**useEffect**

If you have worked class components you would have performed side effects in your components like updating DOM, fetching data from endpoint, setting up subscriptions or timers.

Since **render()** method would be too early to perform side effects then you had to make use of the Life Cycle methods.

Example



**Fig:** Updating the document title to the current counter value

**Code purpose:**

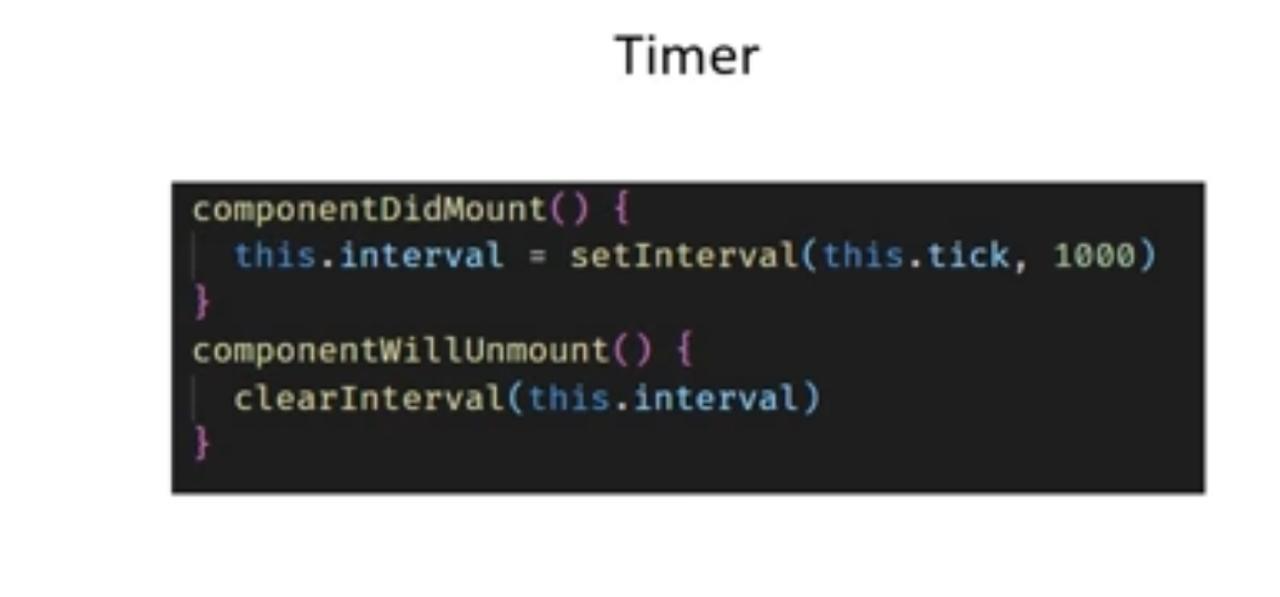
On initial render you want to set **document title = you clicked 0 times**

This code goes in **componentDidMount()** which is executed only once in the component life cycle.

We then, would have a button to increment the “**count**” state value.

But when count value increments we also need to update the **document.title** again, for that we add same code in **componentDidUpdate()** life cycle hook method which is called anytime the component updates.

Another side effect example: **Usage of Timer**



Assume in **componentDidMount()** we set up timer to log to the console, the string “hello” for every 5sec.

And then We want to clear this timer when component is being removed from the DOM. We do this in **componentWillUnmount()** life cycle method.

