**What is React**

* React is a **JavaScript** library created by **Facebook**
* React is a **User Interface** (UI) library
* React is a tool for building **UI components**

**React** is a **JavaScript library** for building **user interfaces** (UIs) on the web. React is a declarative, component based library that allows developers to build reusable UI components and It follows the [Virtual DOM](https://www.geeksforgeeks.org/reactjs-virtual-dom/) (Document Object Model) approach, which optimizes rendering performance by minimizing DOM updates. React is **fast** and works well with other tools and libraries.

**What is NPM?**

NPM is a package manager for Node.js packages, or modules if you like.

[www.npmjs.com](https://www.npmjs.com/) hosts thousands of free packages to download and use.

The NPM program is installed on your computer when you install Node.js.

* The command **npm** is used to download JavaScript packages from [Node Package Manager](https://www.npmjs.com/),
* **npx** is used to execute JavaScript packages downloaded this way.
* This command will download the NPM package [create-react-app](https://www.npmjs.com/package/create-react-app) to a subdirectory of the current working directory named node\_modules:

**npm install create-react-app**

* This command will execute the NPM package create-react-app with the name argument myreactapp, creating a bare-bones React app in the subdirectory myreactapp:

**npx create-react-app ProjectName**

**What is NPX?**

The npx stands for **Node Package Execute** and it comes with the npm, when you installed npm above 5.2.0 version then automatically npx will installed. It is an npm package runner that can execute any package that you want from the npm registry without even installing that package. The npx is useful during a single time use package. If you have installed npm below 5.2.0 then npx is not installed in your system.

* You can check npx is installed or not by running the following command:

**npx -v**

**What is a Package?**

A package in Node.js contains all the files you need for a module.

Modules are JavaScript libraries you can include in your project.

**What is vite?**

Vite is an extremely fast and lightweight web application build tool. It stands out for its fast startup and instant compilation times, thanks to the use of real-time module loading (ESM) during development. Vite supports React, Vue. js, and other popular frameworks.

React is exclusively a UI library. **Vite is a JS bundler()**. Vite takes your React JSX code and transforms it into raw HTML, CSS, and JS files that can be run by most browsers. Those files are then hosted and served to end users.

* Create a new Vite Project

//it is fast as compare to **npx create-react-app**

**npm create vite@latest**

**npm create vite@latest projectName**

To run project – **npm run dev**

**npm install**

**What is createRoot() method ?**

reactDOM.createRoot use to create new container in virtual DOM like you want to inject some component dynamically you can use is like this.

* It also compare it own DOM and our DOM , it will only update that thing that is only in UI.
* But our browser remove complete DOM and repaint(web structure is going to ready again) the DOM again(it is also called page reload).
* But in virtual DOM we can trace DOM in tree like structure or only update those value that is changed or updated.

**React JS Virtual DOM**

React JS Virtual DOM is an in-memory representation of the DOM. DOM refers to the Document Object Model that represents the content of XML or HTML documents as a tree structure so that the programs can be read, accessed and changed in the document structure, style, and content.

**What is React Fragment?**[**​**](https://refine.dev/blog/how-react-fragments-is-works/#what-is-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.

To return multiple elements from a React component, you'll need to wrap the element in a root element.

Eg: <> ….. </>

**Note**:

* Either we are using vite or react we should always start function/component name With uppercase **eg: Function(), Myfun().**
* **To get variable value to print:**

We will use curly braces {}

const username = "Abdul Azeem"

Eg: {VariableName} -> expression (evaluated expression).

My name is {username}.

**What are React Hooks?**

**React Hooks**provide [**functional components**](https://www.geeksforgeeks.org/reactjs-functional-components/)with the **ability to use states and manage side effects**. They were first introduced in **React 16.8**, and allow developers to hook into the state and other React features **without having to write a class**. They provide a cleaner and more concise way to handle state and side effects in React applications.

Although Hooks generally replace class components, no plans exist to remove classes from React.

One of the most well-known React hooks is the useState() hook. It lets you add a state variable to your component. The useState() hook can conveniently hold **strings, arrays, numbers, objects** and much more.

