Framework vs Library

* **Library**: Called within the code and is replaceable. Example: Choosing between Formik or React Hook Form.
* **Framework**: Provides a complete environment to write code and cannot be easily replaced.

Why React Looks Like a Framework?

Although React is a library, it feels like a framework because:

* It provides a comprehensive ecosystem with packages and tools.
* Features like routing, testing, and state management require importing additional libraries, making React flexible but not fully self-contained.

What is React ?

React is a **JavaScript library** developed by Facebook for building user interfaces, particularly for **single-page applications**.

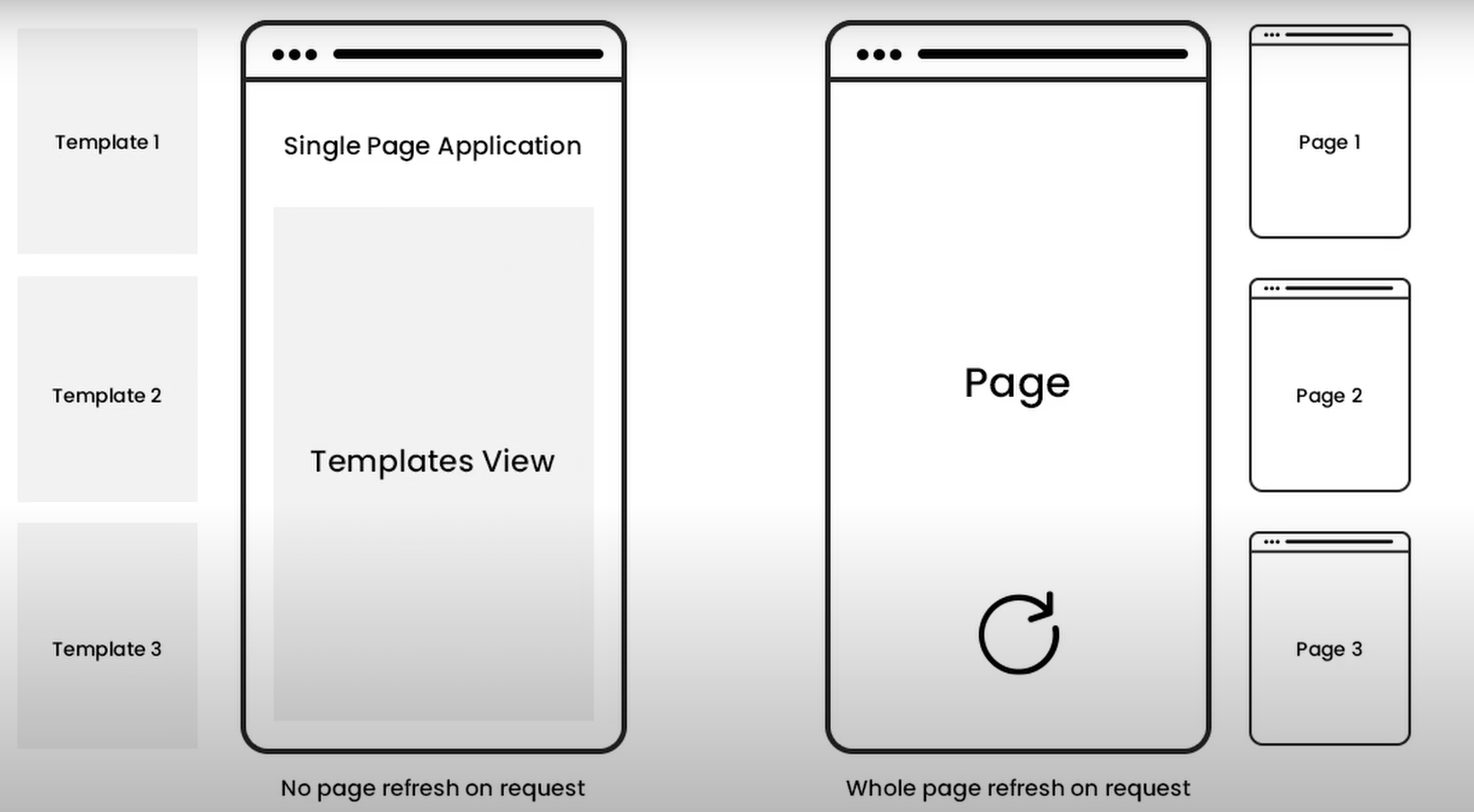
* **Component-Based Architecture**: Allows developers to create reusable UI components.
* **Declarative Programming**: Developers specify what to do, and React handles how to do it.
* **Virtual DOM**: React uses a Virtual DOM for efficient rendering and updating of components.

