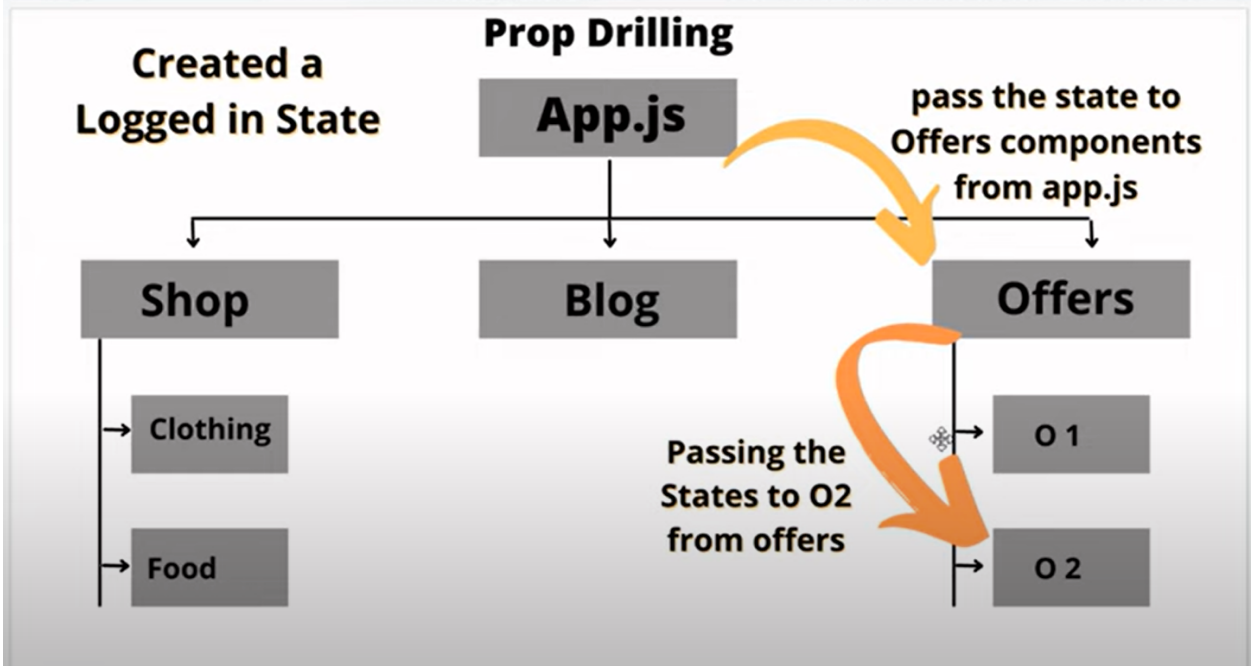
Redux

- Redux is a library for JavaScript Applications.
- You can use Redux together with React or with any other view library (Angular,Vue).
- Redux is a state container.
- For example it keeps Predictable track of state of variables in a Registration form.

```
State={
  Name: " ",
  Email: " ",
    Password: " ",
}
```

- Redux solves the problem of prop Drilling similar to Context Api.



- Difference between Context API and Redux

→**Context API:** Provided by React, it allows you to manage global state and share data between components without prop drilling. It's simpler to use for smaller applications or when you don't need advanced features like middleware.

→**Redux:** Also for managing global state, but it's more powerful and comes with tools for advanced state management, like middleware and time-travel debugging. It's great for larger applications with complex state interactions or when you need predictable state management.

#Core Concepts Of React-Redux

- Store :- Holds state of your application.
- Action :- Describe the changes in the state of application
- Reducer :- Actually carries out the state transition depending on the action.
- Example :- Book Shop

```
Shop (store) -------------- ShopKeeper (Reducer) ---------------
```

#Rules of Redux

- The state of your application is stored in an object tree within a single store.

```
{
  NumberOfBooks: 10
}
```

- only way to change the state is to emit an action, an object describing what happened
- To specify how the state tree is transformed by actions , we write pure reducer.

#Redux Installation

```
npm install redux react-redux
```

#Action in Redux

- Actions are JavaScript object that contains Information.
- Actions are the only source of information for the store. It only tells us what has happened.
- Actions have a type property and it should be defined in string constraint.
- It is compulsory to include the type property in the object.

```
//BookAction.js
import {buyBook} from './BookTypes'
const purchase_book =()=>{
 return {
   type: buyBook
 }
}
```

```
//BookTypes.js

export const buy_book = 'buy_book';
```

#Reducers in React

- Reducers decides how the state of application changes depending upon the action sent to the store.
- Reducers are the function that accepts state and action as parameter and returns the next state of the application. (previousState , action) ⇒ newState

```
//BookReducer.js
import {buy_book} from './BookTypes'
const initialState = {
  NumberOfBooks: 20
}

const BookReducer=(state = initialState, action)=>{
    switch(action.type){
       case buy_book: return{
          ...state, NumberOfBooks  : state.NumberOfBooks -1
       }
       default : return state
    }
}
export default BookReducer;
```

#Redux Store

- Entire Application contains Single Store.
- It is responsible for holding application state.
- getState() method gives access to state it hold.
- dispatch(action) method allow state to be updated.
- It has subscribe(listener) method as well by which we can register listeners.

