

What is React?	2
Start Learning React	5
React Components	8
Types of Components	9
Props	12
Props in Functional Components.....	12
Props in Class Components	18
Children Props.....	21
States	22
States in Class Component.....	22
Event Handling (Changing State).....	24
Asynchronous setState	25
States in Functional Components.....	27
This keyword binding in Class Component.....	29
Events Handling	31
List Rendering	34
Conditional Rendering	36
V-DOM (Virtual DOM)	38
Component lifecycle in React	38
In Class Components.....	39
componentDidMount().....	39
componentDidUpdate().....	42
Prevent infinite api request in componentDidUpdate Method.....	43
componentWillUnmount().....	46
In Functional Components.....	50
useEffect hook (lifecycle hook).....	50
Using sideEffect in useEffect hook.....	54
Context Api	57

React Learning

Create Context API.....	61
Provide Context API and Pass data with Value props.....	61
Consume Context API.....	62
With Consumer method.....	62
With useContext Hook.....	65
useReducer Hook.....	66
React Styling.....	70
Inline Styling.....	70
External CSS Styling.....	71
Module Styling.....	71
React Router.....	72
Additional React-Router-Dom.....	74
useNavigate Hook.....	76
useParams Hook.....	77

Github Repo > [eBook Link](#)

What is React?

React ကို ကျတော်တို့ စပြီးတော့ မလေ့လာခင်မှာ

Javascript Language ကို အောက်မှာပြောထားတဲ့အရာတွေကိုလေ့လာထားခဲ့ပြီးပြီဆိုရင်တော့
လေ့လာလို့ရပါပြီ

အရင်ဆုံး ကျတော်တို့ React ကို မလုပ်သေးခင်မှာအရင်ဆုံး သူ့အကြောင်းလေးကိုနည်းနည်းလေး တော့
လေ့လာလိုက်ကြရအောင်ဗျ

React ?

အခုနောက်ပိုင်း မြန်မာနိုင်ငံမှာ React ဆိုတာကို Web development ဖက်မှာ တော်တော်လေး hot
ဖြစ်လာနေတဲ့ tech stack တခုပါပဲ

သူက ဘယ်လိုမျိုးလည်းဆိုတော့ javascript ရဲ့ library လေးတစ်ခုပါပဲ

သူ့ကို လက်ရှိမှာက maintain လုပ်ထားတာက meta (facebook) ကနေပြီးတော့လုပ်ထားတာပါ

အဲဒါကြောင့်မလို့သူက community တောင့်ပြီးတော့ တော်တော်လေး hot ဖြစ်နေတာပါ

ပြီးတော့နောက်တစ်ခုက သူက လေ့လာရတာ လွယ်တာဖြစ်တဲ့အတွက် ပိုပြီးတော့ popular ဖြစ်တာပါ

Beginner friendly ဖြစ်တော့ တော်တော်လေးကို fast learning လုပ်နိုင်ပါတယ် React က

React Learning

အဲဒါကြောင့်မလို့ မြန်မာနိုင်ငံမှာတော်တော်လေး hot ဖြစ်လာနေတာပါ
သူကနေပြီးတော့ ကျွန်တော်တို့ တွေက ဘယ်လိုမျိုး ထွက်လာလည်းဆိုတော့ frontend ဖက်ဆိုရင်
React / React-native (mobile) / Next JS

Backend ဖက်ကိုကျတော့ nodeJS / ExpressJS

Database ကို ကျတော့ Mongo DB (NOSQL) အသုံးပြုပြီးတော့

MERN (Mongo , Express , React , Node) ဆိုပြီးတော့ full tech stack ဖြစ်လာတာပါ

ကဲ ကျတော်တို့ အရင်ဆုံး React ကို မလေ့လာခင်မှာ ဘာတွေလိုအပ်မလဲဆိုတော့

Javascript ကို ES6 module တွေ es6 ရဲ့ ပြောင်းလဲသွားတာတွေ ကို တော်တော်များများ

သိထားရပါမယ်

ပြီးတော့ javascript array method တွေဖြစ်တဲ့ ဥပမာ *map / filter* တို့လိုမျိုးကို အနည်းဆုံးတော့
သိထားဖို့လိုပါမယ်

Function တွေအကြောင်း return တွေ အလုပ်လုပ်ပုံရယ်ကို သိထားဖို့လိုပါတယ်

မသိသေးဘူးဆိုရင် သွားပြီးတော့လေ့လာကြည့်ကြည့်ပါ

Link >> [Javascript](#)

ကျွန်တော်တို့ အရင်ဆုံး React ကို မလုပ်ခင်မှာဘာတွေလိုအပ်သေးတာလည်းဆိုတော့

စက်မှာ node js လေး install လုပ်ဖို့လိုပါမယ်

ပြီးတော့ vs code လေး ရှိရပါမယ် (ရှိမယ်လို့လည်းထင်ပါတယ်ဗျာ)

ရှိသွားပြီဆိုရင် ကျတော်တို့ react project လေးတစ်ခုကိုအရင်ဆုံး create လုပ်လိုက်ကြရအောင်ဗျ
ဘယ်လိုမျိုးလုပ်ကြမလဲဆိုတော့

