Date.....

Chapter-11 Data is the new Oil

Data layer + UI Layer

when you have a web application / react application (any single page application) there's a UI layer on it (UI layer → everything you see on the browser screen [things that are visible] (buttons, headings, colors)).
and the whole UI layer is powered by the data layer.
Data layers consists of state and props which maintain the data within the application.

Q. what is the difference b/w state and props ?
Soln: Basic difference is suppose you have a container and you want a variable just within that container/component it is known as state. It is the local variable. If we have to pass value from one component to another component we can use props. So props are kind of the local state for our parent.

Props drilling

Suppose if we have a local state in our AppLayout component and we want to ~~pass~~ use it ~~as props~~ in RestaurantCard ~~so~~ but we cannot pass it directly to the RestaurantCard as an props because we ~~didn't~~ doesn't ~~not~~ have used RestaurantCard directly in AppLayout. So we have to pass it first to body component, then from body component to RestaurantCard.

Prop drilling is a situation where data is passed from one component through multiple interdependent components until you get to the component where the data is needed.

Date.....

but props drilling has some major disadvantages as well

① we have to change more number of files so if we change one prop than all the involved components will re-render

also if we want to send something from children to parent we can do it using custom hooks.

→ we can't change other state while in a state.

eg suppose in our instamart component if we want to make a accordion component :- [lifting the state up] → our parents will take control over childrens

// code in instamart.js

```
import {useState} from "react"
const Section = ({title, description, isVisible, setIsVisible}) => {
  return (
    <div clanName="border">
      <h3 clanName="font-bold">{title}</h3>
      {isVisible ? (
        <button
          onClick={() => setIsVisible(false)}>Hide</button>) :
        (<button
          onClick={() => setIsVisible(true)}></button>)}
      {isVisible && <p>{description}</p>}
    </div>
  );
};

const Instamart = () => {
  const [sectionConfig, setSectionConfig] = useState({
    showAbout: false,
    showTeam: false,
    showCareers: false,
  });
```

Spiral

Teacher's Sign .....

```
<Section
    title = {"Careers"}
    description = {
        "On the other hand, we denounce with righteous"
    }
    isVisible = {sectionConfig.showCareers}
    setIsVisible = {() =>
        setSectionConfig({
            showAbout: false,
            showTeam: false,
            showCareers: true,
        })
    }
/>

<Section
    title = {"About Instamart"}
    description = {
        "On the other hand, we denounce with righteous ---- "
    }
    isVisible = {sectionConfig.showAbout}
    setIsVisible = {() =>
        setSectionConfig({
            showAbout: true,
            showTeam: false,
            showCareers: false,
        })
    }
/>
```

Date.....

```
<Section
    title={"Team Instament"}
    description={
        "On the other hand, we denounce with righ ---"
    }
    isVisible={sectionConfig.showTeam}
    setIsVisible={() =>
        setSectionConfig({
            showAbout: false,
            showTeam: true,
            showCareers: false,
        })
    }
/>
```

but this is a mess (NOT A GOOD WAY OF WRITING CODE !!)

The type of this code is called (shit code) → That is why code written by intern not pushed to production.

So we can optimised the above code

~~HomeWork~~

(will write at last of you copy on last at this chapter)

→ play with profiler of react developer tool
It will give you information about how your code render, how much time it took to render.
Its a very good you can use to build large platform and optimize it.

Date.....

Context React

Whenever you need same data accross all your app you have to store it in a central space that central space can be ~~local storage~~ but local storage is not reliable. It work inside browser but here we are talking about react environment/state and updating a local storage is a heavy/costly opprration. So we need to have this data in our react app at central place, react gives us accss to this central space known as Context. (react context) Some companies also use redux store.

→ we are not use global variables instead of react context because react if not watching it.

→ State and props are tied to components but context is separate, It is not specific to any component, we can have it in a central place.

// create a react context

① // create a file UserContext.js (you can make context anywhere all separate file or same file)
// in ~~conte~~ UserContext.js

```
import { createContext } from "react";
const UserContext = createContext({
    user: {
      name: "Dummy Name",
      email: "dummy@gmail.com",
    },
});
export default UserContext;
```

Date.....

2) // Now to use it (suppose we want to use it in header.js)

```
// in header.js
import {useState, useContext} from "react";
import UserContext from "../src/utils/UserContext.js";
:
:
const Header = () => {
  const {user} = useContext(UserContext);
return (
  <div>
    <h1 — —
    :
    <h3> {user.name} </h3>
  </div>
)
};
```

→ Why do we even need props then, can I keep everything inside my context?

No, because context has seperate things and state/props are different. State/props are tied to the components, context is a central store so we don't use context for everything, we use context for data which is required all across the application at different-different places.

Context is like useState for whole big application.

Date.....

(8) // use react context in class based component

// in our About.js (that is a class based component) if we have to use UserContext, we have to use it as a component, code :-

~~import { outlet } from "react-router-dom"~~
~~import fetchProfileFunction~~

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

class About extends Component {

  constructor(props) {
      super(props);
  };
  componentDidMount() {

  }

  render() {

    return (
      <div>
        <h1>About us page</h1>
        <UserContext.Consumer>
          {({user}) => <h4>{user.name}</h4>}
        </UserContext.Consumer>

      </div>
    );
  }

}
```

, destructuring

Date.....

9) update react Context

// in our root file App.js if we want to update our react context we have something called • provider.

```
const AppLayout = () => {

  const [user, setUser] = useState({
    name: "Akshay Saini",
    email: "support@namastedev.com",
  });

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

→ in component mein context ki data update ho jaega.

It means I can modify my context for the smaller portion of my app.

// now suppose we want to update our react context from body so for that our code will look like.

Date.....

// in app.js

```
const AppLayout = () => {

  const [user, setUser] = useState({
    name: "Akshay Saini",
    email: "support@namastedev.com",
  });

  return (
    <UserContext.Provider value={{ user: user,
                                   setUser: setUser,
                                 }}>

      <Header />
      <Outlet />
      <Footer />
    </UserContext.Provider>

  );
};
```

// in our body.js

```
const {user, setUser} = useContext(UserContext);
  :
<input value={user.name} onChange={(e) => setUser({
                                     name: e.target.value,
                                     email: "namaste@dev"
                                   })
}></input>.
```

Date.....

// Now if I want to change only one at time (ie. name or email)

```
<input value={user.name} onChange={(e) => setUser({
                                        ...user,
                                        name: e.target.value,
                                    })}></input>

<input value={user.name} onChange={(e) => setUser({
                                        ...user,
                                        email: e.target.value,
                                    })}></input>.
```

// → In components of react-develop tool we can see context.provider. wrapping our

```
<UserContext.Provider>
    <Header>
    <Outlet>
    <Footer>
</UserContext.Provider>
```

but in our component of react-develop tool we have written (context.provider) → in debugging

so it could confuse us, if we have other context.provider, but react gives us this power to name my context.provider.

// UserContext.js

```
import {createContext} from "react";
const UserContext = createContext({
    user: {
        name: "Dummy Name",
        email: "dummy@gmail.com",
    },
});
UserContext.displayName = "UserContext";
export default UserContext;
```

Spiral

Teacher's Sign .....