class component or using the useState hook in a functional component.
4. **Updating State**: In functional components using hooks, useState hook provides a setter function to update the state.
5. **Asynchronous Updates**: State updates in React are asynchronous for performance reasons. If you need to update the state based on the current state, you should use the function form of setState()
6. **Rendering**: When the state of a component changes, React automatically re-renders the component, updating the UI to reflect the new state.

### When to Use State:

* **Dynamic Data**: Use state to manage data that can change over time, such as user input in forms, data fetched from an API, or toggling UI elements (like showing/hiding a modal).
* **Component Interaction**: State is useful for managing the local state of a component that isn't shared with other components. For shared state between components, consider using props and lifting state up to a common ancestor component.

What is Hooks ?

In ReactJS, hooks are special functions that let you "hook into" React state and lifecycle features from function components. Hooks allow you to use state and other React features without writing a class. They were introduced in React 16.8.

useState()

useState is a Hook that allows you to add state to a functional component. Before the introduction of Hooks in React (before version 16.8), state could only be used in class components. Hooks provide a way to use stateful logic in functional components.

· **Syntax:** const [state, setState] = useState(initialState);

· state: The current value of the state.

* setState: A function to update the state. When called, it schedules a re-render with the updated state value.
* initialState: The initial value of the state, used only during the first render.

· **Multiple** useState **Hooks:** You can use useState multiple times in a single component to manage different pieces of state independently.

useEffect()

React that allows functional components to perform side effects. Side effects in this context refer to operations such as fetching data, manipulating the DOM, subscribing to external events, or any other operation that isn't directly related to rendering the UI.

· **Asynchronous:** useEffect functions are asynchronous and do not block the browser’s painting process.

· · **Multiple** useEffect **Hooks:** You can use multiple useEffect hooks in a single component to separate different side effects.

· · **Replacing Lifecycle Methods:** useEffect can replace lifecycle methods (componentDidMount, componentDidUpdate, componentWillUnmount) in class components.

· · **Dealing with Effects:** Use useEffect for data fetching, subscribing to events, updating the DOM, or any other side effect.

**Props**

Props are a fundamental concept used for passing data from one React component to another. Props are read-only.

### Benefits of Props

* **Component Reusability:** Props facilitate the creation of reusable components that can be customized with different data.
* **Component Composition:** Props enable composition of components into larger, more complex UIs.
* **Data Flow Control:** Provides a clear and predictable way to manage data flow in React applications.

**Props Drilling**

Prop drilling refers to the process where props are passed from a parent component down to deeply nested child components through intermediary components that do not actually use the prop themselves. This happens when data needs to be passed through multiple levels of nested components.

**Context API**

The Context API is a feature that allows you to share data between components without having to explicitly pass props through every level of the component tree. It's designed to solve the problem of prop drilling, where props are passed from a parent component through intermediary components to reach deeply nested child components.

Steps to use Context API:

1. Create
2. Provider
3. Consumer

**useContext()**

The useContext Hook in React allows functional components to consume context that has been created with the React.createContext API. It provides a more concise way to subscribe to a context compared to using the Consumer component from the Context API.

Steps to use Context API:

1. Create
2. Provider
3. useContext

**useMemo()**

useMemo is a Hook in React that is used for memoization. Memoization is a technique used in programming to optimize performance by caching the results of expensive function calls and returning the cached result when the same inputs occur again. In the context of React, useMemo memoizes the result of a function so that the function is only re-executed when one of its dependencies has changed.

**useCallback()**

useCallback is a React Hook that is used to memoize functions. It is similar to useMemo, but instead of memoizing a computed value, useCallback memoizes a function. This is particularly useful when passing callbacks to child components that rely on reference equality to prevent unnecessary renders.

**Note:**

**useMemo()** and **useCallback()** is similar but useMemo() returns the memoized value and useCallback() returns the memoized function.

One reason to useCallback() is to prevent a component re-rendring unless its props have changed.

**Memo**

memo is a higher-order component in React that is used to memoize functional components. It works similarly to useMemo, but for functional components instead of values. When a component is wrapped with memo, React memoizes the component and re-renders it only if its props change

### Differences

* **Purpose:** useMemo is used to memoize values or computations, while memo is used to memoize functional components.
* **Usage:** useMemo is used inside functional components to memoize values, while memo wraps functional components to memoize their rendering.
* **Dependencies:** useMemo depends on an array of dependencies to determine when to recompute, while memo depends on props changes to determine when to re-render the wrapped component.

### When to Use

* **useMemo:** Use when you need to memoize values derived from props or state, especially for expensive computations or calculations.
* **memo:** Use when you need to optimize the rendering of functional components that depend on props and do not need to re-render on every parent render.

Redux:

Store:

* Entire application contains single store.
* It is responsible for holding application state
* getState() method gives access to state it holds.
* Dispatch(action) allow state to be updated.
* It has subscriber(listener) method as well by which we can register listeners.

This method accept function (listener) as a parameter which execute anytime when the state in redux store changes.

Reducer:

Reducer decides how the state of application changes depending upon the actions sent to the store.

Reducers are the functions that accepts state and action as a parameter and returns the next state of the application.

(previousState,action)=> newState

Action

* Action are javascript object that contains information.
* Actions are only source of information for the store. It only tells us what will happen.
* Actions have a “type” property and it should be defined in string constraints
* It is compulsory to include “type” property in the object.

Suntax:

Const action ={

Type = ‘ ’

}

Reducer

* Reducer describes how the state of application changes depending upon the action sent to the store.
* Reducers are the functions that accepts state and action as parameter and returns the next state of application.

Store:

* Application Contains single store.
* It is responsible for holding application State.
* getState() method gives access to state it holds.
* Dispatch(action) method all state to be updated.
* It has subscriber(listener) method as well by which we can register listeners.

This method accept function (listener) as a parameter which execute anytime when the state in redux store changes.