* Context in ReactJS :-

Context in ReactJS provides a way to share data that is required by multiple components in our application without the need to pass props down the component tree manually. It allows us to pass data to components at different levels in the component tree, without having to pass props down through every level.

ReactJS Context API consists of two components, a 'provider' component, and a 'consumer' component. The 'provider' component holds the data that needs to be shared, and the 'consumer' component is used to access the data from the Provider.

- **Why we use it?**

"When we want some piece of data to be accessible anywhere in my app, we use context API."

And other reasons are :—

- Share data across multiple components:
- Reduce Prop drilling:
- Centralize data management:
- Provide a way to access global data:
- Simplify the component tree:

* What is redux?

Redux is a tool for managing the state (data) in a React application. It allows us to store all our application's data in one place (known as the "store"). And This makes it easier to build complex, real-world

applications and keep the state organized.

* **Cons of using Redux :—**
  - It takes time to learn and set up, especially for small or simple applications.
  - Debugging, the state updates can be difficult to follow and trace through the application.

That being said, Redux has been widely adopted and can be very beneficial for large, complex applications.

If our applications is very small, then we don't need redux, Only use when we have to build large scale applications involves a lot of data handling.

- Redux comes with lot of complexities.
- A huge learning curve.

React and Redux are both different things

* **Redux Toolkit :—**

Redux Toolkit is a library that makes it easier to work with Redux in a React application. It provides a set of tools and conventions for working with the state management and reduces the amount of boilerplate code that needs to be written.

Redux have a single store only, where we can store any thing

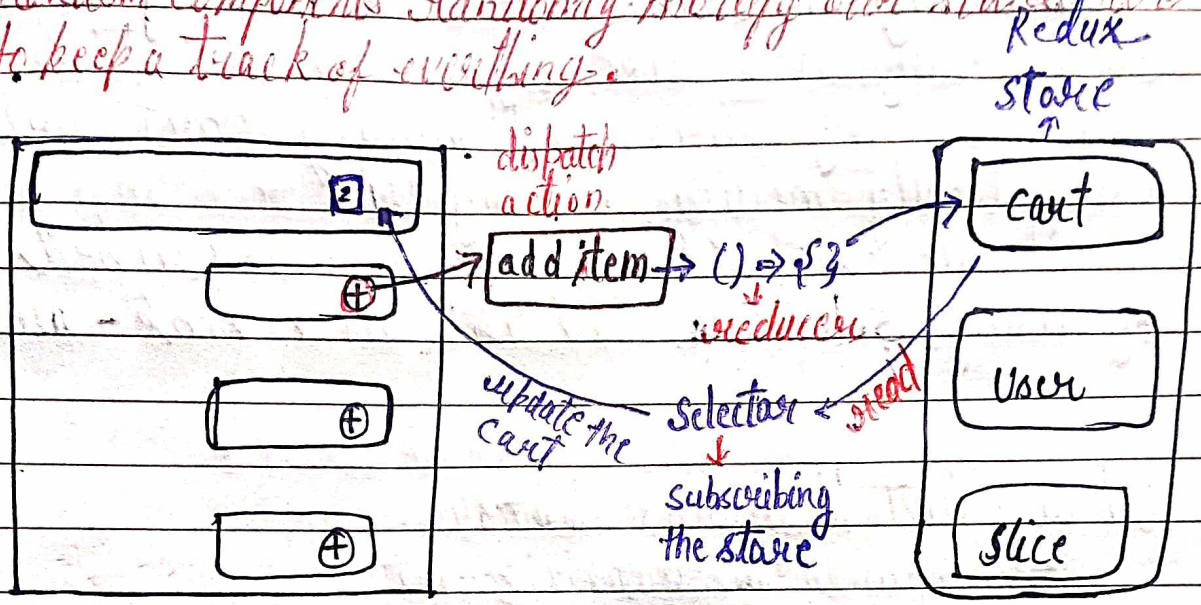
It is poor thing to keep all data at one place, but not it's not a poor thing. Because we will create logical separation into our store. We will create 'slices' of our store.
Each store will have different slices. What all slices can have in our app. It can have a
- user slice
- authentication slice
- theme slice
- cart slice

A 'slice' is a small portion of the store.

