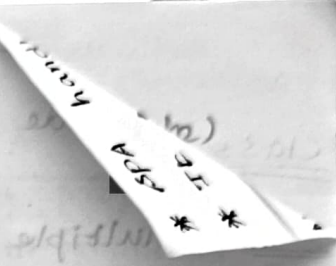
```
Let calc = new calculation()
calc.solve()
sum()
log(result)  ⟶ 30
mul()
log(result)  ⟶ 200
```



React -JS → 2013 - open browser. but Since 2011.

* JS Library React
* Declarative
* created by facebook.
* JS framework angular
* component based
* was created in 2011 by Jordan walke.

Based on components:

header.js, about.js → create individual page for each section

Declarative:

JS, HTML, CSS → JSX → JS XML notation.

with declarative you just tell what to do and in imperative you also tell how to do (too much of instructions or in other words (coding steps))

SPA:

* SPA - Single page Application.
* Application operates in Single web page.
* instead of loading separate html pages for different interaction, SPA dynamically update the content on the same page.
* It initially loaded the entire content and update parts of the page dynamically.

* SPA handle Client - side routing for navigation
* It provide smoother and interactive experience

MPA - Multiple page Application.

* Each page has own set of html, css and js
* full page reload occurs when navigating between page.
* Server-side rendering used to generate dynamic content.

why do FrontEnd framework / Library exists:

* Handling data + displaying data in a UI
* UI needs stay sync with data
* Very hard problem to solve.

Library

Less no.of rules

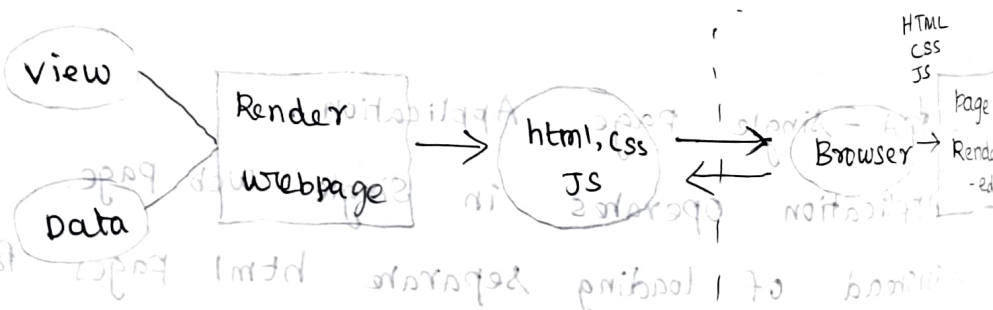
ex: Home

frame work:

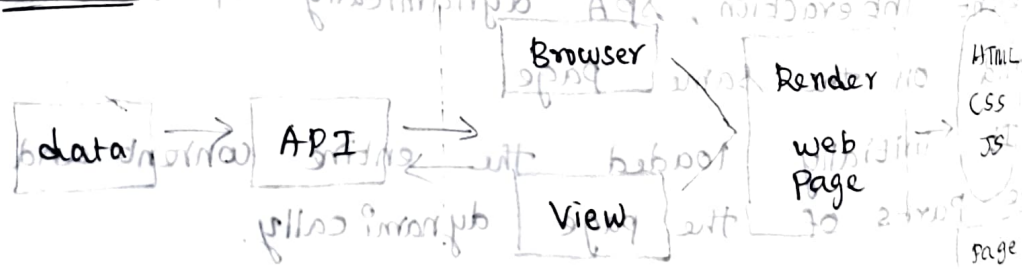
More no.of rules

ex: School

Before:



Current:



React - Install → node package

npx - Package installed in project Only. (npx execute)

npm - package installed in both system and project. (npm manager)

step by step:

* First Install node
* Next Create a folder in needed place.

Ex: fabery → react folder

* Next select the folder and open, next click the path bath, type cmd
* Now, the cmd.exe is open.

[Now, npm change to npx] → in some system

* Type npm install npx -g → global

Next npx (space) create-react-app file-name

Next Type Y (system ask yes (or) No, so put Y)

Virtual DOM:

copy of original Dom. update the specific part. not recreated. → using virtual DOM in RJ

function - component:

My file:

```
import React from 'react'

const functionname = () => {
    return (
        <div>
        </div>
    )
}

export default function Name.
```

App.js file

```
import functionname
    from 'myfile.js'

function App(){
    return (
        <div>
        <functionname/>
        </div> )
    }
```

class Component:

default export

Class

named export

```
import React, {Component} from 'react'

class Classname extends Component{
    render(){
        return (
            <div>
            </div>
        )
    }
}
export default Classname
```

App.js

```
import Classname from 'myfile.js'

function App() {
  return(
    <div>
    <Classname/>
    </div>)
  }
```

- Jordan Walke - React - Advandage: - Open Source

* Component based language. (render - rerun)
* Single page Application.
* Intial Name FAXJS
* developed by facebook 2011, instragram 2012 official release 2013
* reusable code

open - Next in Terminal:

cd filename.

export Home.js

*named export

Ex: export const x = 10

*defalut export.

Ex:
```
const x = 10
export default x
```

import.

