Lifting Up The State

Lifting state up in React is a common pattern where you move the state from a lower-level component to a higher-level component in the component hierarchy. This allows you to share and manage state across multiple components, making your application more flexible and easier to maintain.

Let's explore the concept of lifting state up with an example:

Suppose you have a simple counter application consisting of two components: Counter and App. The Counter component represents a counter with buttons to increment and decrement the count, while the App component is the top-level component that manages the state of the counter.

Here's how you can implement this:

Create the Counter Component

Start by creating the Counter component that displays the count and has buttons to increment and decrement it. This component doesn't manage its state; instead, it receives the count value and functions as props from its parent.

```
import React from 'react';

function Counter({ count, increment, decrement }) {
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
      <button onClick={decrement}>Decrement</button>
    </div>
  );
}

export default Counter;
```

Create the App Component

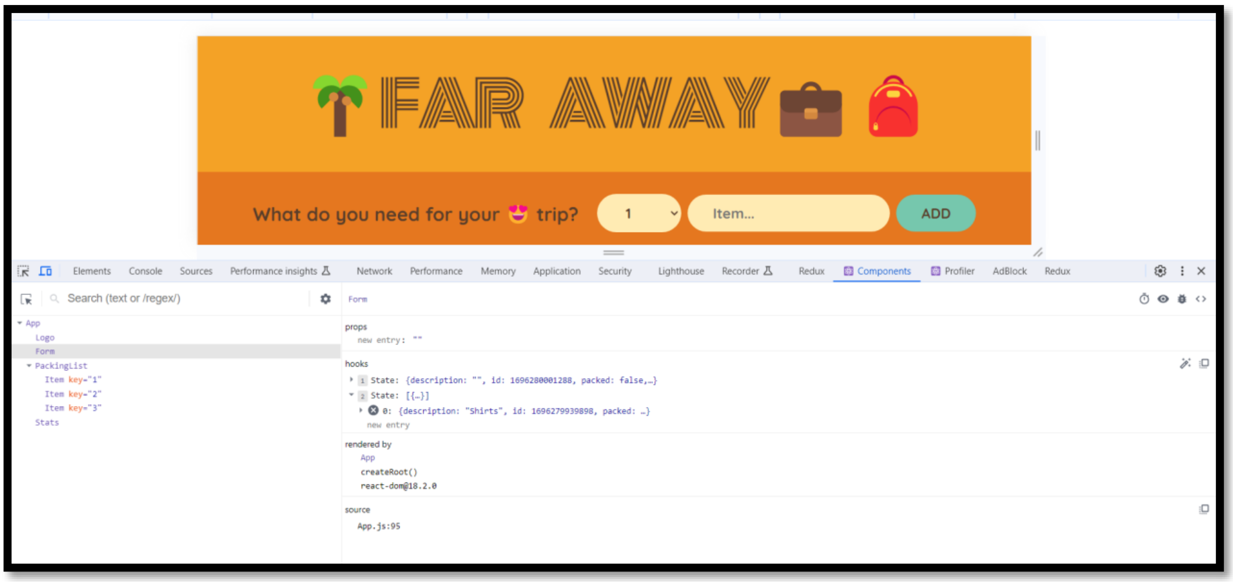
Next, create the App component, which will manage the state of the counter and pass it down to the Counter component as props.

```
import React, { useState } from 'react';
import Counter from './Counter';

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

  const increment = () => {
    setCount(count + 1);
  };

  const decrement = () => {
    setCount(count - 1);
  };

  return (
    <div>
      <h1>Counter App</h1>
      <Counter count={count} increment={increment} decrement={decrement}
/>
    </div>
  );
}

export default App;
```

In this example:

1. The App component manages the state of the count using the useState hook.
2. It defines two functions, increment and decrement, which update the count when called.
3. The Counter component receives the count value and the increment and decrement functions as props and displays them accordingly.

By lifting the state (count and state-modifying functions) from the Counter component to the App component, you centralize the state management, making it easier to share the state and functionality with other components if needed. This pattern is especially useful as your application grows and you need to manage state across multiple components.



```
▾ App
    Logo
    Form
  ▾ PackingList
      Item key="1"
      Item key="2"
      Item key="3"
    Stats
```

```
Form
props
  new entry: ""
hooks
 ▸ 1 State: {description: "", id: 1696280001288, packed: false,…}
 ▾ 2 State: [{…}]
   ▸ 0: {description: "Shirts", id: 1696279939898, packed: …}
     new entry
rendered by
  App
  createRoot()
  react-dom@18.2.0
source
  App.js:95
```

The state is nowhere displayed in UI. So, we are not using this items state variable anywhere in our JSX. The reason for that is we do not need these items in the Form component. We need this in PackingList component.

But we can not pass items as props to PackingList component from Form component. Because PackingList component is not child component of Form component. PackingList component is a sibling component of Form component.

Data can not flow sideways or up the component tree. Data can flow down the component tree.

In such situations we use a mechanism called as Lifting Up the State.

Here, the immediate parent of Form component is App component. Therefore, we will lift the state from Form component to it’s immediate parent i.e. App component.

In summary we can say whenever multiple sibling components need access to the same state, we move that piece od state up to their immediate parent component.