ကျတော်တို့ react project ကို create လုပ်တဲ့ နည်းတွေအများကြီးရှိတဲ့အထဲမှာ မှ ကျတော်က တော့
လက်ရှိမှာ vite js ကို သုံးပြီးတော့ လုပ်ပြပါမယ်

ပထမဆုံး ကျတော်တို့ create လုပ်ချင်တဲ့ နေရာမှာ command prompt / terminal / git bash တစ်ခုခု
ကိုဖွင့်လိုက်ပါ ပြီးရင်တော့ ကျတော်တို့ က ဒီcommand လေးရိုက်ပီးတော့ enter လေး ခေါက်လိုက်ပါ

```
>>> npm create vite@latest
```

အဲလိုလေးရိုက်လိုက်ရင် ကျတော်တို့ကို သူက project name လေးထည့်ခိုင်းပါလိမ့်မယ်

အဲမှာကျတော်တို့ ကြိုက်တဲ့ name ပေးလို့ရပါတယ်

Name ပေးတဲ့အချိန်မှာ ကျတော်တို့ သတိထားရမှာက uppercase letter ရေးလို့မရပါဘူး

ပြီးတော့ name ကို react / react-native / next / node အဲလိုမျိုး package name တွေ

သွားပေးလို့မရပါဘူး

Dash (-) လေး သုံးပီးရေးလို့ရပါတယ်

Camel Case ကို မသုံးခိုင်းပါဘူး ဗျ

React Learning

ပြီးတော့ အစစလုံးကို number တွေ သုံးလို့မရဘူးဗျ

ဒါဆိုရင်တော့ကျတော်တို့ က ဘယ်လိုမျိုးရေးမလဲဆိုတော့

react-lesson1 အဲလိုမျိုးရေးပြီးသွားရင် enter နှိပ်လိုက်ပါမယ်

အဲကျရင်သူက ကျတော်တို့ကို select လုပ်ခိုင်းပါလိမ့်မယ် arrow down key လေးနဲ့ react ကိုရွေးပီးတော့ enter ခေါက်လိုက်ပါ

ပြီးသွားရင် ကျတော်တို့ကို typescript လား javascript လားရွေးခိုင်းပါလိမ့်မယ်

Javascript ရှိရိုးကိုပဲ ရွေးပီးတော့ enter ခေါက်လိုက်ပါ

အခုဆိုရင် project ကို create လုပ်လို့ပီးသွားပါလိမ့်မယ်

```
USER PC@DESKTOP-6MD87NH MINGW64 /d/KMH Data/test/react-swiper
$ npm create vite@latest
✓ Project name: ... react-lesson1
? Select a framework: » - Use arrow-keys. Return to submit.
  Vanilla
  Vue
>  React
  Preact
  Lit
  Svelte
  Solid
  Qwik
  Others
```

ပုံ ၁

```
USER PC@DESKTOP-6MD87NH MINGW64 /d/KMH Data/test/react-swiper
$ npm create vite@latest
✓ Project name: ... react-lesson1
✓ Select a framework: » React
✓ Select a variant: » JavaScript

Scaffolding project in D:\KMH Data\test\react-swiper\react-lesson1...

Done. Now run:

  cd react-lesson1
  npm install
  npm run dev

USER PC@DESKTOP-6MD87NH MINGW64 /d/KMH Data/test/react-swiper
$
```

ပုံ ၂

အဲလိုလေးပေါ်လာပြီးသွားရင် ကျတော်တို့ project folder ထဲကို ဝင်လိုက်ပါမယ်

ပြီးသွားရင်တော့ npm install လုပ်လိုက်ပါမယ်

Install လုပ်လို့ပြီးသွားရင် ကျတော်တို့

Vscode လေးကို code . လေးနဲ့ ဖွင့်လိုက်ပါမယ်

ပြီးသွားရင်တော့ ကျတော်တို့ vscode ရဲ့ terminal မှာ npm run dev လေး ရိုက်လိုက်ပါမယ်

အခုဆိုရင် သူက localhost:5173 မှာ live server လေး run ပြီးတော့

React project လေး ပွင့်လာပါလိမ့်မယ် ပွင့်လာတဲ့ project ကို ကျတော်တို့ ဖွင့်ချင်တယ်ဆိုရင် browser မှာ localhost:5173 လို့ရိုက်ပီးတော့ ဖွင့်လို့လည်းရသလို

သူပြလိုက်တဲ့ localhost:5173 ဆိုတာလေးကို ctrl+left click နှိပ်ပြီးတော့လည်း ဖွင့်လို့ပါတယ်ဗျ

အဲလောက်ဆိုရင် ကျတော်တို့ react project တစ်ခုကို create လုပ်လို့ပြီးသွားပါပြီဗျ

Start Learning React

ဒီ section မှာက ကျတော့ ကျတော်တို့ က React ကိုစတင်ပြီးတော့ လေ့လာကြရအောင်ဗျ

React က Component based architecture ဖြစ်တော့ ကျတော်တို့ အနေနဲ့ web page ပဲဖြစ်ဖြစ် web application ပဲဖြစ်ဖြစ် ရေးကြတဲ့အခါမှာ အကုန်လုံးကို ကျတော်တို့က ခွဲခြမ်းစိတ်ဖြာပြီးတော့