* import {x} from './Home.js'

* import x from './Home.js'

Name - optional
because one default is

Attribute Name in React, it is called Prop

Style

Inline:

import './Home.css' → external file

Syntax: style = {{}}

For Ex:

```
return (
    <div>
        <h1 style={{color:"Red"}}> Hi </h1>
    </div>
```

Internal:

Ex:

```
const Home = () => {
    let HomeStyle = {
        Color : "Red",
        fontSize : "24px"
            }
return (
    <div>
        <h1 style={HomeStyle}> Hello </h1>
    </div>
    )}
```

Array of Object – List

```
const Home = () => {
    let a1= [{ name : "Tamil",
               work : "FED"},
             { name : "Durga"
               work : "BED"}]
return (
    <div>
      {
        a1. map ((a,b,c) => {
```

value ↑ position ↑ Array of obj →

only display Obj, JSON.stringify ()

Now, It display full array of object

```
return (
    <div>
        <h1> {a.name} , {b} </h1>
        <h2> {a.work} </h2>
    </div> ) } ) } ) }
```

Output:

document:

Tamil, 0
FED

Durga, 1
BED

inspect

Element

```
<div>
    <h1> Tamil, 0 </h1>
    <h2> FED </h2>
</div>
<div>
    <h1> Durga, 1 </h1>
    <h2> BED </h2>
</div>
```

Console:

Through Error msg. ⇒ key

ஒவ்வொரு time-லும் new, new dive-ஐ create ஆகும். So ஏதாவது change பண்ணாலும், delete பண்ணாலும் கூட, div எல்லாம் ஒரு மாதிரி இருப்பதால் எதை delete (or) change ஆனது-என்று கண்டுபிடிக்க முடியாது. So, unique-ஆ ஒரு key கொடுக்கனும். index is only gives unique value. So we index

For EX:

```
{ a.map ((a,b) ⇒ {
    return (
        <div>{
            <div key={b}>
                <h1> {a.name} </h1>
            </div>
        ) }
    ) }
```

Now, output: `<div key=0> <h1> Tamil </h1> </div> <div key=1> <h1> purga </h1> </div>`

Now, the error msg is gone. So use key.

icon- Package download:

command : npm i react-icons

---

import:

```
import {FaFacebook, FaWhatsapp} from 'react-icons/
```

JSX:

```
<FaFacebook /> <FaWhatsapp />
```

Nexted style:

Install: npm i sass

save: Home.scss

JSX

Java script extension that allows us to describe React's object tree using a syntax that ressembl that of an HTML template. it is just an XML Like extension that allows us to write JS that looks like markup and have it returned from a Component.

Props:

console.log (this) ⇒ this is one of the object, havi many key and value. There are,

- context
- props ⇒ this also object.
- refs
- updater.

How to get parent property in child.js

Parent.js Function component:

```
import React from 'react'

const Parent = () => {
```

```
Let x = 10;
Let a = [1,2,3,4]
Let b = [{
          name : "Durga",
          age : 20,
        },
        { name : "kalai";
          age : 22.
        }
        ].
```

```
return <section>          // Destructuring (use only spare)
        <child value={x}, array={a}
               obj={b} />
       </section>
}
export default parent.
```

Child.js:

class

```
import React, {Component} from
                'react'

class child extends Component{
  render() {          only use props
    console.log (this.props)
    Let card = this.props
return (
    <div>
      <h1> {card.value} </h1>
      <div>
        { this.props.name.map
            (a) => {
        return <div key={i}>
                {a.name}
                {a.age}
              </div> } </div> ) } }
```

function:

```
import React from 'react'

const child = ( collection) => {
                 obj & name
                 array
                 object
    console.log (collection)
return (
    <div>
      .obj.
    {collection.map((a,i) => {
      return (
        <div key={i}>
          {a.name}
        </div>
      )
      } }
    ) }
```

react: * it is a library. open source

* React run in SPA (Single page Application)

* component based Language. create package using command

* developed by FB -2011. Later Insta 2012

* released on 2013 - Jordan Walke

Component:

* reusable bit of code. 2 Type of component is Class component and function component.

* component Data - content, logic (JS), Appearence (style)

JSX:

* Javascript syntax Extension.

* It is used for Create HTML element in JS

DOM	Virtual DOM.
*actual structure of webpage	* copy of the original DOM.
*update the whole document	* update only the difference of the original DOM.
*re-render everytime	* doesnot re-render everytime. re-render Only state update.
*re render all state	

Any Packages:

* npm i react-icons npm i sass

* npm i react-slick

props: Count Increment the particular card:

Parent.js

```
import React from 'react'
import Child from './child.js'

const Parent = () => {
    let Phone = [
        { name: 'never',
          Color : 'Red'
          Count: 0.
        },
        { name : 'vivo',
          Color : 'yellow',
          Count : 0
        },
        { name: 'RealMe',
          Color : 'blue',
          Count: 0
        } ]

  let arr=[]
  let b = true

  let fun = (val,rate) => {
     if (b===true) {
       let x = Phone.find((a) => {
            return a.name===val ? {...a, count: a.count++} : ""
            arr.push(x)
            b= false
       }
     else {
          let tem = arr.find((e) => {
               return e.name===val
          })
        if (tem === undefined ) {
```

By Buy Now Click பண்ணும் போது 1st time card click பண்ணும் போது card Array push ஆகும். second time click பண்ணும் போது, count Mattum increase ஆகும் card push ஆகாது.

எல்லாத்தையும் பார்த்து அதில் val count மட்டும் change பண்ணும், மத்த property அப்படியே இருக்கும்.