Our component cannot directly modify the store. Suppose if we click on the '+' button, we cannot directly modify the store. or cannot modify the 'cart'.
So, what we can do — so, we have to do 'dispatch' and 'action'.
'action' can be supposed as 'add item' (for example) when we will click on 'add item' it means we are dispatching an action.
What will this action does —
It will call an ~~action~~ a function.
↳ normal JavaScript function.

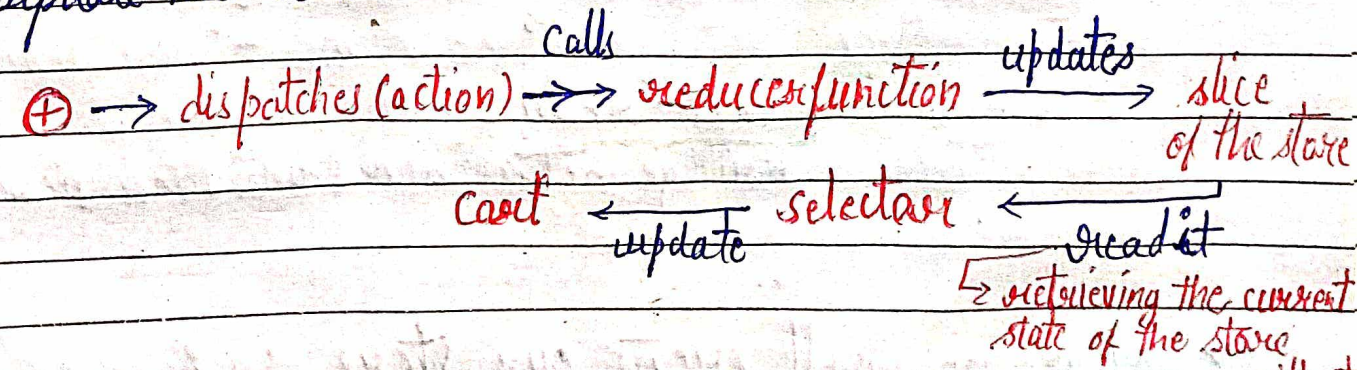
Date: / /

And this function will modify our 'cart'. We cannot directly modify our app. Why ??? So whenever there is large application, we don't want random components randomly modify our states we need to keep a track of everything.



"When we click '+' button, we dispatch an action which calls a function known as reducer function, which modify or update the slice of the store."

There is something known as selector, if we want to read cart, we have to call this selector, the selector will give us the information that will update the cart



'Selector' means, we are selecting the slice out of the store, or we are selecting the portion of the store. Selector is a hook end of the day and hook is a normal javascript function. / useSelector()

There is one more jargon, that is - when we use selector - it is known as subscribing to the store.

subscribing the store $\xrightarrow{\text{means}}$ reading the store

So, basically, it is like async with the store. Whenever my store would modify, cart will automatically modify in my UI

It means the component is subscribed by using **selectors**. and this is a hook - useSelector()

→ Let us install redux library.

→ npm i @reduxjs/toolkit

→ npm i react-redux

??? Why we are installing two libraries.

Core Job of Redux is to maintain the store (data), create the slices.

npm i @reduxjs/toolkit

core of the redux

npm i react-redux

bridge between redux and react.

→ Now, we will create our store, so for creating a store we need configureStore() and this configureStore will come from "@reduxjs/toolkit"

This configureStore will contain slices:

Now, this store is different and our app is also different, so how we will connect our store to our app.

So will we need a provider to connect our store to / with the app.

We can put this provider to our whole app or for certain components also.

So, here we will provide this provider to our whole app. So in our case, the root component is App layout. So we will provide this store to our whole app.