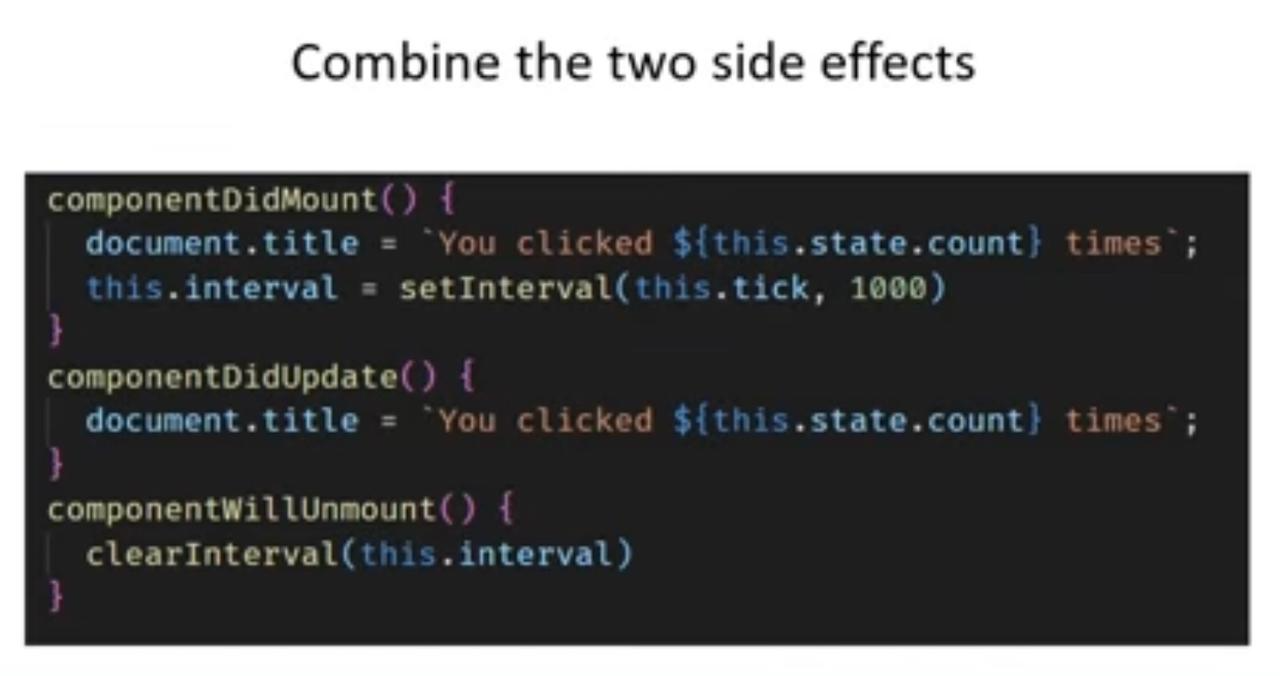
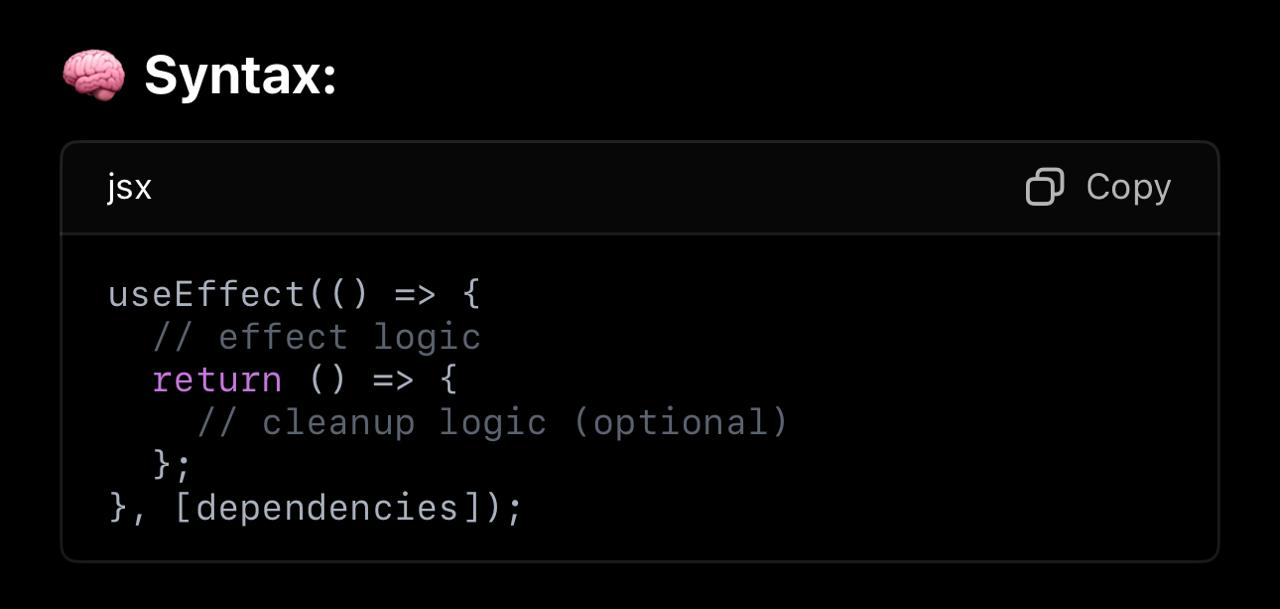


