Assignment 11 - Data is the new oil

Owner  Pankaj Kumar

1: What is prop drilling?

- Prop drilling is a concept in React where data is passed down from a parent component to nested child components through props. It refers to the process of passing data through multiple layers of components in order to reach a deeply nested child component that needs access to that data.
- In a React application, components are organized in a hierarchical tree structure. Data typically flows from parent components to their child components through props. When a piece of data is required by a component deep down the component tree, it needs to be passed through all the intermediate parent components that do not directly use the data. This process of passing data down through several levels of components is known as prop drilling.

```
```
import React from 'react';
import ChildComponentA from './ChildComponentA';
const ParentComponent = () => {
  const data = "Hello, Prop Drilling!";
  return (

      <ChildComponentA data={data} />

  );
};
```

```
// ChildComponentA.js
```
```

```
import React from 'react';
import ChildComponentB from './ChildComponentB';
const ChildComponentA = ({ data }) => {
  return (
    <div>
      <ChildComponentB data={data} />
    </div>
  );
};
export default ChildComponentA;
```

```
// ChildComponentB.js
import React from 'react';
import ChildComponentC from './ChildComponentC';
const ChildComponentB = ({ data }) => {
  return (
    <div>
      <ChildComponentC data={data} />
    </div>
  );
};
export default ChildComponentB;
```

```
// ChildComponentC.js
import React from 'react';
const ChildComponentC = ({ data }) => {
  return <div>{data}</div>;
};
export default ChildComponentC;
```

In this example, the data prop is passed from the ParentComponent down to ChildComponentA, then from ChildComponentA down to ChildComponentB, and finally from ChildComponentB to ChildComponentC. This pattern continues if we need to pass the data further down to more child components.

2: What is lifting the state up?

- Lifting the state up is a concept in React where the state data is moved from a lower-level component to a higher-level component in the component tree. This is done to share and manage the state data at a common ancestor, making it accessible to multiple child components that need to interact with the same data.

- In React, each component manages its own state data. However, when two or more child components need to share the same state or need to synchronize their data, it becomes beneficial to lift the state up to a common parent component. By doing so, the parent component becomes the "single source of truth" for the shared state, and any changes to the state will propagate down to all child components that use the state.
- Here's an example to illustrate lifting the state up:

```
```
// ParentComponent.js
import React, { useState } from 'react';
import ChildComponentA from './ChildComponentA';
import ChildComponentB from './ChildComponentB';
const ParentComponent = () => {
  const [count, setCount] = useState(0);
  const incrementCount = () => {
    setCount(count + 1);
  };
  return (

      <ChildComponentA count={count} incrementCount={incrementCount} />
      <ChildComponentB count={count} />

  );
};
export default ParentComponent;
```

```
// ChildComponentA.js
import React from 'react';
const ChildComponentA = ({ count, incrementCount }) => {
  return (

      Count: {count}
      <button onClick={incrementCount}>Increment</button>

  );
};
export default ChildComponentA;
```

```
// ChildComponentB.js
import React from 'react';
const ChildComponentB = ({ count }) => {
  return (

```
```

```
      <p>Count: {count}</p>
    </div>
  );
};
export default ChildComponentB;
```

- In this example, the state variable count and the function incrementCount are declared in the ParentComponent, which serves as the common parent for ChildComponentA and ChildComponentB. The count state is lifted up to ParentComponent and passed down as a prop to both ChildComponentA and ChildComponentB. When ChildComponentA increments the count using the incrementCount function, the updated count will be reflected in both ChildComponentA and ChildComponentB, as they share the same state.
- Lifting the state up promotes better data flow, simplifies state management, and helps to avoid inconsistencies in the application state. It is particularly useful when components need to interact with each other or when multiple components depend on the same data source.

3: What is Context Provider and Context Consumer?

1. `Context Provider` is used to provide access to a context between multiple components of the application. We can provide the access to the context or the data layer to the whole application to the and its subcomponents.

   For example:

   ```
   return (
     <UserContext.Provider value={{ user: user, setUser: setUser }}>
       <Head />
       <Outlet />
       <Footer />
     </UserContext.Provider>
   );
   ```

   In the code above , we are providing the access of UserContext to <Head /> <Outlet /> and <Footer /> component

2. `Context Consumer` is used to consume the context data , provided by react context. We can do this using `useContext` hook for functional components

and `Context.Consumer` in clasees based components.

For example:

- In Class based components, we can use the Context and use the consumer.

```
<UserContext.Consumer>
  {({ user }) => <h1>{user.name}</h1>}
</UserContext.Consumer>
```

- In Functional components we can use `useContext` hook to consume the context.

```
import UserContext from "../utils/UserContext";
import { useContext } from "react";

const { user } = useContext(UserContext);
```

4: If you don’t pass a value to the provider does it take the default value?

If we do not override the values of context it takes the default values form the context , when we initialise the context.