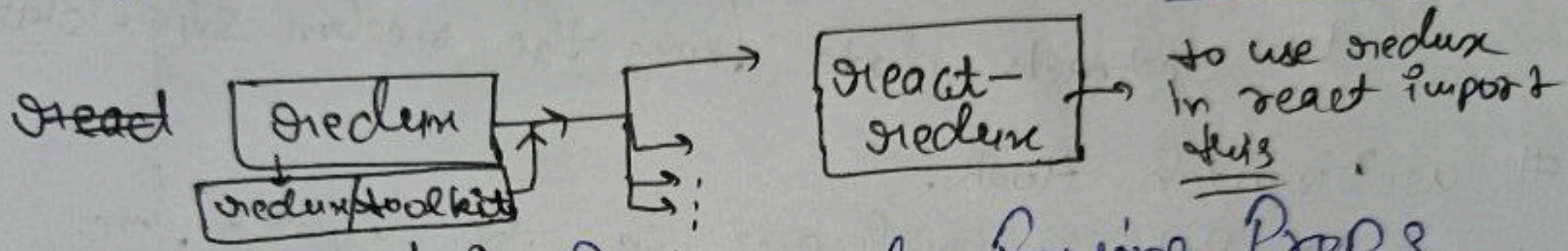
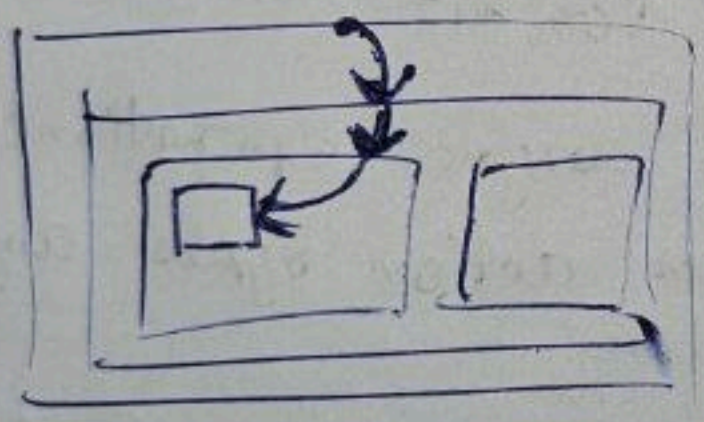
28-Oct | Redux | REACT CONTINUE -15 | ①

→ Redux is an independent State Management library
→ It helps manage application state in a centralized store, making state changes predictable & easier to debug.
→ It is similar to Context-Api the inbuilt State Management tool for react internally → useContent.



→ Back of the Mind ⁰ Problem of Passing Props



Intermediate Components are getting access to useless props to avoid this we use central Stores & that is the core functionality react-redux or redux provides

Initially flux was used (Developed by FB) But Now Redux

Three Principles of Redux

1) Single Source of Truth: The entire state is stored in a single object tree within one store.
2) State is Read-Only: The only way to change State is to dispatch an action.
3) Changes are Made with Pure Functions: Reducers must be pure functions that returns new state objects.

Redux Toolkit . RTK.

↳ RTK is the official, recommended way to write Redux logic

→ CREATING A SLICE.

→ CONFIGURING THE STORE.

TODOS.

Using REDUX WITH REACT.

PROVIDER COMPONENT

↳ wraps your app to make the Store available.

~~use~~

useSelector Hook :-

↳ extracts data from the redux storeState.

useDispatch Hook.

↳ Return a reference to the dispatch func.

Async. Logic with Redux Toolkit.

createAsyncThunk => Handles async. operations and Generates action types auto.

BEST PRACTICES :-

→ use Redux Toolkit Instead of legacy Redux.

→ Keep State Normalized (avoid nested data).

→ Use Selector functions to access state.

→ Keep reducers pure & side effect free.

→ Don't put non-Serializable values in state (functions promises etc).

→ Handle Async. Logic with createAsyncThunk.

Project: Redux Toolkit - ToDo.

i) Installation :- It is Redux Toolkit.

1) configureStore() : wraps createStore to provide simplified configuration options and good defaults. It can auto combine your slice reducers, adds whatever Redux middleware you supply, includes redux-thunk by default and enables use of the Redux DevTool Extention.

2) createReducer() : that lets you supply a lookup table of action type to case reducer functions, rather than writing switch statements. In addition it auto uses immer lib. to let you write simple immutable updates with normal mutative code like { state.todos[3].completed = true }

↳ we use slice in reducers.

3) Slice is a bigger version of reducer.

4) reducer is a functionality by ~~redux toolkit~~ Redux.

↳ Action : Plain JS object that describes what happened. They must have a type property.

action object.

```
const addTodo = {
    type: 'ADD_TODO',
    payload: {
        id: 1,
        text: 'Learn Redux'
    }
}
```

6) Action Creators : func. that creates & returns action objects

```
const addTodo = (text) => ( { type: 'ADD_TODO',
    payload : {id : Date.now(), text } });
```

4) Reducers :-

Pure function → that takes current State and an action, then return new State.