→ diff card click பண்ணும் போது, arr-ல் already val card இல்லை match ஆகாது, so tem → undefined ஆகும்

```
Let z = Phone.find((a)=>{
        return a.name===val? {...a, count:acount++}:""
      })
    arr.push(z)
  }
else {
    arr.map((e)=>{
      return e.name===val? {e...e, count: e.count++}: a
      })
  }
}}

return <section>
      <h2> props </h2>
      <p> props means property </p>
      <child  mobile={phone} fun={fun} />
    </section>
export default Parent.
```

array and function pass

---

child.js:

```
import React from 'react'

const Child = (props)=>{

  return <section>
      <div className="con">
        {props.mobile.map((a,b)=>{
          return (
            <div key={b}>
              <h2>{a.name}</h2>
              <h2>{a.Color}</h2>
              <h2>{a.count}</h2>
            <div  className="btn" onclick={()=>
              props.fun{a.name, a.rate)}>
            Buy Now
          </div></div>
        </div></section>
export default Child
```

State

State	props:
* It is an object	* Object.
* mutable → change.	* immutable.
* use to create a dynamic website	* static
* can be update	* Cannot be update.
* cannot be passed.	* Can be passed parent component to child component.
Managed with in component	

State in function: → State use useState Hooks use use[name]

```
import React from 'react'
import {useState} from 'react'
Const parent = () => {
```

Hooks: simple JS function that we can use to isolate the reusable part from a functional Component. Added to React version 16.8

Ex: * use State * useNavigate → cannot declare (return) into the function

```
Const [name, setname] = useState("Tamil")
Const [age, setage] = useState("10")
return <section>
        <h1> {age} </h1>
        <h2 onClick={() => change = () => {
                setname("Resh")
            }>{name}</h2>
</section> }
export default parent.
```

before Click	after click:
10	10
Tamil	Resh

State in Class:

```
Import React, {Component} from 'react'
import './child.css'
class Child extend Component {
    constructor(){
        super()
        this.state={
            name:"tamil"
            }
        }
render() {
    Let handle=()=>{
        this.setstate({name:'kalai'})
        }
return <section>
    <h1 onClick={handle}>{this.state.name}</h1>
    </section>
    }}
export default Child.
```

constructor and super use parnanum

Update

function	Class
setname("any")	this.setState ({name:"any"})

Lifecycle

Each component in React has a lifecycle which you can Monitor and manipulate during its 3 main phases.

* Mounting *updating *unmounting.
* Initial phases

Class

```
import React {Component} from 'react'
in
class parent from extends Component {
    constructor (){
        super()
        console.log ("I am constructor")
        }
    componentDidMount () { → only one time execute
        console.log ("I am component did Mount")
        }
    componentDidUpdate (p,s) { → After update repeatly
        console.log ("update") execute
        }
render () {                    if (s.name != this.state.name)
    return <section>                log("name updated")
                                    }
        <h1> hello render execute </h1>
        </section>
    }
export default parent
```

Before Update	After update
* first constructor.	* first constructor
* Next render	* Next render
* Next Component Did Mount	* Next Component update
I am constructor	I am constructor
hellow render execute	hello reder execute
I am component did Mount	update.

React carousal

1) npm install react-slick

2) npm install react-slick-carousel → nav, footer any thing

import css:

```
import Slider from 'react-slick'  → slider
import 'slick-carousel/slick/slick.css'
import 'slick-carousel/slick/slick-theme.css'
```
} ← css

Changing Any:

```
var settings = {
    autoplay: true
}
```

Function - lifecycle

The useEffect Hook allows you to perform side effect in your component. Ex: fetching data, update DOM.

```
import React, {useState, useEffect} from 'react'
Let parent = () => {
    const [count, setCount] = useState(0)
    const [name, setname] = useState('Tamil')
    const Previous count = use previous (count) → previous count.
useEffect (() => {
    console.log ('Iam Mount')
}, [])
```
↓ dependency array

empty braces

```
useEffect (() => {
    console.log ('I am update')
}, [count])
```
↓ count update

```
return <div> </div> }
```

componentDidMount (p,s) {
  console.log (p,s)
}
↳ previous state, previous props

useEffect (() => {
  console.log (props, state)
}, [props, state])

First one → point out props, second one → point out state.

useEffect (() => { })	useEffect (() => { }, [ ])	useEffect (() => { }, [count])
↓ without empty, run every render	↓ run only once (dependency array)	↓ run every update (count)

Param / search param:

both are access the value from the url. but quary param it has key and value at the end of the url with ? character.

Difference:

Param – Search

1) Router: <path = "/" >
2) Home:
   nav ("/?id = ${val}")
3) Let [useparams] = usesearch Param()
4) using: params.get ("id")

Another level to pass state in the url.

Search param

Path = "/:id"

nav ("/${val}")

Let param = useparams

using: params.id

useNavigate:

Navigate between pages in your React Application use Navigate doesnot use into the fanction

For Ex: var a = useNavigate()

```
Let goto = () => {
    a('/Home') }
```

Another Method: <Link to = "/Home" >

Register Form:

```
import React from 'react'
import {useState} from 'react'
import {useNavigate} from 'react-router-dom'

const Reform = () => {
    Let nav = useNavigate()
  const [name, setname] = useState('')
  const [password, setpass] = useState('').
  const [Boo, setboo] = useState(false)

const obj = [ { resname : "raji"
                password : "12345" },
              { resname : "tamil",
                password : "aptal234" }]

const handle = (e) => {
        if (e.target.name === "name") {
            setname(e.target.value) }
      else if ( e.target.name === "pass") {
            setpass (e.target.pass) }
        }
const Sub = (e) => {
        e.preventDefault()
    if (name === "" || password === "") {
            setboo (true)
      }
    else {  Let a= false.
            Let b= false
```

find
map
some

```
obj.some((e) => {
        return  e.resname === name ? e.password ===
                password ? a=true : b=true : "" });
if (a===false && b===false){
        alert ('name not correct and unsuccess')
        }
else if (b==true) {
        alert ('password not correct')
        }
else { alert ('success')
       nav('/About')
     }
}
return <section>
   <form onSubmit={sub}>
        <input type="text" name="name"
               onchang={handle}/>
                              → error through tag
   { name==="" && boo ? <p> pls Enter the username </p>
     : ""
   }
<input type="password" name="pass" onchange={handle} />
   { password==="" && boo ? <p> pls enter password </p>:
     password !=="" && (password.length < 8) ?
       <p> password must be 8 Character </p> : ""
   }
  <button> submit </button>
  </form>
  </section> }
export default Resform
```