ခွဲရေးလို့ရသွားတယ် ဗျာ

ပြီးတော့အဲလိုမျိုးရေးလိုက်ခြင်းအားဖြင့် ဘာအကျိုးကျေးဇူးရသွားလည်းဆိုတော့ ကျတော်တို့က code တွေကို ခန့်ခန့် ပြန်ရေးနေစရာမလိုတော့ပါဘူးဗျ

React Learning

အဲလောက်လေး နားလည်ထားရင်တော့ beginner အပိုင်းကတော့ တော်တော်လေး အဆင်ပြေနေပါပြီ ဗျာ

ကဲ နောက်တမျိုးကို ဆက်လေ့လာကြရအောင်

ကျတော်တို့ react project တခုကိုကြည့်လိုက်ရင် ဘာထူးဆန်းတာတွေ့ရမလဲဆိုတော့ သူ့မှာ projectတစ်ခုလုံး မှာ ကြည့်လိုက်ရင် ကျွန်တော်တို့တွေ့ရမှာက html file လေးက တစ်ခုတည်းပဲရှိတာပါပဲ ပြီးတော့ ကျတော်တို့ က react ကို နောက်တမျိုးခေါ်ကြတာက SPA (single page application) လို့ခေါ်ကြတယ်ဗျာ

ဘာလို့ဆိုတော့ သူက html file တခုတည်းမှာပဲ လိုအပ်တဲ့ code တွေအကုန်လုံးကို javascript ကနေပြီးတော့ run ပေးသွားတာမလို့ပါ

ဒါဆို ကျတော်တို့ react project တွေမှာ တွေ့တွေ့နေရတဲ့ html code တွေက html မဟုတ်ဘူးလားလို့ မေးစရာ ရှိလာပါတယ်

ဟုတ်ပါတယ် အဲဒါတွေက html code တွေမဟုတ်ပါဘူး

ကျတော်တို့ကို သုံးရလွယ်အောင်ဆိုပြီးတော့ JSX (javascript XML) ဆိုတဲ့ tech လေးကို အသုံးပြုထားတာပါ

သူက ဘယ်လိုမျိုးအလုပ်လုပ်ပေးတာလည်းဆိုတော့ ကျတော်တို့ developer တွေက ကျတော့ ရေးတဲ့အခါကျရင် html code တွေလိုမျိုးရေးလို့ရတယ်ဆိုပေမယ့် သူ့အလုပ်လုပ်ပုံက javascript ကနေပြီးတော့ html တွေကို createElement တွေနဲ့ ပြန်ထုတ်ပေးတာပါ မြင်သာအောင်ပြောရရင် compiler လိုမျိုးပေါ့ဗျာ

JSX ကို အသုံးပြုထားတာမလို့ ကျတော်တို့ က React ကို လေ့လာရတာပိုပြီးတော့ လွယ်ကူသွားစေတယ်ဗျာ

အဲအကြောင်းနဲ့ပတ်သက်ပီးတော့ လေ့လာချင်ရင်တော့ ဒီမှာသွားဖတ်လို့ရပါတယ်

Link [JSX](#)

ကျတော်တို့က JSX ကို မသုံးချင်ဘူးဆိုရင် react ကနေပြီးတော့ သူ့မှာ

သုံးလို့ရတဲ့ဟာလေးတွေလည်းရှိပါတယ်

တကယ်လို့ရေးချင်တယ်ဆိုရင် ရေးလို့ရပါတယ်ဗျာ

ဒီကုဒ်လေးကို ကြည့်လိုက်တဲ့ အခါကျရင် ဒီဟာ နှစ်ခုက

ဘာလိုကွာသွားလည်းဆိုတာကိုမြင်ရပါလိမ့်မယ်ဗျာ

React Learning

```
const JSX = () => {  
  return (  
    <h1>Hello</h1>  
  )  
}  
  
import React from 'react'  
const notJSX = () => {  
  return (  
    React.createElement("h1", null, "Hello")  
  )  
}
```

ကျတော်တို့ jsx ကို မသုံးချင်ဘူးဆိုရင်တော့ React.createElement() ဆိုတဲ့ဟာလေးကို အသုံးပြုပေးရမှာပါ

ကျတော်တို့က element တခု ကိုရေးတဲ့အခါမှာ အဲလိုမျိုးရေးရတာဖြစ်ပြီးတော့ nested ဖြစ်လာတဲ့အခါမှာ ကျတော်တို့က ဒီလိုလေးရေးရမှာပါ

```
const JSX = () => {  
  return (  
    <div>  
      <h1>Hello</h1>  
    </div>  
  )  
}  
  
import React from 'react'  
const NotJSX = () => {  
  return (  
    React.createElement("div", null, React.createElement("h1", null, "Hello"))  
  )  
}
```

ကျွန်တော်တို့က nested ဖြစ်လာတာနဲ့အမျှ ကျတော်တို့က ဒီလိုလေးရေးလာရတဲ့အခါ ကျရင် နောက်ပိုင်းကျရင် readable က မရှိဖြစ်သွားနိုင်တာမလို့ ကျတော်တို့က JSX ကိုအသုံးပြုပြီးတော့ ရေးကြတာပါ

