React-2

Rendering a List in React

1. Setting Up React

HTML Structure with React CDN:

- Include React and ReactDOM via CDN links.
- Use Babel for JSX support in the browser.
- Basic HTML setup:

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>React List Rendering</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/18.2.0/umd/react.production.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react-dom/18.2.0/umd/react-dom.production.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/7.19.3/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    // Your React code will go here
  </script>
</body>
</html>
```

2. Understanding the `map` Method

`map` Method Overview:

- A JavaScript array method.
- Transforms each element of the array based on a function.
- Returns a new array with transformed elements.

Example:

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
console.log(doubled); // Output: [2, 4, 6, 8, 10]
```

3. Rendering a List in React

Using `map` to Render Lists:

- Use the `map` method to generate JSX elements for each item in an array.
- Wrap list items in `ul` and `li` tags.

Example:

```
function App() {
  const items = ['Apple', 'Banana', 'Cherry', 'Date', 'Elderberry'];

  return (
    <div>
      <h1>Fruit List</h1>
      <ul>
        {items.map((item, index) => (
          <li key={index}>{item}</li>
        ))}
      </ul>
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

4. Warning: Need for `key` Props

Importance of `key` Props:

- Helps React identify which items have changed, been added, or removed.
- Should be unique and stable for each list item.
- Avoid using array indexes as keys for dynamic lists.

Example with Unique Keys:

```
function App() {
  const items = [
    { id: 1, name: 'Apple' },
    { id: 2, name: 'Banana' },
    { id: 3, name: 'Cherry' },
    { id: 4, name: 'Date' },
    { id: 5, name: 'Elderberry' }
  ];

  return (
    <div>
      <h1>Fruit List</h1>
      <ul>
        {items.map((item) => (
          <li key={item.id}>{item.name}</li>
        ))}
      </ul>
    </div>
  );
}

ReactDOM.render(<App />, document.getElementById('root'));
```

5. Complete Code Example

Putting It All Together:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>React List Rendering</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/react/18.2.0/umd/react.production.min.j
s"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/react-dom/18.2.0/umd/react-
dom.production.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
standalone/7.19.3/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    function App() {
      const items = [
        { id: 1, name: 'Apple' },
        { id: 2, name: 'Banana' },
        { id: 3, name: 'Cherry' },
        { id: 4, name: 'Date' },
        { id: 5, name: 'Elderberry' }
      ];

      return (
        <div>
          <h1>Fruit List</h1>
          <ul>
            {items.map((item) => (
              <li key={item.id}>{item.name}</li>
            ))}
          </ul>
        </div>
      );
    }

    ReactDOM.render(<App />, document.getElementById('root'));
  </script>
</body>
</html>
```

Summary

- `map` **method**: Used to transform array elements.
- **Rendering lists in React**: Use `map` to dynamically create list elements.
- **Key prop**: Essential for list items to help React optimize rendering and manage list changes efficiently.

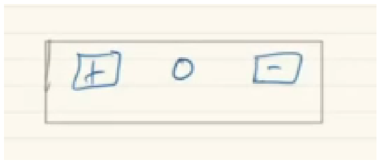
useState

case-1 Counter

Let's understand how we can create a dynamic application:
So the easiest example for that is a simple counter component. Intially we are not using any Create-React-App because you should be clear with the concept first.

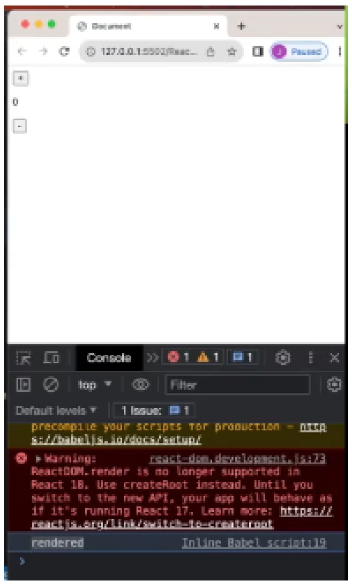
What are the use cases of react(algorithm), react-dom(library actually changes the UI) and Bable (transpiler converts JSX to JS).