Single Page Applications (SPAs) are not SEO-friendly. In Multi-Page Applications (MPAs), SEO crawlers have multiple pages to explore, improving indexing. However, in SPAs, content is dynamically loaded, which limits Google's ability to index the pages effectively.

How react is a single page application?

A Single Page Application (SPA) is a web application that loads a single HTML page and dynamically updates the content of that page in response to user interactions. In a traditional web application, when the user interacts with the application, the browser sends a request to the server, which responds by sending a new HTML page. This process results in a full page reload, which can be slow and disrupts the user's experience.

In contrast, React applications use JavaScript to manipulate the content of the existing HTML page, allowing for a faster and more seamless user experience. Instead of requesting new pages from the server, React applications retrieve data from APIs and update the content of the page dynamically, without the need for a full page reload.



The lifecycle of an Effect

Every React component goes through the same lifecycle:

* A component ***mounts*** when it’s added to the screen.
* A component ***updates*** when it receives new props or state, usually in response to an interaction.
* A component ***unmounts*** when it’s removed from the screen.

What are hooks?

Hooks in React are functions that allow you to use state, lifecycle methods, and other features in functional components. They simplify code, enhance reusability, and enable managing side effects, context, and more, without needing class components.

Hooks make functional components more powerful and reduce the need for class-based components.

With the help of hooks we can achieve lifecycle methods in functional component.

UseState Hook

**useState Hook is** a special function that **allows you to manage state**within functional components without the need for class-based components.

useState returns an array with exactly two values:

* The current state. During the first render, it will match the initialState you have passed.
* The set function that lets you update the state to a different value and trigger a re-render.

**setState** is a React method used to update the component's state and trigger a re-render.

* **Callback in setState**: It ensures that some code runs after the state update is complete, especially when the state update is asynchronous.

UseEffect Hook

The useEffect hook in React is used to handle **side effects** in functional components. It allows you to run code after the component has rendered, making it useful for tasks like data fetching, timers, or DOM manipulation.

useEffect(() => {

    // Side effect logic goes here

    return () => {

        // Cleanup logic (optional)

    };

}, [dependencies]);

* **Effect function**: This is where your side effect code runs.
* **Cleanup function**: This optional return function cleans up side effects like subscriptions or timers when the component unmounts.
* **Dependencies array**: React re-runs the effect if any of the values in this array change.

**UseContext Hook**

useContext simplifies sharing data across components by avoiding prop drilling. It works with React's Context API, where you create a context, wrap components with a Provider to pass data, and use useContext in child components to access it.

**Diff between UseContext and Redux**

useContext is for sharing state locally in specific parts of an app; Redux manages global state across the entire app.

useContext may cause unnecessary re-renders; Redux ensures only affected components update.

Use useContext for simple apps, while Redux is better for large, complex apps with global state needs.

**useDebounce**

It delays the execution of a function until after a specified time has passed since the last event. It reduces unnecessary API calls, improves performance, and is commonly used in search inputs to avoid calling the API on every keystroke.