ပြီးတော့ JSX ကိုအသုံးပြုလိုက်ခြင်းအားဖြင့် ကျတော်တို့ တော်တော်လေး လေ့လာရတာ မြန်ဆန်လာပြီးတော့ တော်တော်လေးကို လေ့လာရတာ လွယ်ကူလာတာပါ ဗျ
ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ react မှာ ဘာကြောင့် jsx ကို အသုံးပြုတာလည်းဆိုတာကိုမြင်သာသွားမယ်လို့ထင်ပါတယ်ဗျာ

JSX ကို နားလည်သွားပြီးရင်တော့ ကျတော်တို့ နောက်တဆင့်ကို လေ့လာလိုက်ကြရအောင်ဗျာ

File structure ကိုလည်းလေ့လာ လိုက်ကြရအောင်

ကျတော်တို့ create လုပ်လိုက်တဲ့ folder ကို ကြည့်လိုက်တဲ့အခါမှာ ဘာကိုတွေ့ရမလဲဆိုတော့ ဗျ

React Learning

Src folder လေးကို မြင်ရပါလိမ့်မယ် အဲဒါလေးက ကျတော်တို့ရဲ့ react project တခုလုံးရဲ့ folder ပါပဲဗျာ

Nodes-modules ကိုလည်းမြင်ရပါလိမ့်မယ် အဲဒါကတော့ ကျတော်တို့ရဲ့ project မှာလိုအပ်တဲ့ file တွေ library တွေကို install လုပ်ပေးထားတဲ့ ဟာလေးပေါ့ဗျာ ပြီးရင် ကျတော်တို့ package.json ဆိုတဲ့ ဖိုင်လေးကို အရင်ကြည့်လိုက်ကြရအောင်ဗျာ အဲမှာကြည့်လိုက်ရင် dependencies မှာ install လုပ်ထားတာက react နဲ့ react-dom လေး ၂ခုကိုပဲ install လုပ်ပေးထားတာကိုမြင်ရပါလိမ့်မယ်ဗျာ ပြီးတော့သူ့ရဲ့အောက်မှာက dev-dependencies ဆိုတဲ့ object တခုကိုပါမြင်ရပါလိမ့်မယ် သူက ကျတော့ ကျတော်တို့ project ကို development ဖက်မှာ ပိုအဆင်ပြေအောင်လို့ လုပ်ထားပေးတဲ့အရာတွေပါ အဲဒါလေးကို ကြည့်လို့ပြီးသွားရင် ခုနကပြောခဲ့တဲ့ ၁ ခုတည်းသော html file ကို ကျတော်တို့ ဝင်ကြည့်လိုက်ရအောင်ဗျာ

ဒီထဲကိုဝင်ကြည့်ရင် ဘာတွေ့ရမလဲဆိုတော့ html ရဲ့ body ထဲမှာ div id="root"

လို့ပေးထားတဲ့ဟာလေးတခုကိုပဲ တွေ့ရပါလိမ့်မယ်ဗျာ အဲကောင်လေးထဲမှာ ဘာတွေ့လာမှာလည်းဆိုတော့ ကျတော်တို့ရဲ့ react project ကြီးတခုလုံးကို ဒီထဲမှာလာပြီးတော့ render ချပေးသွားမှာပါ ပြီးတော့သူ့အောက်ကို ဆက်ကြည့်လိုက်ရင် script tag ထဲမှာ main.jsx ကို connect လုပ်ထားတာကိုမြင်ရပါလိမ့်မယ်ဗျာ

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Trading</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

ပုံ - index.htm

main.jsx ကို မြင်ရတော့ ကျတော်တို့က npm run dev လို့လုပ်လိုက်တဲ့အခါမှာ ဦးဆုံး run ပေးတဲ့ file က main.jsx ဖြစ်တာကို ကျတော်တို့ သတ်မှတ်ပေးလို့ရပါတယ်ဗျာ

ကျတော်တို့ main.jsx ထဲကို ဝင်ကြည့်လိုက်တဲ့အခါမှာ

ReactDOM တွေကို အသုံးပြုပြီးတော့ ခုနက တွေ့ခဲ့တဲ့ div id="root" ထဲကို render

လုပ်ပေးထားတာကိုမြင်ရပါလိမ့်မယ်ဗျာ ဘယ်ဟာကို render လုပ်ထားတာလည်းဆိုတော့ ကျတော်တို့

ဆက်ကြည့်လိုက်တော့ app.js ဆိုတဲ့ file ကို render လုပ်ထားတာကိုမြင်ရပါလိမ့်မယ်ဗျာ

React Learning

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

ပုံ - main.jsx

ဒါဆိုရင် ကျတော်တို့ app.jsx ဆိုတဲ့ ဖိုင်ထဲကိုကျတော်တို့တချက်ကြည့်လိုက်ကြရအောင်ဗျ
ကျတော်တို့ app.jsx ဖိုင်ထဲကို ဝင်ကြည့်လိုက်ရင် ကျတော်တို့တွေ့ရမှာက browser မှာပေါ်နေတဲ့ html
တွေကို တွေ့ရပါလိမ့်မယ်ဗျ
အဲဒါကိုကြည့်လိုက်ခြင်းအားဖြင့် ဘယ်လိုမျိုးသတ်မှတ်လို့ရလည်းဆိုတော့ app.jsxဆိုတဲ့ file လေးက
ကျတော်တို့ project ရဲ့ နောက်ဆုံး file လေး လို့ပြောလို့ရမယ်ဗျ main.jsx က ကျတော့ သူက app.jsx က
ရလာတဲ့ ဒေတာကို html ပေါ်တင်ပေးတဲ့အခြေအနေဖြစ်တဲ့အတွက် ဒီထဲမှာ ဘာမှမရေးတာက
ကျတော်တို့ အတွက်ပိုကောင်းမှာပါ နောက်ပိုင်း ကျတော်တို့ redux တို့ context api တို့လိုမျိုး
ဟာတွေအတွက်ကတော့ ဒီထဲမှာ လာရေးကြပါတယ်