Random Question:

using props method

parent:

```
import React from 'react'
import {useState} from 'react'
import Child from './Child.js'

const parent = () => {
    const random = Math.floor (Math.random()*10)+1
const ques = [
    { id=0,
      ques:"what is React",
      Ans: "Library",
      boo: true
    },
   { id:1,
     ques:"what is component",
     Ans: "function and class",
     boo: true
    } ]
const [quesobj, setQues] =useState (ques)
Let quesansfun = (idval) => {
    Let ansdisplay = quesobj.map((e) => {
        return idval === e.id ? {...e, boolean:false,
            ans:e.ans, ques:e.ques} : ""
        setQues (ansdisplay)
        }
return <div>
        <h2> Ques & Ans </h2>
        < Child question={quesobj}
        </div> } quesans={quesansfun} />
```

export default Parent.

Child:

```
import React, {useState} from 'react'
const Child = (props) => {
  cons ran = math.floor (Math.random () * 10))
  var [condition, setcon] = usestate (false)
  return <section>
    <div>
    <button Onclick = {() => setcon (true)} > click </button>
    { props.question.map ((a,b) => {
      return condition === true ? a.id === ran ?
        <h2 > Onclick = {() => props.ques (a.id)} >
          {a.ques} </h2>
        : a.boo === true ? " " : <h3> {a.ans} </h3>
        : " "
    }) }
    </div>
  </section> }
export default Child.
```

Local storage card create using class

Form.js:

```
import {React, Component} from 'react'
import {Link} from 'react-router-dom'
class form from extends Component {
  constructor () {
    super()
```

```
this.state = {
        phonename:'',
        Phone edit: true,
         phoneid :''
        arr : JSON.parse (localStorage.getItem ('localobj')),
         index :''
         Phoneboo: true.
    } }
componentDidMount(){
    this.state.arr.find((e)=>{
        if (a.phoneedit ===false){
            this.setState ({phoneid : a.phoneid})
            this.setState ({phonename: a.phonename})
        } } )
    }
render () {
    const handle = (e) => {
        if (e.target.name ==='name') {
            this.setState ({phonename: e.target.value})
    else if (e.target.name ==='id') {
        this.setState ({phoneid: e.target.value}) }
const sub = (e) => {
    e.preventDefault()
var localobj = {
        phoneid: this.state.phoneid,
        phonename: this.state.phonename,
        phoneboo : this.state.phoneboo,
        phoneedit: this.state.phoneedit }
```

```
    Storedata (localobj)
  }
const  Storedata =(localobj) => {
   var  storeform = JSON. parse (local Storage .get('localobj'))
                                          || []
      Store form . push (localobj)
      var b = Store form .
Local Storage . setItem ('localobj', JSON. stringify (Store form))
    b= JSON. parse (localStorage .getItem ('localobj'))
b. map ((a) => {
   if (a.phoneedit === false) {
         localobj. phoneedit = true
        Let x = this. state. arr. map((val) => {
          return val. phoneedit === false ? localobj : v
            })
 localStorage . setItem ('localobj', JSON. strigify (x))
    }
 })
    this. setState ({Phonename: ''})
    this. setstate ({phoneid : ''})
  }
return <section> <form  Onsubmit = {sub}>
    <input  name = 'id'  onchange = {handle}
                value = {this. State. phoneid} />
<input  name = "name"  onchange = {handle}
                value = {this. state. phonename} />
    <button > Submit </button>
  </form > </section>

export default form.
```

Home.js: import {React, Component} from 'react'

import {Link} from 'react-router-dom'

class Home extends Component {

constructor() {

super()

this.state = {

obj: '',

arr: JOSN.parse(localstorage.getItem

('localobj'))

}}

render() {

const del = (e) => {

Let y = this.state.arr.filter((a) => {

return e !== e.phoneid

})

localstorage.setItem('localobj', JSON.stringify(y))

this.setState({arr: JSON.parse(localstorage.getItem

('localobj'))})

}

const edit = (id) => {

Let y = this.state.arr.map((e) => {

return e.phoneid === id ? {...e, phoneEdit:

e.phoneedit = false} : e

})

Localstore.clear()

localstorage.setItem('localobj', JSON.stringify(y)

}

```
return <section>
    { this.state.arr.map((a,i)=>{
        return a.phoneboo===true?
            <div key={i}>
                <h1> {a.phonename} </h1>
                <h1> {a.phoneid} </h1>
                <button onclick={()=> del(a.phoneid)}>
                    delete </button>
<button onclick={()=>editCard}>
<Link to="/" Onclick={()=>edit(a.phoneid)}>edit
                    </Link>
            </div>
    </section>
    }
export default Home.
```

Context - card - create: - function.