* If we want to update variable in react , react allow to us to update variable but if we want to update the variable in UI (To show on browser) then react will handle that , so the concept of hooks comes here (hooks give us some method to update tha data on UI).

We can say react control the UI updation.

**React useState hook**

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.

**Import useState**

To use the useState Hook, we first need to import it into our component.

**import { useState } from "react";**

**How to write hooks**

* Hook is nothing but a function and it return values (eg. [state, setState]).

const [variable, function] = useState(initial values goes here)

const [state, setState] = useState(initial values goes here)

const [calories, setCalories] = useState(initial value of calories).

**React Fibre**

<https://github.com/acdlite/react-fiber-architecture>

**What is reconciliation?**

***reconciliation***

The algorithm React uses to diff (differentiation algorithm) one tree with another to determine which parts need to be changed.

Reconciliation is the algorithm behind what is popularly understood as the "virtual DOM." A high-level description goes something like this: when you render a React application, a tree of nodes that describes the app is generated and saved in memory. This tree is then flushed to the rendering environment — for example, in the case of a browser application, it's translated to a set of DOM operations. When the app is updated (usually via setState), a new tree is generated. The new tree is diffed with the previous tree to compute which operations are needed to update the rendered app.

* It is a algorithm which compare two tree.

***update***

A change in the data used to render a React app. Usually the result of `setState`. Eventually results in a re-render.

**The key points are:**

* In a UI, it's not necessary for every update to be applied immediately; in fact, doing so can be wasteful, causing frames to drop and degrading the user experience.
* Different types of updates have different priorities — an animation update needs to complete more quickly than, say, an update from a data store.
* A push-based approach requires the app (you, the programmer) to decide how to schedule work. A pull-based approach allows the framework (React) to be smart and make those decisions for you.

**What is a fiber?**

We're about to discuss the heart of React Fiber's architecture. Fibers are a **much lower-level abstraction** than application developers typically think about. If you find yourself frustrated in your attempts to understand it, don't feel discouraged. Keep trying and it will eventually make sense. (When you do finally get it, please suggest how to improve this section.)

* We've established that a primary goal of Fiber is to **enable React to take advantage of scheduling**. Specifically, we need to be able to
* pause work and come back to it later.
* assign priority to different types of work.
* reuse previously completed work.
* abort work if it's no longer needed.

In order to do any of this, we first need a way to break work down into units. In one sense, that's what a fiber is. A fiber represents a **unit of work**.

**What is props?**

Props are used to makes component reusable.

It means make a card and put it in the component and we can use that component how many times we want.

Eg: 1

function Card(props){

    console.log(props.username);

    return(

    )

}

Eg: 2

function Card({username}){ // destructuring

    console.log(username);

    return(

    )

}

* **We can take components from here -** [**DevUI**](https://www.devui.io/components)

**Custom hook**

* When we create custom hooks file we name it with .js extension.
* we didn't name it as .jsx bcz hooks return purely javascript so we use .js extension.
* Custom hooks can also use built in hooks (eg. useEffect, useState, useCallback).

function useCurrencyInfo (currency){

    useEffect(() =>{}, [])

* ()=>{} it is callback function
* [] it is dependency array

}

**useId hook**

useId is a React Hook for generating unique IDs that can be passed to accessibility attributes.

const id = useId()

**Reference**

**useId()**

Call useId at the top level of your component to generate a unique ID:

import { useId } from 'react';

function PasswordField() {

const passwordHintId = useId();

// ...

**Parameters**

useId does not take any parameters.

**Returns**

useId returns a unique ID string associated with this particular useId call in this particular component.

**Caveats**

* useId is a Hook, so you can only call it **at the top level of your component** or your own Hooks. You can’t call it inside loops or conditions. If you need that, extract a new component and move the state into it.
* useId **should not be used to generate keys** in a list. [Keys should be generated from your data.](https://react.dev/learn/rendering-lists#where-to-get-your-key)

You can then pass the generated ID to different attributes:

<>

<input type="password" aria-describedby={passwordHintId} />

<p id={passwordHintId}>

</>

**Note**: While looping always remember to pass an attribute called as "key" to increase the performance of iterations in React