This method accept function (listener) as a parameter which execute anytime when the state in redux store changes.

```
//Store.js
import {createStore} from 'redux';
import BookReducer from './BookReducer';

const store = createStore(BookReducer);
export default store;
```

```
function App(){
  return (
    <Provider store={store} >
    <Provider>
  )
}
```

#React Redux + Hooks

- React Redux offers set of hooks to - subscribe to redux store and dispatch actions.

1. useSelector() Hook-
    - useSelector is a hook react-redux library provides to get hold of any state that is maintained in the redux store.

      Syntax:- const xyz = useSelector(selector: Function , equalityFn?: Function)

    Selector functn accepts the redux state as its argument and return a value.

    ```
    //BookContainer
    import React from 'react'
    ```

```
import {useSelector} from 'react-redux'

function BookContainer(){
   const noOfBooks = useSelector(state => state.NumberOfB
 return (
    <>
    <div>BookContainer</div>
    <h2>No of Books - {noOfBooks} </h2>
    </>
  )
}

export default BookContainer;
```

2. useDispatch()
   - This hook returns a reference to the dispatch function from the Redux store. You may use it to dispatch actions as needed.

   Syntax:- `const dispatch = useDispatch()`

```
//BookContainer
import React from 'react'
import {useSelector} from 'react-redux'
import purchase_book from './BookAction'

function BookContainer(){
   const noOfBooks = useSelector(state => state.NumberOfB
   const dispatch = useDispatch()
 return (
    <>
    <div>BookContainer</div>
    <h2>No of Books - {noOfBooks} </h2>
    <button onClick={()=>{dispatch(purchase_book())}}>Buy
    </>fbook
  )
}
```

```
export default BookContainer;
```

#Redux Toolkit (RTK)

React-Redux: React-Redux is the official binding library that integrates Redux with React applications. It provides components like `Provider` and `connect` to connect your React components to the Redux store. It simplifies the process of subscribing to the Redux store updates and dispatching actions from React components.

Redux Toolkit: Redux Toolkit is the official opinionated toolset for Redux development. It includes utilities to simplify several aspects of Redux, such as store setup, reducer creation, immutability helpers, and middleware setup. It encourages best practices and reduces boilerplate code.

Use Case:

- **React-Redux:** You use React-Redux when you have a React application and want to efficiently connect your components to a Redux store. It's essential for managing state across a large or complex React application, ensuring that state updates are handled efficiently and components are re-rendered only when necessary.
- **Redux Toolkit:** Redux Toolkit is used when you want to streamline your Redux development process. It's especially useful when setting up a new Redux project or refactoring an existing one to adopt Redux best practices. Redux Toolkit's `configureStore` function simplifies store configuration, `createSlice` reduces boilerplate when defining reducers, and `createAsyncThunk` helps manage asynchronous logic like API requests seamlessly.

```
//todoSlice.js
//createSlice in Redux Toolkit (RTK) is used to make it ea
import {createSlice, nanoid } from '@reduxjs/toolkit';

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



export const todoSlice = createSlice({
    name: 'todo',
    initialState,
    reducers: {
        addTodo: (state, action) => {
            const todo = {
                id: nanoid(),
                text: action.payload
            }
            state.todos.push(todo)
        },
        removeTodo: (state, action) => {
            state.todos = state.todos.filter((todo) => to
        },
    }
})

export const {addTodo, removeTodo} = todoSlice.actions

export default todoSlice.reducer
```

```
//AddTodo.jsx
import React, {useState} from 'react'
import {useDispatch} from 'react-redux'
```

```jsx
import {addTodo} from '../features/todo/todoSlice'

function AddTodo() {

    const [input, setInput] = useState('')
    const dispatch = useDispatch()

    const addTodoHandler = (e) => {
        e.preventDefault()
        dispatch(addTodo(input))
        setInput('')
    }

  return (
    <form onSubmit={addTodoHandler} className="space-x-3 m
      <input
        type="text"
        className="bg-gray-800 rounded border border-gray-
        placeholder="Enter a Todo..."
        value={input}
        onChange={(e) => setInput(e.target.value)}
      />
      <button
        type="submit"
        className="text-white bg-indigo-500 border-0 py-2
      >
        Add Todo
      </button>
    </form>
  )
}

export default AddTodo
```

```jsx
//todos.jsx
import React from 'react'
import { useSelector, useDispatch } from 'react-redux'
import {removeTodo} from '../features/todo/todoSlice'

function Todos() {
    const todos = useSelector(state => state.todos)
    const dispatch = useDispatch()

  return (
    <>
    <div>Todos</div>
    <ul className="list-none">
        {todos.map((todo) => (
          <li
            className="mt-4 flex justify-between items-cer
            key={todo.id}
          >
            <div className='text-white'>{todo.text}</div>
            <button
             onClick={() => dispatch(removeTodo(todo.id))
              className="text-white bg-red-500 border-0 py
            >
              <svg
                xmlns="http://www.w3.org/2000/svg"
                fill="none"
                viewBox="0 0 24 24"
                strokeWidth={1.5}
                stroke="currentColor"
                className="w-6 h-6"
              >
                <path
                  strokeLinecap="round"
                  strokeLinejoin="round"
                  d="M14.74 9l-.346 9m-4.788 0L9.26 9m9.9(
```

```
                    />
                  </svg>
                </button>
              </li>
            ))}
          </ul>
        </>
      )
}

export default Todos
```

```
//store.js
import {configureStore} from '@reduxjs/toolkit';
import todoReducer from '../features/todo/todoSlice';

export const store = configureStore({
    reducer: todoReducer
})
```