Form.js:

```
import React, {useContext, useEffect, useState}
    from 'react'
import {useSearchParams, useNavigate} from 'router..
import {globalState} from '../path'

export const form = ()=>{
    let [name, setname] = useState()
    let [about, setabout] = useState()
    let [id, setid] = useState()
    const {state, dispatch} = useContext(globalState
        let a = useNavigate()
```

```
const goto=()=>{
    a('/card')}

const handle = (e)=>{
    if (e.target.name = "name" ){
        Setname(e.target.value)}
  else if (e.target.name = "about") {
        Setabout (e.target.Value) }
  else if (e.target.name = 'id') {
        Setid (e.target.Value)}
        }
                                    -> useparams
  Let [params]= useSearch params()      -னா
useEffect =()=>{                        undefined-ன்னு
                                          இருக்கும்.
    if (params.get ('id') ! ==null) {   use Searchparam
        Let X= State.arr.find((e)=> {     -னா null.
            return  e.id === params.get('Id')
            })
        Setname (x.name)
        Set about (x.about)
        Set id(x.id) }, []}
 Let   Submit=(e)=>{
        e.prevent Default()
     Let  tem = {name, id, about}
   if (Params.get('id') !==null) {
        Let X = State.arr.map((e)=>{
          return eid === params.get('id') ? tem :e
          })
      dispatch ({type :'updatearr', payload : x })
```

```
else {
    dispatch ({type:'update arr', payload : [...state.arr,
                                              item]})
    }
  Setname ('')
  Set id ('')
  Set about ('')
}
return <div>
  <form  On Submit = {Submit} >
  <input  name ="name"  value={name}  onChange = {handle
<input  name = "about"  value = {about}  onchange={handle}/>
<input name='id' value={id}  onchange = {handle} />
  <button > Submit </button>
  </form>
  <button  onClick = {goto} > goto </button>
  </div
  }
```

---

card.js: import React, {useContext} from 'react'

```
import {global State} from 'path'
import {useNavigate} from 'react-router-dom
const Card = () => {
    Let {state, dispatch} = useContext (global State)
    let a = useNavigate ()
  const go = (id) => {
      a ('/?id=${id}')
    }
  return <section>
            <h1> Card </h1>
```

```
{ state.arr.map ((e,i) => {
   return  <div   key={i}>
           <h1> {e.name} </h1>
           <h2> {e.about} </h2>
          <button  onClick={() => goid()}> Edit </button>
     })
   }
 </div>
 }
export  default  Card.
```

Red-Bus:

Product.json:

```
{ "bus" : [
        {"id":1,
        "busseat" : false,
        "name" : "RB Travels",
        "count" :0,
        "place" : "Thenkasi"
        "Amount" : 660,
        "avai" : 17,
        "Seat": [
            {"id":1,
             "selected": false,
             "booked": false},
           { "id": 2,
             "selected": Truefalse;
             "booked" : True },
                  .... ] },
```

```
{"id": 2,
"busseat": false,
"name" : "VKV Travels",
"time" : "9.54",
"total": 18,
"place": "chennai",
"Amout": 850,
"count": 0,
"avai" : 14,
"seat": [
        {"id": 1,
         "selected": false,
         "booked": false,
         },
        {"id": 2,
         "selected": false,
         "booked": True
         }
 ]}
 ]
}
```

Context.js:

```
import {createContext} from "react"
Const globalstate = CreateContext()
export default global state.
```

Reduce.js:

```
import data from '../product.json
```

```
export const initialstate = {
        buses : data.bus,
        count : 0,
    }
export const Reduce = (state, action) => {
    if (action.type === "updatebuses") {
        return {... state, buses: action.payload }
    }
    if (action.type === "updatebus1") {
        return {... state, bus1 : action.payload }
    }
    if (action.type === "updatecount") {
        return {... state, count: action.payload }
    }
}
```

Routing.js:

```
import React, { useReducer } from 'react'
import globalstate from './context.js'
import {initialstate, Routes, Route } from "./Reduce.
import { BrowserRouter, Routes, Route } from "react-
import Na                                  router-dom
import Home from "./Home/Home.js"

    const Routing = () => {
    let [state, dispatch] = useReducer (Reduce,
                                        initialstate)
return <div>
    < globalstate.provider  value= {{state, dispatch}
    < BrowserRouter >
    < Routes > < Route path='/' element = {<Home />
    </Routes>
    </BrowserRouter>
    </globalstate.provider> </div> }
export default Routing
```

App.js:

```
import './App.css'
import Routing from './component/Routing.js'

function App(){
    return <div>
        <Routing />
        </div>
    }
export default App.
```