React Components

ကျတော်တို့ က React ကို component based architecture လို့ပြောခဲ့တော့ ကျတော်တို့
နောက်ပိုင်းကျရင် ဖန်တီးမယ့် file တွေအကုန်လုံးက လည်း component တွေပါပဲ ဗျ
ဥပမာ app.jsx ဆိုရင် app component ပေါ့ဗျ အဲလိုမျိုးခေါ်ကြတာပေါ့
ဒါဆိုရင် ဒီ component တွေကို ကျတော်တို့ ဘယ်မှာရေးကြမှာလည်းပေါ့ ကျတော်တို့က အဲဒါတွေကို src
folder ထဲမှာ ရေးကြရမှာပါ အရင်ဆုံး ကျတော်တို့ hello.jsx ကို အရင် create လုပ်လိုက်ပါမယ်ဗျ
ပြီးရင်တော့ ကျတော်တို့ ဒီထဲမှာ function လေးတစ်ခုကိုရေးလိုက်ပါမယ်
ကျတော်တို့ hello.jsx ထဲမှာ

```
function Hello () {
  return (
    <h1>Hello</h1>
  )
}
export default Hello;
```

React Learning

ဒီလိုလေးရေးလိုက်ပါမယ်

ဒီမှာ ကျတော်တို့ file name ကိုပေးတဲ့အခါမှာ capital letter နဲ့ရေးကြတာများပါတယ်

ပြီးတော့ function name ကိုလည်း capital letter နဲ့ရေးရပါမယ်

ဘာလို့အဲလိုရေးခိုင်းတာလည်းဆိုတော့ react ကနေပီးတော့ component တွေကို capital letter အနေနဲ့ သတ်မှတ်ပေးထားတာမလို့ပါ

ပြီးတော့ကျတော်တို့ export default အနေနဲ့ထုတ်လိုက်တဲ့ component တွေကို ကျတော်တို့

အသုံးပြုချင်တဲ့ component မှာပြန်ပြီးတော့ import လုပ်ပေးရပါမယ်

ကျတော်တို့က app.jsx မှာ အသုံးပြုချင်တာဖြစ်တဲ့အတွက် app.jsx မှာ import လုပ်လိုက်ပါမယ်

```
import Hello from './Hello.jsx'

function App () {
  return (
    <Hello />
  )
}

export default App
```

ဒီလိုလေးရေးလိုက်ပါမယ် ပြီးရင်တော့ ကျတော်တို့ ctrl + s နဲ့ save လုပ်ပီးတော့ browser

မှာပြန်သွားကြည့်လိုက်ရင် h1 tag နဲ့ Hello ကိုမြင်ရမှာပါ

ဒီလောက်ဆိုရင်တော့ နည်းနည်းလေး နားလည်သွားမယ်လို့ထင်ပါတယ်

Types of Components

ကျွန်တော်တို့ react ကိုလေ့လာတဲ့အခါမှာသူက component based လို့ပြောတာနဲ့အညီသူ့မှာလည်း

component types လေးတွေ ကွာတာလေးတွေရှိပါတယ်

ဘယ်လိုမျိုးကွာတာလည်းဆိုတော့ အခြေခံအားဖြင့်

၂မျိုးခွဲပါတယ်

ဘယ်လိုမျိုးလည်းဆိုတော့ *functional component* နဲ့ *class component* ဆိုပြီးတော့ပါပဲဗျ

ခုနကကျတော်ရေးပြသွားတဲ့ဟာက functional အမျိုးအစားဖြစ်ပြီးတော့ class component က ကျတော့ ကျတော်တို့ OOP base ဖြစ်တဲ့ class type နဲ့ရေးကြရပါတယ်

အရင်ကတော့ react မှာ အခုနောက်ပိုင်း သုံးတာတွေဖြစ်တဲ့ React Hook ဆိုတာမျိုးတွေ မရှိတုန်းက တော့ ကျတော်တို့က ပြောင်းလဲမှုတခုခု လုပ်ဖို့အတွက်ဆိုရင် ကျတော်တို့က OOP base ဖြစ်တဲ့ class

React Learning

component ကို အသုံးပြုကြရပါတယ် ဘာလို့ဆိုတော့ သူ့မှာ ကျတော်တို့ (state) ပြောင်းလဲမှုတွေကို လုပ်လို့ရတဲ့အတွက်ကြောင့်ပါ

သူ့ကိုအရင်တုန်းက ဘယ်လိုခေါ်ကြတာလည်းဆိုတော့ state component လို့လည်း နောက်တမျိုးခေါ်ကြပါတယ်

ပြီးတော့ functional component တွေက React 17.8 version update မထွက်ခင်အထိ တုန်းကဆိုရင် သူ့ကို က သိပ်မသုံးကြပါဘူး ဘာလို့ဆိုတော့သူ့မှာက class component တွေလိုမျိုး state တွေကို management လုပ်လို့မရလို့ပါ