Fig: combine the two side effects

What we see in above code is,

* Same code (document.title) put in two different life cycle methods
* ⁠**setInterval()** and **clearInterval()** which are related methods but put in different life cycle methods.
* The code to update the DOM and setting an interval are completely unrelated code but put together in **componentDidMount()**

⁠Hence useEffect helps to avoid duplicate code and keep only related code together.



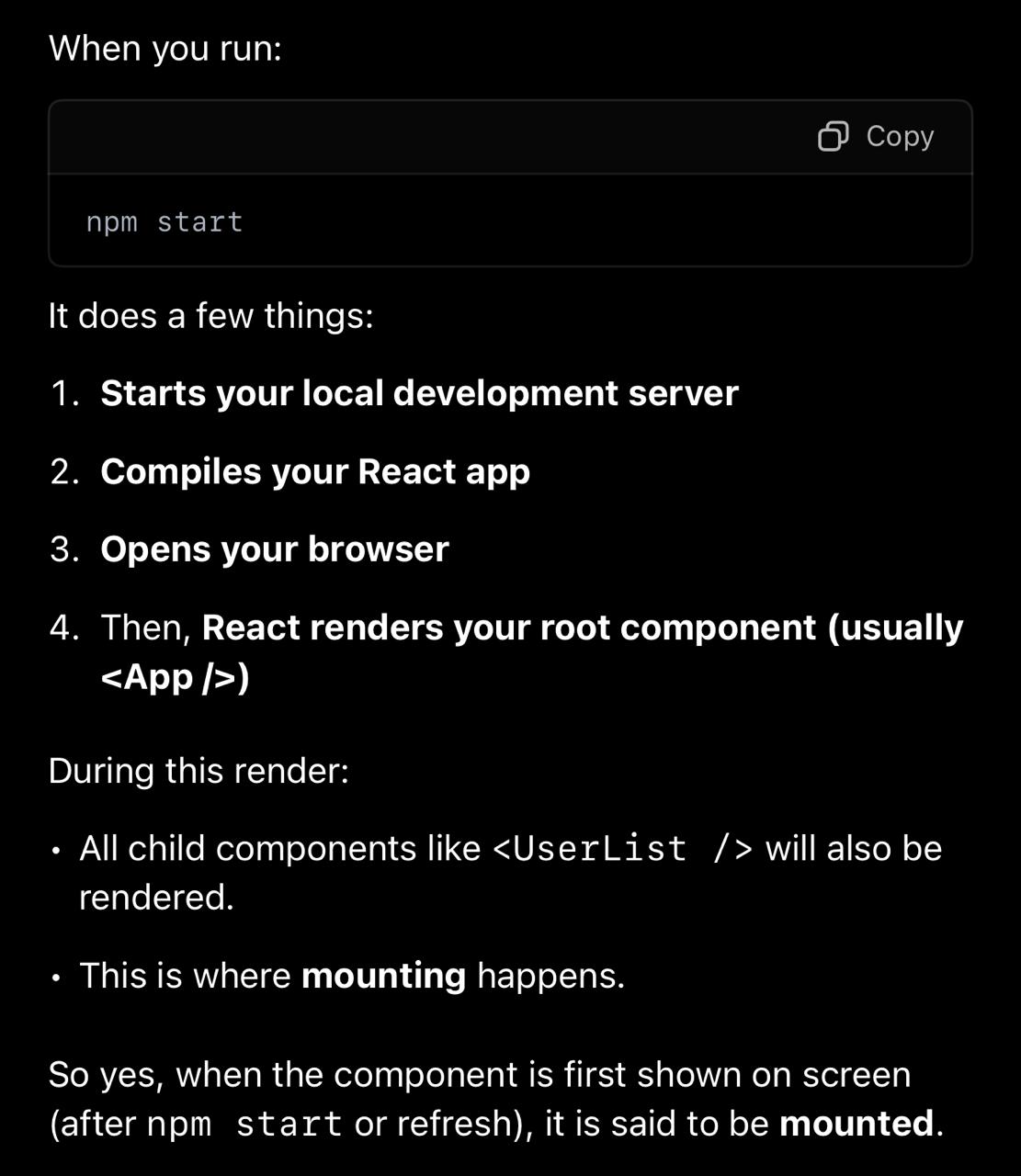
**Remember,** in **useEffect()**, the dependencies array are optional.