Home.js:

```
import React, {useContext, useState} from 'react
import globalState from '../context.js'
import './Home.css'

const Home = ()=>{
    const {state, dispatch} = useContext(globalState)
    var [amount1, setAm1] = useState(0)
    var [amount2, setAm2] = useState(0)

  const viewseat = (busid)=>{
    Let x = state.buses.map((e)=>{
      return e.id === busid ? e.busseat === false ?
          {...e, busseat: true} : {...e, busseat: false}:e
      })
    dispatch({type: "updatebuses", payload: x})

  const seatbook = (id, busid)=>{
    let y = state.busses.map((a)=>{
      return a.id === busid ? ({
        seat: (a.seat.map((e)=>{
          if (e.id === id){
            if (e.selected === false){
```

```
dispatch ({type: "updateCount" , payload: state.count>6?
                state.count: state.count +1})
busid ===1 ? setam1 (amount1 +660) : setam2 (amout2
                                          + 850)
    return {...e, selected: true}
      }
else {
    dispatch ({type: "updateCount", payload: state.
                                          count-1})
  busid ===1 ? setam1 (amount1 - 660 ) : sekam2 (
                                amout2 - 850)
    return {...e, selected: false}
      }
else {
    return e
      }
  })
  )} ) : a
})
   dispatch ({type: "updatebuses", payload: y})
}
if (state.count>6){
    alert ("can't book more than 6 seat")
  }
let booked = ()=>{
    alert ("can't book more than 6 seat")
  }
const booking = (id)=>{
    let x = state.buses.map ((e)=>{
      e.id ===id ? e.id ===1 ? setam1(0):
            setam2(0) : setam2 ( amount2)
    return e.id ===id ? ({
```

```
seat : e.seat.map((a)=>{
  return a.selected===true? {...a, selected:false,
    booked:true}:a
  })
 }):e
})
dispatch({type:"updatecount", payload: state.count===0
dispatch({type:"updatebuses", payload: x})

let not=()=>{
   alert("oops! This seat is not available")
}
return <div style={{cursor:"pointer"}}>
  <div>
    {
      state.buses.map((e,i)=>{
    return <section key={i}>
      <div><h3>{e.name}</h3>
      </div><p>{e.time}</p>
        <p>{e.place}</p>
      </div>
      <div><p>start from</p>
        <h4>Rs. {e.amount}</h4>
      </div>
      <div><h3>Total seat is {e.total}</h3>
        <p>{e.avai} Available</p>
      </div>
      </div>
  { e.busseat === false?
      <div>
        <div onClick={()=>viewseat(e.id)}>
          view seat
        </div>
```

```
<div key={i}>
  <div onClick={()=> viewseat(e.id)}> Hide seat </div>
<div> <div>{
        State.count <6 ?
    e.seat.map((a,i)=>{
    return a.selected === false ?
      (a.booked === false ?
        <div onClick={()=> Seatbook(a.id, e.id)
style={{ border:"2px solid black" }> </div>:
  <div> onClick={not} Style={{backgroundColor: "gray"}}
    Seat {i+1}
  </div>:
<div> Onclick={()=> Seatbook(a.id, e.id)}
    Style={{backgroundColor: "black"}}>
     Seat {i+1}
</div> }) :
  e.seat.map((a,i)=>{
  return  a.selected === false ?
    a.booked === false ?
<div .Onclik={booked} </div>:
<div onclick={not}  Style={{backgroundColor: "gray">
     Seat {i+1}
</div>
<div' onClick={()=> Seatbook(a.id, e.id)}
          Style={{background: "black"}} seat {i+1}</
  })
}
</div>
<div> <h1> Amount </h1>
    { e.id ===1 ? <p> {amount1} </p> : <p> {amount2}
</div>                                          </p>
```

```
</div>
<div>
    <p onClick={()=> booking(e.id)}>proceed</p>
</div>
</div>}
</section>})
}
</div></div>}
export default Home.
```

shouldComponentUpdate

This Method allows us to exist the complex react update life cycle to avoid calling it again & again. on every re-render. It only updates the component if the props (or) state Chages.

Syntax:

```
shouldComponentUpdate(nextprops, nextstate)
```

(passed value : changed)

componentWillUnmount()

It allows you to perform any neceessary cleanup, such as ~~cookie~~ canceling timer, removing Even Listeners (or) clearing any data structes that were setup during the mount phase. It invoked immediately before component is unmounted and destroyed.

unmount:

Components are unmounted, when the parent Component is no longer rendered (or) the parent Component performs an update that does not render this instance.

componentWillMount() / componentDidMount() – difference

Invocation timing of componentDidMount() and the deprecated componentWillMount.

After rendering and mounting the component onto the DOM, componentDidMount() gets called.

On the other hand, componentWillMount() is called before the components is mounted.

Ex:

```
import React, {Component} from 'react'
class ComponentdidChild extends Component {
    constructor() {
        super()
        this.state = {
            name: "child",
            age: 10,
            count: 0 }
    }
shouldComponentUpdate (nextprops, nextstate) {
    if (this.state.name !== nextstate.name ||
        this.state.age !== nextstate.age ||
        this.state.count !== nextstate.count) {
            return true }
    else {return false }
    }
ComponentwillUnMount() {
        clearInterval (this.timer)
                }
ComponentDidMount() {
        this.timer = setInterval (()=> {
            this.setstate ({count: this.count +1})
                }, 1000)
                }
```

```
render() {
    Let handle = () => {
        this. setstate ({name: "life cycle"})
                    }
    Let handleage = () => {
        this.setstate ({age: 30})
                    }
return  <section>
        <h1> {this.state.name} </h1>
        <button onClick= {handle} > Click </button>
        <h1 > {this.state.count </h1>
    </section>
  }}
export  default  componentdidChild.
```

Function Component – Memo:

using memo will cause React to skip rendering a component if its props have not changed. designed to re-render whenever the state (or) props value changes.

Function – memo:

```
import  React, {useEffect, usestate} from "react"
import  {memo} from "react"

const  use = () => {
        Let  [name, setname] = usestate ("child")
        Let  [count, setcount] = usestate (0)

    useEffect (() => {
        Let x = setInterval (() => {
            setcount (count) => count + 1))
                (3, 1000)
```

```
    return () => {
        clearInterval(x)
    }
}, [])

return <section>
    <h1> {name} </h1>
    <button onClick={() => setName("React")}>
        Click </button>
    <h2> {count} </h2>
</section>
}

export default memo(use)
```

---

core principle of redux?

* Single source of truth * The state is read Only * Changes are made with pure functions.

getState is retriever the current state of our redux store.

My SqL (X) cross platform.

* My Structured Query Language.
* MySQL relation database Management system.
* Open source. Speed, Easy to use, security
* MySQL is cross platform.
* released in 1995.
* develop, distributed and supported by Oracle.
* My → Co-founder daughter name
  ↓ monty ↙ widenius's
* Founder - Swedes David Axmark, Allan Larsson.

context:

context is primarily used when some data needs to be accessible by many components at different nesting levels. (Avoid props-drilling)

used to parent component to deep level child component.

---

2 folder → one is set to global variable, another one is change format.

3 Type of file create:

create a global State:

context.js: import {createContext} from 'react'

```
const globalState = createContext()
export globalState.
```

* globalState gives provider and consumer.

New version, we context (Hook) is replace the consumer place.

