Certainly! Below are coding examples demonstrating the usage of the eight React Hooks mentioned previously:

```jsx
import React, { useState, useEffect, useContext, useReducer, useRef, useMemo, useCallback } from 'react';

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

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

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

// useEffect()
const DataFetcher = () => {
  const [data, setData] = useState(null);

  useEffect(() => {
    // Simulating fetching data from an API
    setTimeout(() => {
      setData('Data fetched successfully!');
    }, 2000);
  }, []); // Empty dependency array to execute only on mount

  return <div>{data ? <p>{data}</p> : <p>Loading...</p>}</div>;
};

// useLayoutEffect()
const MeasureWindow = () => {
  const [width, setWidth] = useState(window.innerWidth);

  useLayoutEffect(() => {
    const handleResize = () => {
      setWidth(window.innerWidth);
```

```
    };

    window.addEventListener('resize', handleResize);

    return () => {
      window.removeEventListener('resize', handleResize);
    };
  }, []); // Empty dependency array to execute only on mount

  return <p>Window width: {width}px</p>;
};

// useContext()
const ThemeContext = React.createContext('light');

const ThemedComponent = () => {
  const theme = useContext(ThemeContext);

  return <p>Current theme: {theme}</p>;
};

// useReducer()
const reducer = (state, action) => {
  switch (action.type) {
    case 'increment':
      return { count: state.count + 1 };
    case 'decrement':
      return { count: state.count - 1 };
    default:
      return state;
  }
};

const CounterReducer = () => {
  const [state, dispatch] = useReducer(reducer, { count: 0 });

  return (
    <div>
      <p>Count: {state.count}</p>
      <button onClick={() => dispatch({ type: 'increment' })}>Increment</button>
      <button onClick={() => dispatch({ type: 'decrement' })}>Decrement</button>
    </div>
  );
};
```

```
// useRef()
const TextInputWithFocusButton = () => {
  const inputRef = useRef(null);

  const focusTextInput = () => {
    inputRef.current.focus();
  };

  return (
    <div>
      <input ref={inputRef} type="text" />
      <button onClick={focusTextInput}>Focus Input</button>
    </div>
  );
};

// useMemo()
const MemoizedExpensiveComponent = () => {
  const expensiveComputation = useMemo(() => {
    // Simulating an expensive computation
    let result = 0;
    for (let i = 0; i < 1000000; i++) {
      result += i;
    }
    return result;
  }, []); // Empty dependency array to execute only on mount

  return <p>Result of expensive computation: {expensiveComputation}</p>;
};

// useCallback()
const ButtonWithCallback = () => {
  const handleClick = useCallback(() => {
    alert('Button clicked!');
  }, []); // Empty dependency array to memoize the function

  return <button onClick={handleClick}>Click me</button>;
};
```

These examples demonstrate various React Hooks in action. Each component showcases the usage of a specific hook according to its intended purpose.