**React Revision**

* **Origins and Founder**
* React was created by Jordan Walke, a software engineer at Facebook, in 2011. The initial prototype, called "FaxJS," was inspired by Facebook’s earlier work on XHP, an HTML component system for PHP.
* React was first used internally at Facebook for the News Feed and then on Instagram in 2012. It was open-sourced to the public at JSConf US in May 2013, which led to rapid adoption by developers worldwide.
* Jordan Walke’s innovation was later developed and maintained by the broader team at Facebook (now Meta) and an active open-source community.
* **Why Was React Created? (When JavaScript Already Existed)**
* While anything you can do in React *could* be achieved with vanilla JavaScript, React arose out of a need to better handle the *growing complexity* and *performance bottlenecks* Facebook engineers faced in large-scale apps:
* **Complex UI State:**As Facebook's interfaces grew (like News Feed and Ads), managing the interaction logic and state changes directly with JavaScript and libraries like jQuery became increasingly messy, error-prone, and hard to maintain.
* **Performance Issues:**Manual DOM manipulation (as with traditional JavaScript or jQuery) slowed down apps, especially when many parts of the page updated frequently or in unpredictable ways.
* **Cascading Updates:**Developers often struggled to keep the UI consistent as changes in one part could cause unintended side effects elsewhere. The codebase became challenging to update and scale.
* **How React Solves These Problems**
* **Component-Based Architecture:**React encourages breaking the UI into reusable, independent pieces (components), making interfaces easier to reason about, maintain, and scale.
* **Virtual DOM:**Instead of updating the browser DOM directly every time something changes, React first updates a lightweight, in-memory representation (the Virtual DOM). It then calculates the minimal real DOM changes needed for better performance.
* **Declarative Style:**Rather than imperatively telling the browser *how* to update the UI, you declare *what* you want the UI to look like for a given application state, and React takes care of efficiently applying the changes.
* **Improved State Management**: React and its ecosystem (like Redux) make it much easier to predict and manage UI changes as data evolves, which was a major pain point for developers using only JavaScript.
* **Why to learn React?**
* Makes easy to manage & build complex frontend.
* React is the JavaScript Library.
* **Topics in react?**
* Core of React (state or UI manipulation, JSX)
* Component Reusability
* Reusing of Component (Props)
* How to propagate change (Hooks)
* Router
* State management – Redux, Redux toolkit, Zu stand, Context Api
* Class based component (old concept)
* BAAS Apps (backend as a service) eg: firebase
* **Road Map :   
  HTML – CSS – JAVASCRIPT – REACTJS - NEXTJS - ANGULAR-JS**

**How to install react?**

* npx create-react-app appname
* npm vite@latest
* **npx create-react-app appname**

Bootstraps a React app with Webpack as the build tool.

Hides most configuration unless you "eject".

Widely used and reliable, with many tutorials and resources.

Development server and build times are slower, especially for larger apps.

Limited to React only.

* **npm create vite@latest (or npm init vite@latest)**

Bootstraps a frontend app with Vite, which is framework-agnostic (supports React, Vue, Svelte, etc.).

Uses native ES modules and esbuild for rapid cold starts and Hot Module Replacement (HMR), resulting in much faster dev experience.

Offers an open and easily editable configuration from the start.

The build process for production is highly optimized with Rollup.

Designed for modern browser and JavaScript standards, making it more "future-proof".

- [Note : Install node module folder if you are using vite – npm install ]

* **Reconciliation in Simple Words**
* When something changes in a React app (like data or user interaction), React rebuilds a virtual copy of the UI (called the Virtual DOM).
* It then compares the old and new virtual DOMs using a "diffing" algorithm to spot what actually changed.
* Only those changed pieces are updated in the real browser DOM. This makes updates faster and avoids unnecessary changes.
  + It mainly checks:
* If an element's type changed, React replaces it completely.
* If only props or state changed, React updates only those parts.
* For lists, keys help React track which items moved, were removed, or added.

**\* What is Fiber in React?**

* **Fiber** is the name of React's reconciliation engine introduced from React 16 onwards.
* Fiber lets React break large UI updates into small chunks called "units of work," so the browser can handle animations and user input even during big updates.
* Fibers are like tiny data structures that let React pause, resume, or cancel rendering work for smooth user experience.
* Thanks to Fiber, React apps feel more responsive, especially during heavy updates or animations.
* **Reconciliation** is how React decides what needs to change in the UI, and **Fiber** is the advanced technology React uses to make those changes quickly and without freezing the app.