Reducer.js: → update the data

```
export const initialState = {
        name: "tami",
        arr: [1,2,3,4]
    }

export const Reducer = (state, action) => {
    if (action.type === "updatearr") {
        return {...state, arr: action.
                                payload}
    }

console.log(action) → {type: " ",
                       payload: ""}
```

Routing.js: import {BrowserRouter, Route} from 'react-router-dom"

import {useReducer} from "react" → useReducer ஐ import initialState, Reducer set பண்ணுவதற்கு.

import {globalState} from "./path"

import {initialState, Reducer} from "path"

const Routing = () => {

↗ initialState ↗ updateReducer.

Let [state, dispatch] = useReducer(Reducer, initialState)

இந்த line ல, state ல initialState-ஐ, dispatch-ல Reducer இருக்கும். change ஆன உடனே.

return (

<globalState.provider value={{state, dispatch}}>

↙ global ஆ variable set ஆகி விடும்.

<BrowserRouter>

<Routes>

<Route path='/' element={<home/>}></Route>

</Routes>

</BrowserRouter>

</globalState.provider>

)

export default Routing;

---

which file ல set global variable

home.js:

globalState use பண்ணுவதற்கு help பண்ணுவது hook ↑

import React from 'react'

import {globalState} from "path"

import {useContext} from 'react'

```
const Home = () => {
  let { state, dispatch } = useContext(global.state)

  const hand = () => {
    dispatch({type:
      var a = [5,6,7,8].
    dispatch({type : 'update arr', payload : a})
    }

return <div> onClick = {hand} >
  { state.arr.forEach ((e) => {
      <h1> { e } </h1>
    }
  </div>
}
```

output:

before update

1
2
3
4

After update:

5
6
7
8

---

useReducer - Hook:

*The reducer function contains your custom state logic and initial state can be simple value but generally will contain an object

*It returns state and dispatch.

*It lets you add a reducer to your component.

two type of reducer.

Concentric and eccentric

Redux: Redux is Open Source JS Library for managing and Application State. Used for Share a data between component.

3rd part Library.

Package: npm install @reduxjs/toolkit react-redux

App.js:
```
import {provider} from 'react-redux'
import Routing from './Routing.js'
import {Store} from '/store.js'

function App() {
    return (
        <div>
            <provider store={Store}>
                <Routing />
            </provider>
        </div>
    )
}
export default App.
```

Store.js:
```
import {configureStore} from '@reduxjs/toolkit'
import sliceReducer from './slice.js'

export. const Store =
    configureStore({
    reducer: {data: sliceReducer}
    })
```

Slice.js:
```
import {createSlice} from '@reduxjs/toolkit'

const Slice =
```

```
createSlice ({
    initialState : {
        name : "tami"
    },
    reducers : {
    handlename : (state, action ) => {
        state.name = action.payload
    }
    }
})
export default Slice.reducer
export const {handlename} = Slice.actions.
```

Home.js:

```
import React ,from 'react'
import {useDispatch , useSelector } from 'react-redux'
import {handlename} from './slice.js'

const Home = () => {
    Let state = useselector ((e) => e)
    Let dispatch = useDispatch()
    Let change = () => {
    dispatch (handlename (" kalai "))
    }
return <div>
    <h1 onClick ={() => change} > {state.data.name}
    </h1>
    </div>
}
export default Home.
```

1) what is React? (open source)

* powerful JavaScript library, not a framework. React.js operates only in the view layer of the Application Stack. Component based language. So It can be reusable. Single page Application.

* Created by Jordan walke, 2013 open source in Brower, But since 2011 (by facebook)

2) React features?

* Declarative syntax: It's Allows developers to describle, how the UI should look based on the application state.

* Component-based Architecture: It is Modular and reusable approach by breaking down the UI into small, self-contained component.

* virtual DOM: Copy of the original DOM. If you update in original DOM. It compare both, real and virtual DOM. then it update only the particular part, not entire web Application.

* One way data binding: React.js is unidirectional data flow, ensuring that data changes are predictable and easier to debug. It improves code maintainability and reduces the likelihood of bugs.

* JSX: JSX an JavaScript XML, JSX acts as bridge between HTML syntax and JS code in React.js, empowers developers to create elegant and expressive UI components with ease.

Allow to write HTML in JS without using createElement() (or) appendChild().

3) Dis Advandge of React.js?

1) JSX as a barrier: HTML Mixed with JS can be barrier.

2) Due to fast development there is no proper documentation.

3) It is covers only the UI layer. So there is a need for other platforms. cost can be high

4) Staying up-to-date can be difficult

4) Tool Chain?

1) create-React-App - SPA Oriented toolchain.

2) Next.js Server side rendering Oriented toolchain.

3) gatsby - Static Content Oriented tool chain.

5) React.createElement?

* Element tag * Element attribute as object

* Element content.

6) ReactDOM.render

React Element (or) JSX.

Root element of webpage.

Ex: <Script>

element = React.createElement ('div', {},

React.createElement ('h1', {}, 'Hello'))

ReactDOM.render (element, document.getElementById ('react-app'))

</Script>.

7) 3rd party Component

1) Router Management - React Router.

2) React - Animation - React-Transition-group, React Reveal.

3) React - Advance - State - Management - Redux, Mobx, Recoil.

4) React - Rest - Management - JS Fetch, API, etc.

8) React - StrictMode ?

It is a build-in component used to prevend unexpected bugs by analysing the component for unsafe lifecycle, unsafe API usage.

App.js - Root component of the application.

Package Manager - High level management of application. Ex: npm (default)

9) React component gives

1) Initial rendering of the user interface.
2) Management and handling of events.
3) updating the UI whenever the internal state is changed.

10) render ?