This is core Redux concept.

Traditional reducer :→

action - object.

```
Const todosReducer = ( state=[], action ) => {
    Switch (action.type) {
        Case 'ADD_TODO':
            return [... state, action.payload];

        case 'REMOVE_TODO':
            return state.filter( (todo) => (
                todo.id !== actior.payload.id);

        default
            State;
}
};
```

7) Dispatch :

This method is used to send actions to the Store.

```
store.dispatch ( addTodo ('learn Redux'));
```

But these were used Earlier

In Redux Toolkit → we Modern Approach.

↳ we create slice and Store

How? Page ②.

Step 1° first install the core library of redux.

npm install @reduxjs/toolkit

or

goto Redux-toolkit.js.org

Step 2: Install this for react binding with redux-toolkit.

npm install react-redux.

what is included °-

(i) configureStore()
(ii) createReducer()
(iii) createAction()
(iv) createSlice().
(v) combinesSlice()
(vi) createAsyncThunk
(vii) createEntityAdapter
(viii) createSelector.
↓
it is utility from the Reselect library, re-exported for ease of use.

Step 1) # under src create folder → app

Process

↳ ./app/store.js
↳ your central store.

store.js

```
import { configureStore } from '@reduxjs/toolkit'
import todoReducer from '../features/todo/todoSlice';
export const store = configureStore({
    reducer : todoReducer
})
```

4)

Step

Step 2) Create Reducers → we call them Slices (reducers ही)

./src/features/todo/todoSlice.js

```
1  import {createSlice, nanoid} from '@-/-';
                                  ↓
                  generates random id's (for users we need it)

2  const initialState = {
       todos : [{id: 1, text : "Hello World"}]
   }

3  // use createSlice( ) method to create a Slice : takes current state & action, return new state.
   //      ↳ it takes object as parameter.

4  export const todoSlice = createSlice({
       name : 'todo',          // name is a property in RTK. (fixed)
       initialState,           // there is always an initialState for a Slice.
       reducers : {            // property : function
           addTodo: (state, action) => ([...state, action.payload])
       }
   })
```

(OR)

isko under append hota jisse name (data)

fixed hai Keywords.

gives you access to current initial State yes the current State of your initialised obj.

inputs passed goes under actions.

reducers: {

addTodo: (state, action) => {

const todo = {

id: nanoid(),

text: ~~todo~~ action.payload.text
(data)

};

state.todos.push(todo)

},

removeTodo: () => { --- },

}

4) → you ToDo Slice is Ready.

Recap: Slice are functions in redux that wraps your name, initialState and reducers

5th line in todoSlice.js { we need to export our reducers so that they can be ~~used~~ imported in components. }

5 export const { addTodo, removeTodo } = todoSlice.actions

6. export default todoSlice.reducer

↳ to pass these reducers to your central Store.

./src/components :-

① addTodo → it is [ ] Add

↳ to insert a record you need to send some command to your Store.

→ dispatch( ) is used to send action.

AddTodo functionality :-

→ dispatch किसी reducer को use करते हुए store को access करता है।

```
const [input, setInput] = useState('')
const dispatch = useDispatch()
const addTodoHandler = (e) => {
    e.preventDefault()
    dispatch(addTodo(input))
    setInput('') ; }
<form onSubmit = {addTodoHandler}>
```

or use {text: input, completed: false} (used when multiple arguments.)

addTodo is a reducer that expects action.payload.text

---

* Redux is very automative & smart

You just sent your text if it is a string it will auto interpret it as the required input of your destinated function for eg you func. wants a action.payload.text to pass it you have 2 option

↓

```
const actionObject = {
    type : 'todoText',
    payload : {
        text : "abc..."
    }
}
```

this is what your func. want

↓

So you can pass it directly → directly pass a String . it auto consider it as action.payload.text

---

useSelector() method

→ in this method you get access of state state holds your initialState also.

when you need to list all the todos
You will need to access State.

```
Const allTodos = useSelector((State)=> (state.todos));
```

return the list of todos

```
const dispatch = useDispatch();
```

this is our dispatch function.

How to list all using jsx :-

using .map() you get each todo.

```
<>
  <div> Todos </div>
  {todos.map((todo) => (
      <li key={todo.id}>
          {todo.text}
          <button
            onClick={() => dispatch(removeTodo(todo.id))}
          > X </button>
      </li>
  ))}
</>
```

→ ./Components/todos.jsx.

Now you have your Components Ready which you will render in your app.jsx.

```
<addTodo />
<Todos />
```

→ this way

Your Components must be wrapped in Provider

Capital

```
import { Provider } from 'react-redux';
import {store} from './app/store';
```

You need to wrap you components wherever you want → app.jsx —
main.jsx —

main.jsx

```
createRoot(document.getElementById('root')).
  render(
    <Provider store={store}>
      <App/>
    </Provider>
)
```

Project 2 On Redux State Management