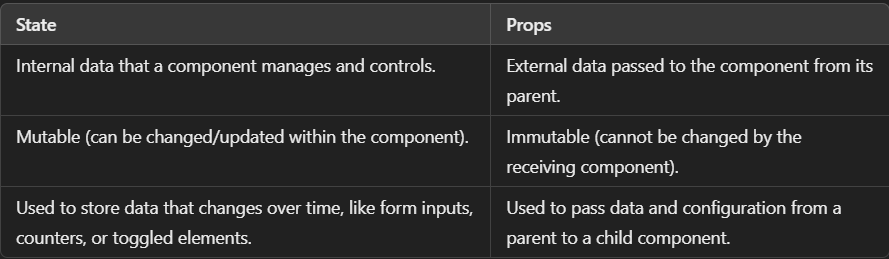
**Example**   
When typing in a search bar, useDebounce waits (like 500ms) after the user stops typing before calling the API. For example, typing "book" triggers one API call after typing is complete, not after each letter. This reduces server load and improves performance.

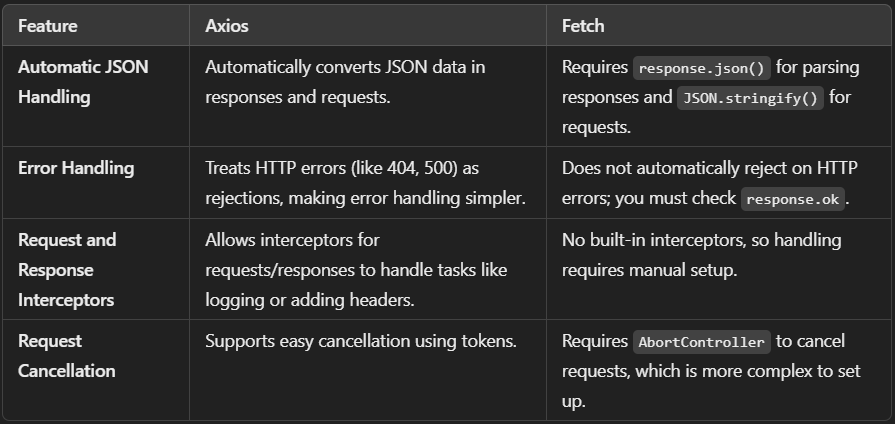
What Is State?

The state is a object that stores properties of the [component.](https://www.simplilearn.com/tutorials/reactjs-tutorial/reactjs-components) Whenever component’s state changes, the component re-renders.

What are **Props?**

Props are arguments passed into React components. **Props in React are the objects used to**pass data/information to a Component. It is responsible for passing the data to components and render the dynamic content.





Prop Drilling

Refers to the process of passing data from a parent component to a deeply nested child component through multiple levels of intermediary components.

How to Prevent Prop Drilling: **Context API**, **State Management Libraries**: Redux

useMemo Hook

useMemo is a React hook that memorizes the result of an expensive computation and recomputes it only when one of its dependencies changes.

const memoizedValue = useMemo(functionThatReturnsValue, arrayDependencies)



**useCallback**

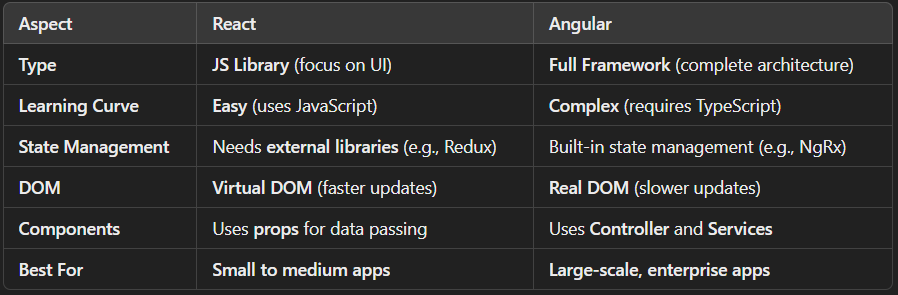
useCallback is a React Hook that lets you cache a function definition between re-renders.

**React useCallback hook** returns a memoized function to reduce unnecessary callbacks. This useCallback hook is used when you have a component in which the child is rerendering again and again without need.