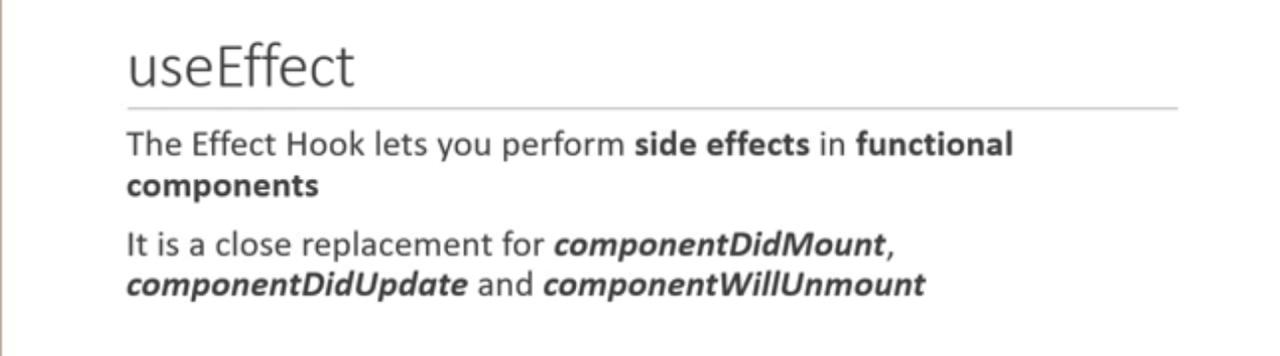
useEffect with no dependency array: Runs after every render. Useful for, debugging and code that should run every time.

useEffect with empty dependency array: [ ] means that this effect runs only once after the first render ( like **componentDidMount()** ). Useful for, Fetching data setting up subscription or timers and initial animations.

