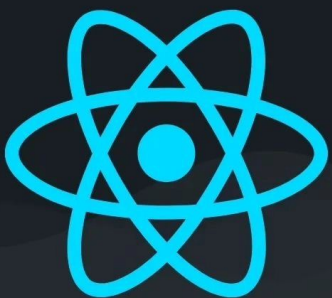
Data Flow In React

The way data is passed



Mallikarjun | @CodeBustler

The way data is passed

- **Parent to Child : Using Props**
  Data is passed from parent components to child components **through props**.

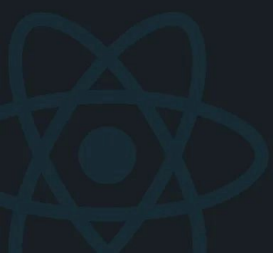
- **Child to Parent : Using Callbacks (Lifting State Up)**
  Child components communicate with their parent components by passing callback functions as props, **lifting state up** in the component hierarchy.

@CodeBustler

The way data is passed

- **Child to Child : Using Context API**
  Context API enables indirect communication between components, creating a shared context.

  It's useful for avoiding prop drilling and allowing components at different levels to access shared data.



Other Ways

- **Using Context API :**

  The Context API is **not limited to child-to-child** communication.

  It can also be employed for **broader state management,** providing a shared context accessible by multiple components.

- **State Management Libraries (e.g., Redux, MobX)**

  State management libraries centralize and manage application state, enabling communication between components regardless of their position in the component tree.

Parent to Child | Props

Parent-To-Child-(Props).jsx

```jsx
// ParentComponent.jsx
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  // Data passed from parent to child
  const dataFromParent = "Hello from parent!";
  return <ChildComponent data={dataFromParent} />;
};


// ChildComponent.jsx
import React from 'react';

const ChildComponent = (props) => {
  // Display data received from parent
  return <p>{props.data}</p>;
};
```

Child To Parent | CallBacks

Child-To-Parent-(Callbacks).jsx

```jsx
// ParentComponent.jsx
import React, { useState } from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  // State to hold data from ChildComponent
  const [childData, setChildData] = useState(null);

  // Callback to update parent state with data from ChildComponent
  const handleChildData = (dataFromChild) => {
    setChildData(dataFromChild);
  };

  return (
    <div>
      <p>Data from Child: {childData}</p>
      {/* ChildComponent with callback to send data to parent */}
      <ChildComponent onSendData={handleChildData} />
    </div>
  );
};
```

Child-To-Parent-(Callbacks).jsx

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

const ChildComponent = ({ onSendData }) => {
  // State to hold child input
  const [childInput, setChildInput] = useState('');

  // Callback to send data to the parent
  const sendDataToParent = () => {
    onSendData(childInput);
  };

  return (
    <div>
      {/* Input for child to enter data */}
      <input
        type="text"
        value={childInput}
        onChange={(e) => setChildInput(e.target.value)}
      />
      {/* Button to send data to the parent */}
      <button onClick={sendDataToParent}>Send Data to Parent</button>
    </div>
  );
};
```

Child To Child | Context

Child-To-Child-(ContextAPI).jsx

```jsx
// DataContext.js
import React, { createContext, useContext, useState } from "react";

// Create a context for shared data
const DataContext = createContext();

// DataProvider component to manage shared state
export const DataProvider = ({ children }) => {
  // State to hold shared data
  const [sharedData, setSharedData] = useState(null);

  // Function to update shared data
  const updateSharedData = (data) => {
    setSharedData(data);
  };

  // Provide shared data and update function to the context
  return (
    <DataContext.Provider value={{ sharedData, updateSharedData }}>
      {children}
    </DataContext.Provider>
  );
};

// Custom hook to access the shared data and update function
export const useData = () => {
  const context = useContext(DataContext);
  if (!context) {
    throw new Error("useData must be used within a DataProvider");
  }
  return context;
};
```

Child-To-Child-(ContextAPI).jsx

```jsx
// ChildComponent1.jsx
import React from 'react';
import { useData } from './DataContext';

// Child component that updates shared data
const ChildComponent1 = () => {
  // Access shared data update function from the context
  const { updateSharedData } = useData();

  // Function to send data to ChildComponent2
  const sendData = () => {
    const data = "Data from ChildComponent1";
    updateSharedData(data);
  };

  return (
    <div>
      <button onClick={sendData}>Send Data to ChildComponent2</button>
    </div>
  );
};
```


Child-To-Child-(ContextAPI).jsx

```jsx
// ChildComponent2.jsx
import React from 'react';
import { useData } from './DataContext';

// Child component that displays shared data
const ChildComponent2 = () => {

  // Access shared data from the context
  const { sharedData } = useData();

  return (
    <div>
      <p>Data received in ChildComponent2: {sharedData}</p>
    </div>
  );
};
```

Child-To-Child-(ContextAPI).jsx

```jsx
// App.jsx
import React from 'react';
import { DataProvider } from './DataContext';
import ChildComponent1 from './ChildComponent1';
import ChildComponent2 from './ChildComponent2';

/* Application component with DataProvider
wrapping child components */

const App = () => {
  return (
    <DataProvider>
      <div>
        <ChildComponent1 />
        <ChildComponent2 />
      </div>
    </DataProvider>
  );
};

export default App;
```


Simple Example Preview

Passing Data from Child-1 to Parent Using (CallBacks Props) & from Parent to Child-2 Using Only Props