This Method defined in Every Component. responsible for generating the JSX markup of the component. Responsible for returning a single root HTML node element. If you don't want render anything, you can return a null (or) false value Everytime render, if the state (or) props update.

11) Component ?

Component is independent and reusable bit of code. They serve the same purpose as JS. but work Isolation and return HTML. 2 Type.

1) function Component 2) Class Component.

Version React 16.8 Function Components were considered as "State-less"

12) Constructor()
This method used to initialize an object's state in a class. It is automatically called during the creation of an object in class.

13) super (key)
In react-js it is used to access the parent properties and methods from the child component. super() function is to call the constructor of the parent class. used to access the few variables in parent class.

Handling two way to data objects in React
1) state 2) props.

14) Why this cannot be allowed before super()
because this is uninitialized if super() is not called.

15) this - keyword.
this keyword used to refer to the current instance of a component class.
Access props (or) state using class component, we need to use "this" keyoword. not necessary to use "this" in function component (console.log("this))
function → undefined.

16) import and Export!
import: It allows using contents from another file. export: It is make the file contents eligible for importing. Named export (or) default export.

17) state: The state is Local Data storage that is local to the component only, cannot be passed to other components. set state property used to update the values in component. It is mutuable.

State in class: The state Object is where you store property values that belongs to the component. when the state Object changes, the component is re-render.

State-in-function: useState Hook is allows you to add state to a functional component. It returns an array with 2 values.
~~One is current state and another one is~~
Const [state, setstate] = useState(initial value)

Props: Props Means property. props are a Type of Object, where the value of attribute of a tag is stored. It is read-only. It passed to one component to anothercomponent. It is immutable

Props drilling: prop drilling occurs, when a parent component generates its state and passes it down as props to its Children Components. data passes from parent to deep level Child. it is called props drilling.

Hook: version 16.8. Hooks allow function component to have access to state and lifecycle methods and other react features.

useState: Allows you to add state to a functional Component. It is work handling and managing State in your application.

useEffect: Allows you to perform side effect in your Component. Ex: fetching data, update DOM (directly) and timer.

useEffect (Setup, dependencies). Accept 2-arguments
1) props 2) State.

useRef: use useRef to track application renders.
useRef only return one item. It return an object called current. when we initialize useRef we set initialvalue initial value useRef (0) (or) null
* unlike useState (or) useEffect Changes to useRef don't trigger re-renders of the component. Let's reference a value, not needed for render

useNavigate: It is help trigger a navigation event from within a component. useNavigation not directly used, if you want, 1st store useNavigation() in variable. Ex: var a = useNavigation()
* cannot be called inside of a callback.

Key: Special String Attribute you need to include when creating lists of elements in React.
It gives identity to the elements in the lists.

Lifecycle: Each Component in React has a lifecycle, which you can monitor and manipulate during its three phases. Mouting, updating, unmounting.

Mounting: Mounting means putting elements into the DOM. 1) constructor() 2) getDerivedStateFromProps()
3) render() 4) componentDidMount().

componentDidMount: It called Mounting phase.
TO perform any setup tasks or face effects required for the element to work decently.
whenever you enter the application its invoke immediately Only once.

componentWillMount(): The function is called before the Component gets loaded in the DOM tree.

componentDidupdate(): Allows us to run React code when the component is updated. it involke whenever props (or) state update.

ShouldComponentsUpdate(): It helps in checking whether the re-rendering of component is required (or) not.

Syntax: ShouldComponentUpdate(nextprops, nextstate)

componentWillUnmount(): involke immediately before a Component is unmounted and destroyed.
perform any necessary cleanup such as invalitating timer, cancelling network requests.

Memo and useMemo: Memo can be imported from 'react' and wrapped around the function component. useMemo is a hook that lets you to cache the result of calculation between re-renders unless memo avoid re-render if any props (or) state changes.

Routing: process of navigation between different pages without triggering a full page reload.

useparams: It is helps to access the parameter of the current route to manage the dynamic routes in the URL. It returns an object of key/value pairs.

useSearchparams: Allows you to access and modify the query parameters in a URL.

3 Type of router?

* Memory router * Browser Router * Hash router

Router?

* use to create a single page web Apps
* Router is utilize to define various routes.

Browser Router?

* store the current location in the browser's address bar, using clean URL and navigate using browser's built-in history stack.

route?

It render some UI, if the location matches the current route path.

provider?

provider is basically a container (or) a storage that stores and provides you with share (or) data.

Context → value = {{ state, dispatch }}

Redux → Store = { store }

reducer? Everytime calculate new state result.

The function you pass to reduce is known as a 'reducer'. It takes current item, then return the next result. 2 types * Concentric
* eccentric.

Context (Avoid props drilling)

Allow us to pass data through our components tree. It gives the components ability communicate and store and share data different level.

create Context?

returns a context object. The object does not hold any information.

use Reducer?

This hook used to store and update states. It accepts a reducer function as 1st parameter and initial state as 2nd parameter.

return array, that holds the current state and dispatch function which you can pass an action and later invoke it.

useContext?

Hook provides function components access to the context value for a context object. return current context value as given by the nearest context provider.

dispatch: The function accept an object that represents the type of action, we want to execute when it's called.

Redux

It is an Open Source JS Library for managing application State. used for share a data between component. 3rd party Library.

configure Store?

Standard Method for Creating a Redux Store. not only create a Store but also accept reducer functions as arguments. automatically setup the Redux DevTools Extension for easy debuggi

createSlice?

The function accepts an initial state, an object of reducer and automatically gener action creators and action types that correspon to the reducers and State.

useDispatch?

hook to dispatch actions that modify the State in the redux Store. avoid harder to test and reuse. use actions creators to create ac that are dispatched by the useDispatch hook.