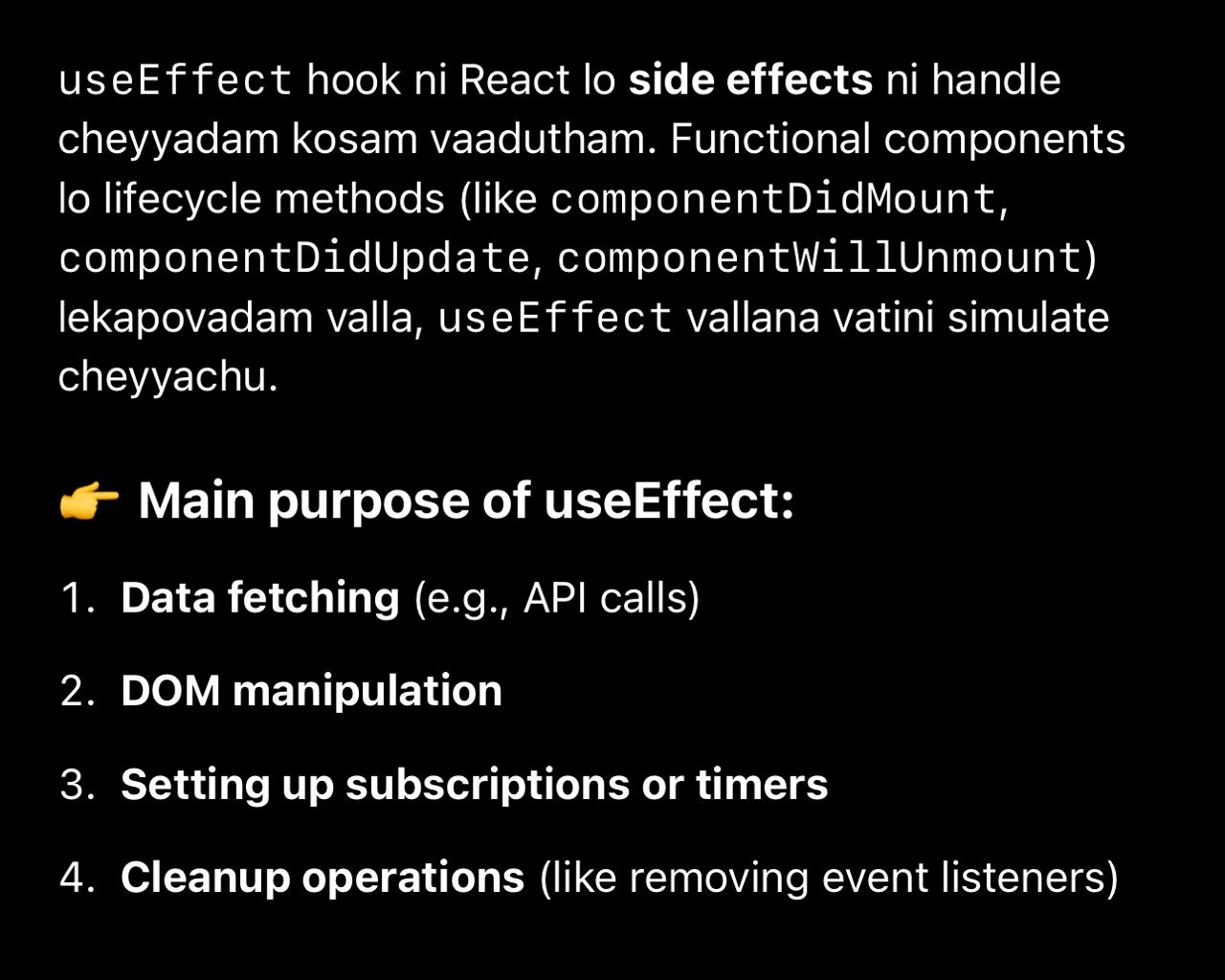
useEffect with dependencies: Runs only when specified state value(s) changes. Usefyl for, watching changes and reacting (like sending analytics, calling APIs etc.)

**NOTE:** component mount means – the moment component is added to the DOM and rendered for the first time like a birth of the component that only happens once per component life cycle.





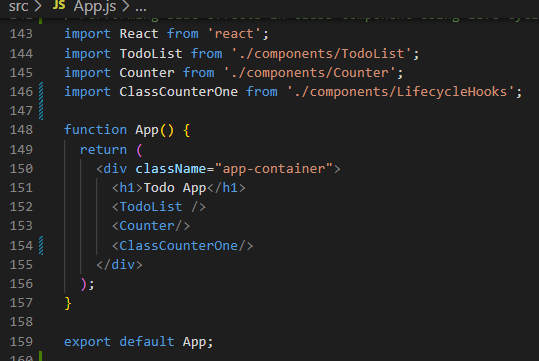
**useEffect** hook ni vaadadam valla, React functional component lo external operations (side effects) ni easy ga handle cheyyachu. For Example, data fetch cheyyadam, timers set cheyyadam, subscriptions create cheyyadam lanti pani ki use chestham.