အဲအတွက်ဘယ်လိုမျိုး လုပ်ကြရတာလည်းဆိုတော့ သူ့ကိုက ကျတော့ ကျွန်တော်တို့ show case ပြဖို့အတွက်ပဲ သုံးကြရတော့ သူ့ကို presentational component လို့လည်းခေါ်ကြပါတယ် နောက်ပိုင်း React ကနေပြီးတော့ React Hook တွေကို စတင်မိတ်ဆက်တဲ့အချိန်မှာ ကျတော်တို့ က functional component မှာပဲအကုန်လုံး အလုပ်လုပ်ခိုင်းစေပြီးတော့ application တွေကို လွယ်လွယ်ကူကူနဲ့ ရေးလာနိုင်တာပါ

ဒီထဲမှာကတော့ ကျတော်ကတော့ class component အကြောင်းတွေကိုပါ အတတ်နိုင်ဆုံးတော့ ကျတော်ရှင်းပြပေးသွားပါမယ်ဗျ

Functional Component

- state less in older react 17 version
- now this is more powerful and more useful
- easy to learn and clean code concept by functional concept

ပုံ - functional component

Class Component

- state full component in older react 17 version
- not efficient use in nowadays
- complex code base and OOP concept flow

ပုံ - class component

အရင်ဆုံး ကျတော်တို့ class component တခုကို အရင်တည်ဆောက်လိုက်ကြရအောင်ဗျ

```
import React from 'react'

class ClassComponent extends React.Component{

  render(){
    return (
      <h1>Hello from Class Component</h1>
    )
  }
}

export default ClassComponent
```

ကျွန်တော်တို့ ကုဒ်ကိုအရင်ဆုံး ဒီလိုလေးရေးလိုက်ကြပါမယ်ဗျာ ဒီအချိန်မှာ ကျတော်တို့ ကဘာလို့ extends လုပ်ရတာလည်းဆိုတော့ ကျတော်တို့က အရှေ့မှာ declare လုပ်လိုက်တဲ့ class component က inheritance ယူထားတာကိုပြောချင်တာပါ ဘာကိုသွားယူထားတာလည်းဆိုတော့ React.Component ကိုသွားယူလိုက်တာပါ အဲလိုသွားယူလိုက်တော့ ကျတော်တို့က ဒီ class component ထဲမှာ render ဆိုတဲ့ကောင်ကို ခေါ်သုံးလို့ရသွားပါတယ်ဗျာ အဲဒီ render ဆိုတဲ့ method က React.Component ထဲက ဟာပါ

နောက်ဆိုင်ရာလေးမှာ သုံးလို့ရမယ့် constructor တွေ super တွေကိုပါ ကျတော်တို့ခေါ်သုံးလို့ရသွားမှာပါ Render ရဲ့အထဲမှာ ကျတော်တို့က return ပြန်ချင်တဲ့ html tag လေးတွေကို ရေးပေးလိုက်ရင်ကျတော်တို့ class component ကြီးကို browser မှာ မြင်ရမှာပါ ဗျာ အဲလိုမျိုး browser မှာ မြင်ချင်တယ်ဆိုရင်တော့ ကျတော်တို့ရဲ့ component ကို app.jsx ထဲမှာ import လုပ်ပြီးတော့ သွားသုံးလိုက်ရင်ရပါတယ်ဗျာ Functional component က ကျတော့ ကျတော် ပထမဆုံးရေးပြထားပြီးပါပြီဆိုတော့ အဲအတိုင်းပြန်ရေးလိုက်ရင်ရပါတယ် ဗျာ

Props

Props အကြောင်းကို ဆက်ပြီးတော့လေ့လာကြရအောင်ဗျာ

ကျတော်တို့က React ကို လေ့လာတဲ့အခါမှာ component တွေကိုခွဲပြီးတော့ရေးလာကြတဲ့အခါမှာ component တွေ တစ်ခုနဲ့ တစ်ခု data တွေကို ပေးဖို့လိုအပ်လာပါတယ်ဗျာ

အဲလိုမျိုး ဒေတာတွေကို ပေးပြီးတော့ရေးလိုက်ခြင်းအားဖြင့် ဘာအကျိုးကျေးဇူးရသွားမှာလည်းဆိုတော့ ကျတော်တို့ code တွေကို ကြိုက်တဲ့နေရာမှာ ပြန်ပြီးတော့အသုံးပြုလို့ရသွားအောင် ရေးသွားလို့ရသွားပါတယ်ဗျာ

တစ်နည်းပြောရရင် reusable component တွေ ဖြစ်သွားတာပေါ့ဗျာ အဲလိုမျိုး ဖြစ်ဖို့အတွက် Props တွေက အရေးကြီးတဲ့ အခန်း ကဏ္ဍအနေနဲ့ ပါဝင်နေတာပါ

React Learning

Props in Functional Components

ကဲ လေ့လာလိုက်ကြရအောင် အရင်ဆုံး သူက ဘယ်လိုအလုပ်လုပ်တာလည်းကိုမလေ့လာသေးခင်မှာ
Props ဆိုတာ ဘာလည်းဆိုတာကို ကြည့်လိုက်ရအောင်