**createRoot(document.getElementById('root')).render()**  
tells React to take control of the HTML element with the id "root" and display your React app inside it.

**Breaking It Down**

* **createRoot(document.getElementById('root'))**:
  + Finds the element in your HTML with id="root".
  + Prepares React to manage this section of the page, called the "root container".
  + This is how React connects itself to your webpage.
* **.render()**:
  + Decides what should appear in that part of the page (usually your main React component, like <App />).
  + Displays your React UI inside the selected HTML element.
  + Any updates to your React components will be shown inside this container.

**Why It Matters**

* This line is the official way (since React v18) to start any React project.
* It lets React update just the "root" part of your page, making things interactive and dynamic without affecting the rest of your HTML.
* Improves performance and supports new features like concurrent rendering in React.

So, this single line connects your React app to your webpage and manages how and where your UI gets shown and updated.

* It tells React: “Take control of this empty spot in the HTML page (the div with id root), and put my React app here.”
* First, createRoot(document.getElementById('root')) finds the empty spot where React should work.
* Then, .render() is used to show your React content (like <App />) inside that spot.

1. The createRoot create's its own DOM and then compare it with the web browser's DOM and only update those components which are actually updated.
2. But the browser removes the whole DOM and then recrates the whole DOM with the updated values this is called reload.
3. However virtual DOM tracks whole DOM like a tree like structure and updates only those values which were only changed.
4. But some values depends on network call so if we update a value it might get update immediately via a network call.
5. So we will have to update it again. To avoid this overhead we can drop the updation calls for the immediate value update.
6. The current algo used by the React is called the React Fibre algo.
7. The algo react uses to differentiate the web browser's tree and React's tree formed through create root is called reconciliation.
8. Reconciliation is the algo behind what popularly known as the Virtual-DOM.
9. UI it is not necessary for every update to be applied immediately.

**Hooks in react**

**1) useState () :**

- useState lets functional components have their own **state** (data that changes over time).  
 - State allows components to remember information between renders, like input values, counters, etc., making the UI interactive.  
 - const [value, setValue] = useState(initialValue);

* initialValue: The starting value for the state.
* value: The current state.
* setValue: The function to update the state.

- example

import React, {useState} from 'react';

function Counter () {

const [count, setCount] = useState (0);

return (

<div>

<p>Count: {count}</p>

<button onClick={() => setCount (count +1)}>Increase</button>

</div>

);

}

**2) useeffect ():**

- useEffect lets components run **side effects**, like fetching data, changing the DOM, or starting/stopping timers, after rendering

- To interact with things outside React (API calls, event listeners), or to do things at specific moments in the component's lifecycle (like mounting or unmounting).

- useEffect (() => {

// side effect

return () => {

// cleanup (optional)

};

}, [dependencies]);

* First parameter: A function containing the code you want to run.
* Second parameter: Dependency array. The effect runs again if these values change.
* Cleanup function: (optional) Runs before component unmounts or before effect re-runs.

- example :

useEffect(() => {

    passwordGenerator()

  }, [length, numberAllowed, charAllowed, passwordGenerator])

**3) useCallback () :**

- useCallback helps to **memoize** (save) a function so it is not created again unless its dependencies change.

- useCallback is useful when passing functions as props to child components, or optimizing components to prevent unnecessary re-renders.

- const memoizedFunction = useCallback (() => {

// your function logic

}, [dependencies]);

* First parameter: The function to memoize.
* Second parameter: Dependency array. The function changes only if dependencies change.

- example

const copyPasswordToClipboard = useCallback (() => {

    passwordRef.current?select();

    passwordRef.current?.setSelectionRange(0, 999);

    window.navigator.clipboard.writeText(password)

  }, [password])

**4) useref ():**

- useRef returns a **ref object** that stays the same between re-renders. It can store a value or a reference to a DOM element.  
- To keep a mutable value that won’t cause re-render, or to keep a reference to a DOM element directly for things like input focus or measuring size

- const myRef = useRef(initialValue);  
 - example

My git hub code of password generator.