So we are building the counter which consist of two two buttons like +, - and the counter in between as shown below.



- So while developing anything in the react first step includes Create static UI.
- Second is always add all the required event listener
- Third is to add the sate.

```
<!DOCTYPE html>
<html lang = "en">

<head>
    <meta charset = "UTF-8">
    <meta http-equiv = "X-UA-Compatible" content = "IE = edge">
    <meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
    <title>Document</title>
    <!-- algorithm to efficiently change your UI -->
    <script src = "https://unpkg.com/react@18/umd/react.development.js"
crossorigin></script>
    <!-- React DOM  -> uses react to update the DOM-->
    <script src = "https://unpkg.com/react-dom@18/umd/react-dom.development.js"
crossorigin></script>
    <!-- transpiler -> JSX -> into javascript -->
    <script src = "https://unpkg.com/babel-standalone@6/babel.min.js"></script>
</head>

<body>

    <div id = "root"></div>
    <script type = "text/babel">
        function Counter(){
        //useState ->initial value
        const [count,setCount] = React.useState(0);
        //const count=arr[0];
        //call this fn given by useState->
        //const updateCount=arr[1];
        //const count=10;

        const increment = () => {
            //tell react -> update count by 1
            console.log("Incremented");
            //setCount(1000);
            setCount(count + 1);
        }
        const decrement = () => {
            //tell react -> update count by -1
            console.log("Decremented");
            setCount(count - 1);
        }
        console.log("rendered");
        return <div>
        <button onClick = {increment}> + </button>
        <p>{count}</p>
        <button onClick = {decrement}> - </button>
        </div>
        }
        ReactDOM.render(<Counter></Counter>, document.getElementById("root"));
    </script>

</body>

</html>
```



```
precompile your scripts for production - https://babeljs.io/docs/setup/
▸ Warning:                          react-dom.development.js:73
ReactDOM.render is no longer supported in
React 18. Use createRoot instead. Until you
switch to the new API, your app will behave as
if it's running React 17. Learn more: https://
reactjs.org/link/switch-to-createroot
rendered                            Inline Babel script:19
```

- The code defines a React component called "Counter," responsible for managing and displaying a count.
- React's `useState` hook is used to manage the state of the "count" variable. It initializes "count" to 0 and provides a function "setCount" for updating it.
- The component includes two functions, "increment" and "decrement," which are triggered when the respective buttons are clicked. They are responsible for updating the "count" state.
- The `render` method of the "Counter" component returns JSX elements, including buttons for incrementing and decrementing the count, and a paragraph displaying the current count value.

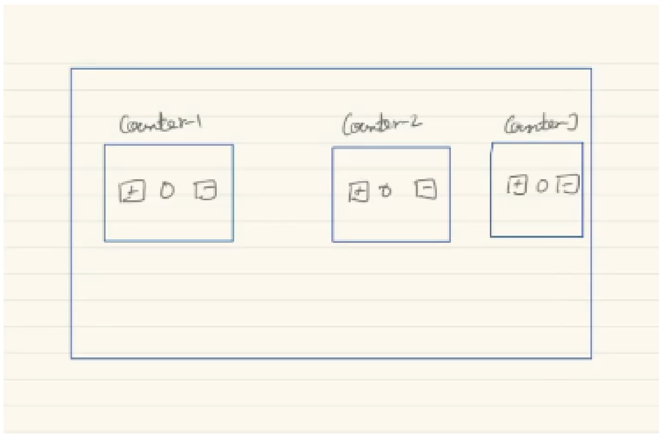
- The last line of code uses `ReactDOM.render` to render the "Counter" component inside the HTML element with the ID "root."
- When the setCount gets called first it updates the count state variable and then call the component again called as rerendering.
- `arr[0]` is state variable and the `arr[1]` is the function to update state variable.