**Let differentiate class component and functional component which perform side effects:**

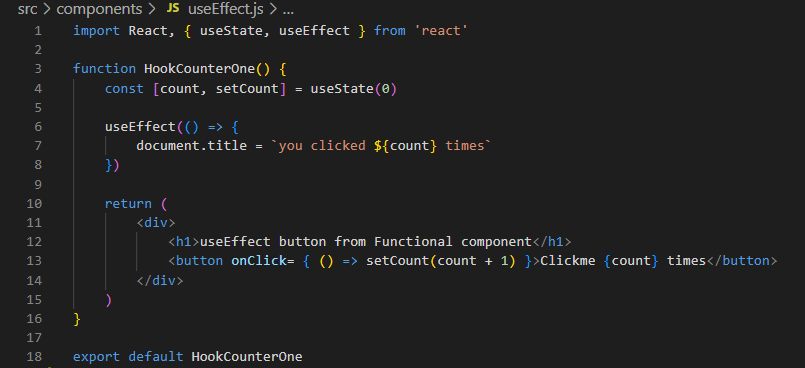
**Create class component:**





Above class component lo count state value update ayina prathisari component re-render avthundi also **componentDidUpdate()** kuda perform avthundi.

Whatever we can do using **componentDidMount**(), **componentDidUpdate(),** and **componentWillUnmount()** in class components, we can handle the same using the **useEffect()** hook in functional components.

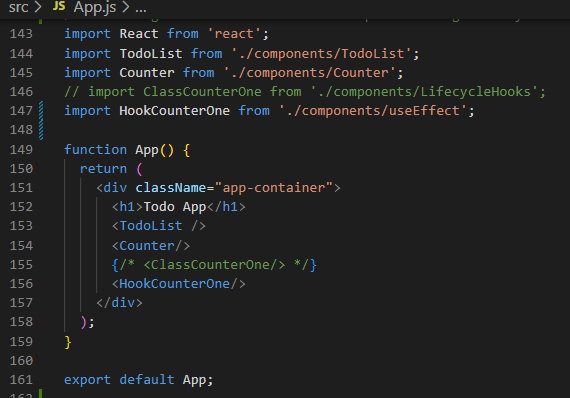
Now see how **useEffect()** hook as a feature can mimic above **componentDidMount()** and **componentDidUpdate()** in functional component. 

**Fig:** Here we run side effects in functional component using React hook **useEffect()**.

In code, with the help of **useEffect()** we change the document title when we click on button “click me” of functional component.

**useEffect()** accept function as a parameter, which gets executed after every render of a component.

Hence, we pass the arrow function to **useEffect()** which updates the document title.



Run npm start and check whether document title is updated on every click of button “**click me**”

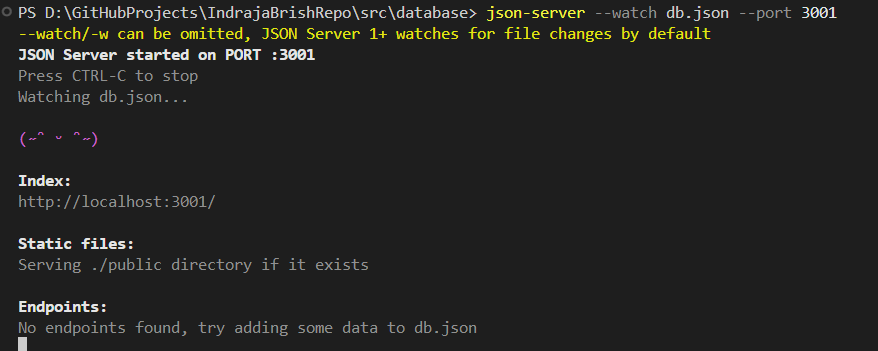
How **useEffect()** works:

When we specify **useEffect()** we are basically request React to execute the function that is passed as an argument every time the component renders.