Here, we will import a provider which comes from react-redux and in this way the Provider component will act as a bridge between our store and app for connecting them.

```
<Provider store={store}>
    <UserContext.Provider
      value={{
        user: user,
        setUser: setUser,
      }}
    >
      <Header />
      <Outlet />
      <Footer />
    </UserContext.Provider>
</Provider>
```

Now, lets create a slice for our cart. We create slices by using createSlice() and it comes from '@reduxjs/toolkit'.
because creating slices is the core job of our redux toolkit.

For creating slice we need two things first
- the name of the slice
- and the initial state

```
1.  const cartSlice = createSlice({
2.      name : 'cart',
3.      initialState : {
4.          items : [],
5.      },
    })
```

So, now we have to modify our app. so will we use Reducer() function to modify our app. And remember when this reducer function calls - on dispatch of an action.
Now, we have to gave a name for this dispatch action, lets gives the name - addItem
So, it means, this reducer function will contain a mapping of actions (addItem) and our reducer function.

```
6.      reducers : {
7.          addItem : () => {
8.
9.          }
        }
```

So, How we can modify our store, this reducers function takes in two things

- state → (Initial state)
- action → (This is the place where I've we get the item which will we have to add to our cart)

↳ action payload

```
reducers: {
    addItem: (state, action) => {
        // some logic
    }
}
```

* * * * * * * * * * * * *

```
reducers: {
    addItem: (state, action) => {
        state.items.push(action.payload);
    },
    removeItem: (state, action) => {
        state.items.pop();
    },
    clearCart: (state) => {
        state.items = [];
    },
},
});
```

all actions name

```
export const { addItem, removeItem, clearCart } = cartSlice.actions;
```

↳ from here we exports all actions.

```
export default cartSlice.reducer;
```

↳ from here we exports all reducers

Syntax

So, Now our slice is ready and we have to put it inside our store. When we will configuring the store we will put the slice inside our store.

Now we will import cartSlice for our store by using

```
import cartSlice from "./cartSlice";
```

So whe we import this cartSlice, what this cartSlice was exporting by default - i.e reducer.
So, basically all these reducers, we will put inside our store, like this —

store.js

```
reducer: {
    cart: cartSlice,
},
```

→ Now, its time to subscribe our store. by using selector - useSelector() which comes from 'react-redux'.

```
import { useSelector } from "react-redux";
```

⇒ Now write this —

```
const cartItems = useSelector(store => store.cart.items)
```

It means our cart items will come from useSelector(). This useSelector gives access to the store. So, we want to subscribe the store cart items.

Now, lets create some actions — addItem

```
const handleAddItem = () => {
    dispatch(addItem("Grapes"));
}
```

dispatch an action
action = addItem

↳ This will come from slice that we have export as named export in cartSlice.js

From above, we have created a button which has handleAddItem()
Now in this handleAddItem() button, as soon as we click on the button, We will dispatch an action (i.e addItem) and we should put in some value (i.e "Grapes").
And this 'dispatch()' will come from another hook known as useDispatch() and my dear friends this 'useDispatch()' will come from 'react-redux'.

*Note :—

→ Always subscribe to the specific portion of the app.

good practice

Ex:- const cartItems = useSelector((store) => store.cart.items);

Now, my cart component is only subscribing to my 'items' array.

→ Important tool → { Redux DevTools }

→ Let's recall :—

When we click on 'addItem' button, then it dispatches an action which calls the 'reducer()' function which is updating the slice of the redux store. And our cart is then subscribed to our store using a 'useSelector()' hook and it is atomatically magically (getting) updated.

* Why we can't directly modify our store in Redux Toolkit.

Because the state is manged by the framework

Redux toolkit helps manage the state (data) of our react app in predictable and organized way. To make sure that everthing stays organized and consistent, we can't just directly change the state whenever we want. We have to use specific actions and functions to change the state. This way, Redux Toolkit, can keep track all of the changes and make sure that everything is updated correctly and consistently.