---

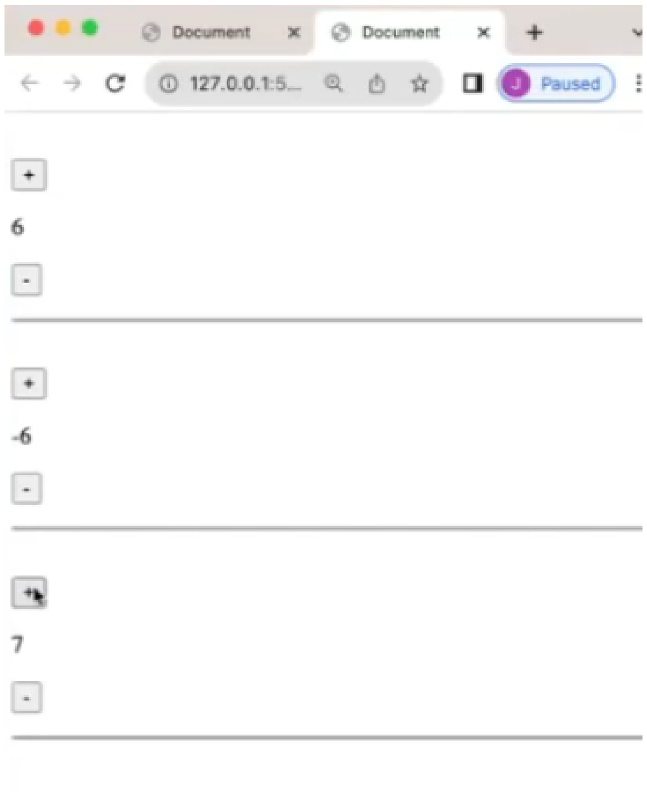
useState case-2 Multiple Counters

Now what should be done to implement the increment and decrement function?
Simply use the `setCount(count+1)` to increment and `setCount(count-1)` to decrement.

What changes will we make in the above code implement the multiple counter as shown below:



```
<!DOCTYPE html>
<html lang = "en">

<head>
    <meta charset = "UTF-8">
    <meta http-equiv = "X-UA-Compatible" content = "IE = edge">
    <meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
    <title>Document</title>
    <!-- algorithm to efficiently change your UI -->
    <script src = "https://unpkg.com/react@18/umd/react.development.js"
crossorigin></script>
    <!-- React DOM  -> uses react to update the DOM-->
    <script src = "https://unpkg.com/react-dom@18/umd/react-dom.development.js"
crossorigin></script>
    <!-- transpiler -> JSX -> into javascript -->
    <script src = "https://unpkg.com/babel-standalone@6/babel.min.js"></script>
    <style>
        .counter{
            margin-top:2rem;
            border-bottom:2px solid gray;
            padding-bottom:1rem;
        }
    </style>
</head>

<body>

    <div id = "root"></div>
    <script type = "text/babel">
        function Counter(){
        const [count,setCount] = React.useState(0);
        const increment = () => {
            setCount(count + 1);
        }
        const decrement = () => {
            setCount(count + 1);
        }
        console.log("rendered");
        return <div>
        <button onClick = {increment}> + </button>
        <p>{count}</p>
        <button onClick = {decrement}> - </button>
        </div>
        }

        fucntion ParentCounter(){
            return <div>
                <Counter/>
                <Counter/>
                <Counter/>
        </div>
        }
        ReactDOM.render(<ParentCounter></ParentCounter>,
document.getElementById("root"));
    </script>

</body>

</html>
```

