Sure, here are two examples of state management in React using different approaches: local component state and Redux.

1. **Local Component State**:

In this example, we'll create a simple counter component that manages its state locally using the `useState` hook.

```jsx
import React, { useState } from 'react';

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

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

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

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

export default Counter;
```

In this example:

- We use the `useState` hook to initialize the state variable `count` with an initial value of `0`.
- We define `increment` and `decrement` functions that update the `count` state when the corresponding buttons are clicked.
- The `count` state is local to the `Counter` component and is not shared with any other components.

2. **Redux**:

In this example, we'll create a counter component that manages its state using Redux.

First, we need to set up the Redux store and define actions and reducers.

```jsx
// store.js
import { createStore } from 'redux';

const initialState = {
  count: 0
};

const counterReducer = (state = initialState, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return { ...state, count: state.count + 1 };
    case 'DECREMENT':
      return { ...state, count: state.count - 1 };
    default:
      return state;
  }
};

const store = createStore(counterReducer);

export default store;
```

Then, we create a counter component that dispatches actions to update the state.

```jsx
// Counter.js
import React from 'react';
import { useSelector, useDispatch } from 'react-redux';

function Counter() {
  const count = useSelector(state => state.count);
  const dispatch = useDispatch();

  const increment = () => {
    dispatch({ type: 'INCREMENT' });
  };

  const decrement = () => {
```

```
    dispatch({ type: 'DECREMENT' });
  };

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

export default Counter;
```

In this example:

- We use the `useSelector` hook to extract the `count` state from the Redux store.
- We use the `useDispatch` hook to get the `dispatch` function, which is used to dispatch actions to update the state.
- When the `increment` or `decrement` button is clicked, we dispatch the corresponding action to update the `count` state in the Redux store.

These are two examples of state management in React using local component state and Redux. Depending on the complexity and requirements of your application, you can choose the approach that best fits your needs.