**Remember,** **useEffect()** runs both after first render of component and after every subsequent render of component.

**More practical useEffect() examples like fetching data from JSON server API:**

1. In your React project directory, create a file db.json.
2. npm install -g json-server
3. json-server --watch db.json --port 3001



1. Your db.json file **must have at least one top-level key**, which represents a "collection"

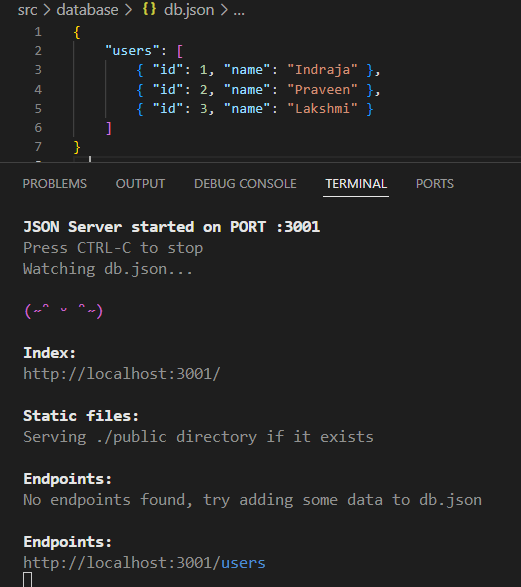


Fig: Add **users** collection which acts as endpoint to JSON API server

1. **Create React functional component which run side effects(API call) using useEffect():**

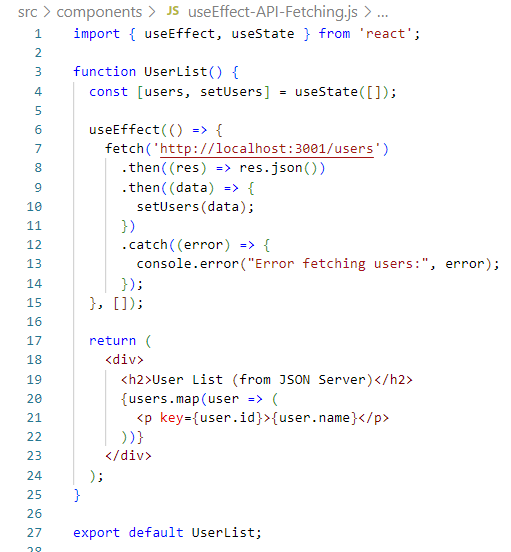
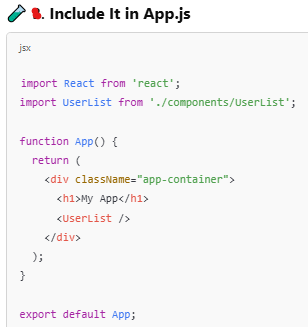


Fig: useEffect runs only once after initial component mounts.



Code Explanation:

* The API call runs only once when the component mounts
* Data is stored in state(users) and displayed.
* We use empty dependency array [ ] so it doesn’t fetch again on every render.

When you run npm start, it:

Starts the React development server.

Opens your React app in the browser at <http://localhost:3000>.

Your **<App/>** starts

<UserList/> component get rendered.

Component function runs then JSX return executes.(**JSX return** is for rendering the UI).

Then, **useEffect()** runs only once after first mount (i.e., after JSX return is completed)because

of empty dependency array.

It sends a GET request to http://localhost:3001/users.

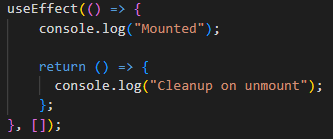
localhost:3001 is your JSON server that you started separately using:

json-server --watch db.json --port 3001

Inside **useEffect()**, it fetches user data and updates state with **setUsers(data).**

React re-renders and displays each user's name in a <p> tag.

**Remember: If any useEffect has return statement like,**

****

useEffect() return runs before **component unmount**(component is removed from DOM) or **effect re-run.**

useEffect() return **Useful**

* when React may unmount your component while the fetch is still in progress. If this happens, calling **setUsers(data)** on an unmounted component will show a warning.
* **Clearing timers**(clearInterval, clearTimeout), **Cancelling subscriptions**, **Aborting fetch requests**, **Avoiding memory leaks on component unmount**.