**React vs angular vs vue**

<https://www.youtube.com/shorts/Rz0hNVi1hlQ>

****

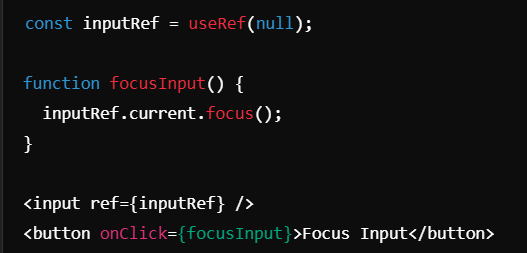
useRef Hook

It can be used to store a mutable value that does not cause a re-render when updated.

useRef  is used to store the values that are not related to render.

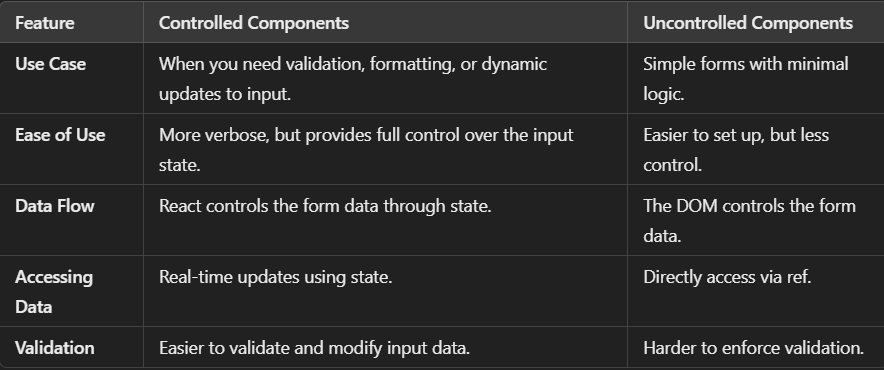
The useRef Hook allows you to persist values between renders.

It can be used to access a DOM element directly.



Controlled vs Uncontrolled Components in ReactJS

<https://dev.to/ale3oula/navigating-reacts-controlled-vs-uncontrolled-components-48kb>



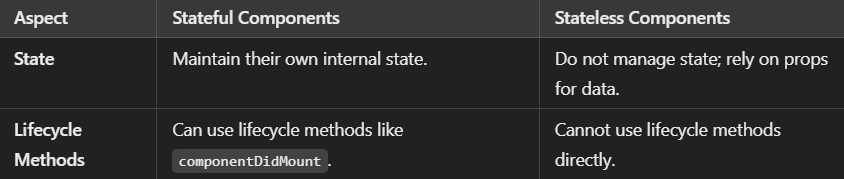
Class vs functional components

Class based has states but functional based has no state, but after react version 16.8 they have introduced states in react.

| **Aspect** | **Function Component** | **Class Component** |
| --- | --- | --- |
| **State** | Uses **Hooks** like useState for state | Uses **this.state** for state management |
| **Lifecycle** | **useEffect** replaces lifecycle methods | Uses methods like **componentDidMount** |
| **Syntax** | **Simple and concise** (less boilerplate) | **More verbose** (needs constructor, this) |
| **Performance** | **Faster** (no instance creation) | **Slower** (requires instance creation) |

***Function components are simpler, modern, and preferred in React. They use hooks like useState and useEffect instead of class-based state and lifecycle methods. Function components have better performance since they don't require instance creation, unlike class components.***

Stateful vs stateless components



React Router DOM

is a library used for routing in React applications. It enables navigation between different components or pages in a single-page application (SPA) without the need for a full page reload.

Main Components: BrowserRouter, Routes, Route

Styling Techniques in React?

**External Stylesheets**: Traditional CSS files linked to components.

import './App.css';

**Inline Styles**: Style applied directly in JSX using the style attribute.

<div style={{ color: 'red' }}>Hello</div>

**CSS Modules**: Locally scoped CSS to avoid conflicts.

import styles from './App.module.css';

<div className={styles.container}>Hello</div>

**Style Libraries (e.g., Tailwind CSS)**: Pre-defined utility classes for styling.

<div className="bg-blue-500 text-white">Hello</div>

How to handle async data fetching?