ကျတော်တို့ က props လို့ပြောနေပေမယ့် သူ့ကို property သိထားရပါမယ်

ဘာလို့လည်းဆိုတော့ ကျတော်တို့ က component တခုကိုရေးတဲ့အခါမှာ `<Component />`

အဲလိုမျိုးရေးကြတယ်မလား သူတို့ရဲ့ အထဲမှာ HTML လိုမျိုး ကြည့်လိုက်ရင် attribute

လိုမျိုးရေးပေးတဲ့ဟာကို props (properties) လို့ပြောကြပါတယ်

သူက ဘယ်လိုမျိုး အလုပ်လုပ်တာလည်းဆိုတော့ parent ကနေပြီးတော့ child ဆီကို ဒေတာ တွေ
passing လုပ်ပေးပါတယ်

တချို့တွေက react ရဲ့ data flow တွေကို ဘယ်လိုတင်စားကြတာလည်းဆိုတော့ waterfall လို့
တင်စားကြပါတယ်

ဘာလို့ဆိုတော့ react ရဲ့ data passing က *parent to child* ပဲအလုပ်လုပ်ပေးတာမလို့ပါ

Child ကနေပီးတော့ parent ဆီကို ပြန်မသွားပါဘူး ဗျ

အဲလိုမျိုး ဆိုတော့ ကျတော်တို့ က ဘယ်လိုမျိုး ရေးကြရမလဲဆိုတော့

တချက်လောက်ကြည့်လိုက်ကြရအောင်ဗျ

အရင်ဆုံး ကျတော်တို့ ကုဒ်လေးတွေကို ဒီလိုလေးရေးလိုက်ပါမယ်

```
function App () {  
  return(  
    <Child />  
  )  
}  
  
export default App
```

ပုံ - App.jsx (parent component)

```
function Child () {  
  return (  
    <h1>Hello</h1>  
  )  
}  
export default Child
```

ပုံ - Child.jsx (child component)

ဒီလိုလေးရေးပြီးတော့ ကျတော်တို့ parent ကနေပြီးတော့ ဘယ်လိုမျိုး data တွေပေးလို့ရလည်းဆိုတော့

React Learning

Html tag တွေမှာ attribute ရေးသလိုမျိုးရေးလိုက်မှာပါ
အဲဒါမျိုးမှာ ကျတော်တို့က ဘယ်လိုမျိုး ရေးမှာလည်းဆိုတော့ ကြိုက်တဲ့ naming system အနေနဲ့
ပေးလို့ရပါတယ်

ကျတော်က တော့ အလွယ်လေး ရေးပြီးတော့ပေးလိုက်ပါမယ်

name="Bate Thar" အဲလိုလေးရေးပြီးတော့ data pass လုပ်လိုက်ပါမယ် ဗျ

```
function App () {  
  return(  
    <Child name="Bate Thar" />  
  )  
}
```

ပုံ - data passing from parent

အဲလိုလေး ပေးလိုက်ပြီးတော့ ကျတော်တို့က child component မှာ ဘယ်လိုမျိုး ပြန်ပြီးတော့
အသုံးပြုရမှာလည်းဆိုတော့ ကျတော်တို့က functional component တွေ အနေနဲ့ ရေးထားတော့ function
parameter တွေလက်ခံတဲ့နေရာ ဖြစ်တဲ့ parenthesis ကနေပီးတော့ လက်ခံလိုက်လို့ရပါတယ်

ဘယ်လိုမျိုးလက်ခံလို့ရလည်းဆိုတော့

props လို့ရေးပြီးတော့လက်ခံလို့ရပါတယ်

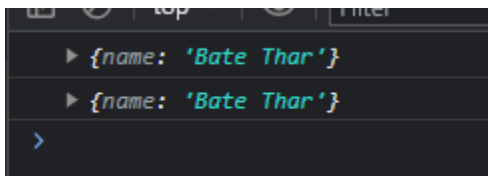
တခြား name တွေ နဲ့ လက်ခံလို့ရပါတယ် တော်တော်များများကတော့ props လို့တော့ရေးကြပါတယ်

အဲလိုမျိုးရေးပြီးတော့ ကျတော်တို့ component ထဲမှာ console.log ထုတ်ကြည့်လိုက်ရင် data pass

ပီးတော့ရောက်လာတဲ့ data က object type လိုမျိုး ရောက်လာတာကိုမြင်ရမှာပါ

```
function Child(props) {  
  console.log(props)  
  return (  
    <h1>Hello</h1>  
  )  
}  
  
export default Child
```

ပုံ - props



```
{name: 'Bate Thar'}  
{name: 'Bate Thar'}  
>
```

ပုံ - console log

React Learning

ဒီလိုလေး ကျတော်တို့က props တခုက data က ဘယ်လိုမျိုးရောက်လာတာလည်းကို မြင်သာသွားမယ်လို့ထင်ပါတယ်

သူ့ကို ကျတော်တို့ ပြန်သုံးချင်တဲ့အခါကျရင် ဘယ်လိုမျိုး အသုံးပြုရမလဲဆိုတော့ ကျတော်တို့ h1 tag ထဲမှာ