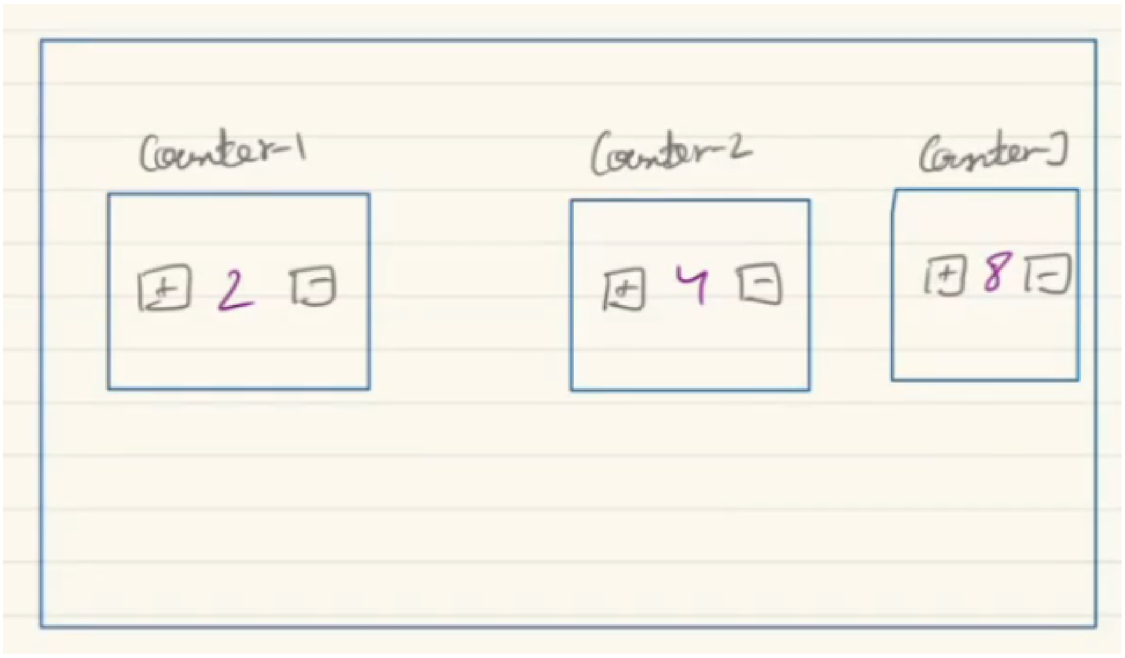


- The "increment" and "decrement" functions in the "Counter" component update the count state when the respective buttons are clicked.
- The "ParentCounter" component is defined to render multiple instances of the "Counter" component, creating a parent-child relationship.
- `ReactDOM.render` is used to render the "ParentCounter" component inside the HTML element with the ID "root." This sets up the initial rendering of the application.

The code creates a React application that consists of a "Counter" component for counting and a "ParentCounter" component that renders multiple "Counter" components. Users can interact with each "Counter" independently to increment and decrement the count value. The application is rendered inside an HTML element with the ID "root."

useState case-3 Counter with Custom Start

How we can send the send the custom start values for the counters?



We can simply pass the Props,

```
<!DOCTYPE html>
<html lang = "en">

<head>
    <meta charset = "UTF-8">
    <meta http-equiv = "X-UA-Compatible" content = "IE = edge">
    <meta name = "viewport" content = "width = device-width, initial-scale = 1.0">
    <title>Document</title>
    <!-- algorithm to efficiently change your UI -->
    <script src = "https://unpkg.com/react@18/umd/react.development.js"
crossorigin></script>
    <!-- React DOM  -> uses react to update the DOM-->
    <script src = "https://unpkg.com/react-dom@18/umd/react-dom.development.js"
crossorigin></script>
    <!-- transpiler -> JSX -> into javascript -->
    <script src = "https://unpkg.com/babel-standalone@6/babel.min.js"></script>
    <style>
        .counter{
            margin-top:2rem;
            border-bottom:2px solid gray;
            padding-bottom:1rem;
        }
    </style>
</head>

<body>
    <div id = "root"></div>
    <script type = "text/babel">
```

```
        function Counter(props){
        const {index,value} = props;
        const [count,setCount] = React.useState(value);
        const increment = () => {
            setCount(count + 1);
        }
        const decrement = () => {
            setCount(count + 1);
        }
        console.log("rendered");
        return <div>
        <h2>Counter Number:{index} </h2>
        <button onClick = {increment}> + </button>
        <p>{count}</p>
        <button onClick = {decrement}> - </button>
        </div>
        }

        fucntion ParrentCounter(){
            return <div>
                <Counter index = {1} value = {2}/>
                <Counter index = {2} value = {3}/>
                <Counter index = {3} value = {4}/>
        </div>
        }
        ReactDOM.render(<ParrentCounter></ParrentCounter>,
document.getElementById("root"));
    </script>

</body>

</html>
```

- You can create the complex component by using the multiple independent component.
- usecase of state wrt to a component: to make it dynamic
- Props: to pass the data to the children component. Majorly the data specific to that component.

NOte : follow this document to install the vite