You can use useEffect along with async functions or use React Query for a more advanced solution.

**keys**

In React, keys are unique identifiers used to help React identify which items have changed, added, or removed in a list. They improve rendering efficiency by minimizing re-renders.

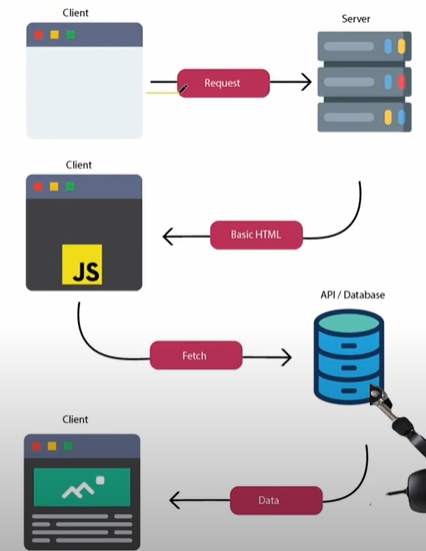
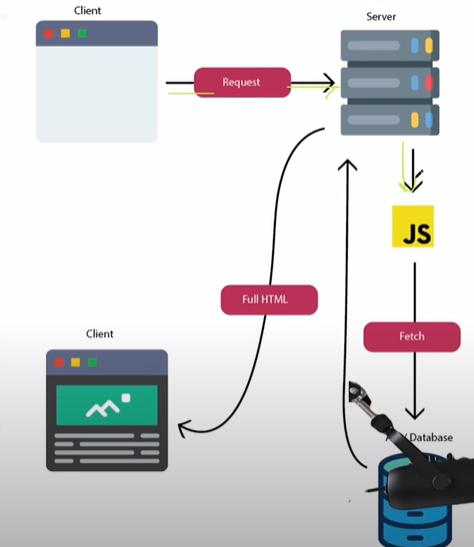
**Using index as key** is possible but not recommended unless:

* The list items are static and do not change.
* The list does not get reordered.

Conditional rendering in React allows rendering different UI elements based on conditions using if-else, ternary operators, logical AND (&&), or switch statements.

CSR: Renders in the browser, slower initial load, fast interactions.

SSR: Renders on the server, faster initial load, better SEO, slower interactions.

Challenges Using React

* **CSR (Client-Side Rendering)** Slower initial load compared to SSR, affecting performance.
* **SEO (Search Engine Optimization)** CSR can make content harder for search engines to index.
* **State Management** Complex state requires external libraries like Redux or Zustand.
* **Handling Complex Forms** Forms with many fields need tools like Formik or React Hook Form.
* **Performance Issues** Frequent updates lead to excessive re-renders.
* **Limitations of Context** Context triggers re-renders for all consuming components on state updates.

How to optimize React app performance?

1. Use React's built in optimizations with React.Memo, Pure Components etc
2. Optimise State Management ( State lifting, redux, code splitting, lazy loading)
3. useMemo, useCallBack

Babel**:** is a JavaScript transpiler used to convert JSX into plain JavaScript.

Higher-Order Component (HOC) in React is a function that takes a component as input and returns a new component with additional functionality.

Synthetic Events in React are **cross-browser wrappers** around native browser events. They standardize event handling across different browsers, providing consistent behavior and better performance.

Lifting the state up is a concept in React where the state is moved from a child component to its parent component in order to share data or manage state between multiple components.

Pure Components in React are components that only re-render when their props or state change. They perform a shallow comparison of the props and state to determine if a re-render is necessary, improving performance by avoiding unnecessary updates.

React.memo is a **higher-order component** that can be used to memoize a functional component. It only re-renders when its props change, making it similar to a Pure Component for functional components.

What is virtual DOM?

The **Virtual DOM** is a lightweight copy of the real DOM. When changes occur, React updates the Virtual DOM first and compares it with its previous version (diffing). It then updates only the changed parts in the real DOM (reconciliation), improving performance and speed.