Object.key လို့ရေးပြီးတော့ခေါ်လိုက်မယ်ဆိုရင်တော့ ကျတော်တို့ အဖြေ ထွက်လာပါလိမ့်မယ် အဲမှာ ကျတော်တို့တစ်ခု သတိထားရမှာက ဒီအတိုင်း object.key ကို သွားခေါ်လိုက်ရင် ဘယ်လိုမျိုးပြမှာလည်းဆိုတော့ သူက object.key ဆိုတဲ့စာအတိုင်းပေါ်နေပါလိမ့်မယ် ပုံလေးကို တချက်ကြည့်လိုက်ရင်ပိုပြီးတော့မြင်သာသွားပါလိမ့်မယ်

```
function Child(props) {  
  console.log(props)  
  return (  
    <h1>props.name</h1>  
  )  
}  
  
export default Child
```

ပုံ - object.key

props.name

ပုံ - result

ဘာလို့အဲလိုမျိုး ထွက်တာလည်းဆိုတော့ ကျတော်တို့က jsx တွေ နဲ့ရေးတာဖြစ်တဲ့အတွက် သူ့အထဲမှာက ကျတော်တို့က javascript expression တွေ တခုခု အသုံးပြုချင်တဲ့အခါ curly bracket တွေအသုံးပြုပေးရပါတယ် မြင်သာအောင် ပြရရင် ကျတော်တို့ 1+2 ကို ဒီအတိုင်း လေးရေးကြလိုက်ရအောင်ဗျ

React Learning

```
function Child(props) {  
  console.log(props)  
  return (  
    <h1>1+2</h1>  
  )  
}  
  
export default Child
```

ပုံ - 1+2

အဲဒါကို ကျတော်တို့ result မှာသွားကြည့်ကြရအောင်ဗျ

1 + 2

ပုံ - result

ဒီလိုလေး ထွက်လာပါလိမ့်မယ် ဘာလို့ အဲလိုထွက်ရတာလည်းဆိုတော့ ကျတော်တို့က jsx tag နှစ်ခုကြားများက သူတို့က template literals (back tips) အဲလိုလေး တွေ့လက်ခံပေးတာမလို့ပါ တစ်နည်းပြောရရင် string data ကိုပဲ သူတို့ကလက်ခံပေးထားတာပါ အဲဒါကို ကျတော်တို့က javascript expression (calculation / execution) တွေ အသုံးပြုလာချင်တဲ့အခါကျရင် curly bracket (တွန့်ကွင်း) လေးကို အသုံးပြုပေးရတာပါ

```
function Child(props) {  
  console.log(props)  
  return (  
    <h1>{1+2}</h1>  
  )  
}  
  
export default Child
```

ပုံ - use curly bracket (js expression)

3

ပုံ - browser result

အဲဒီတော့ ကျတော်တို့က props ကိုပြန်ခေါ်ချင်ရင် ဘယ်လိုမျိုးခေါ်ရမှာလည်းဆိုတာကို မြင်သာမယ်လို့ထင်ပါတယ်

အဲဒီတော့ကျတော်တို့ ကုဒ်ကို နည်းနည်းလေး ပြင်လိုက်ပါမယ်

```
function Child(props) {  
  console.log(props)  
  return (  
    <h1>{props.name}</h1>  
  )  
}  
  
export default Child
```

ပုံ - use curly bracket

Bate Thar

ပုံ - browser result

ကျတော်တို့ props တွေကို ဒီလောက်ဆိုရင် တော်တော်လေး နားလည်သွားလောက်မယ်လို့ထင်ပါတယ်
ကျတော်တို့က props တွေကိုအသုံးပြုပြီးတော့ component တွေဆီကို data တွေ passing လုပ်ရင်းနဲ့
reusable component တွေ ဖန်တီးပြီးတော့ ကျတော်တို့ရဲ့အချိန်တွေကို
သက်သာအောင်လုပ်လို့ရသွားပါတယ်
props ကနေပြီးတော့ ကျွန်တော်တို့ data တွေကို string အနေနဲ့ပဲ ပို့လို့ရတာ မဟုတ်ပါဘူး js data type
တွေအကုန်လုံးကို ပို့ပေးလို့ရပါတယ်

Object

Array

Function

Number တွေ အစရှိတဲ့ data တွေကိုအကုန်လုံးကို props ကနေပြီးတော့ data passing လုပ်ပြီးတော့
component တွေ ကို အသုံးချလို့ရပါတယ် ဗျ
ကျတော်တို့ Props တွေ က Object type ဖြစ်တော့ ကျတော်တို့ က props တွေကိုလည်း destructuring
လုပ်ပြီးတော့ အသုံးချလို့ရပါသေးတယ်ဗျ
ဘယ်လိုမျိုး လည်းဆိုတော့ ကျတော်တို့က props အစား object destructuring
ရေးပြီးတော့အသုံးချလို့ရပါတယ်

```
function Child({name}) {
  return (
    <h1>{name}</h1>
  )
}

export default Child
```

ပုံ - props destructuring

React Learning

ဒီလိုလေးရေးမယ်ဆိုရင်လည်းရပါတယ်ဗျ

Props in Class Components

ကျတော်က ဒီစာအုပ်မှာ class component တွေနဲ့ပါ တွဲပီးတော့ ရေးသွားဖို့ တွေးထားတာဆိုတော့ props တွေကိုလည်း class component တွေထဲမှာ ဘယ်လိုမျိုး အသုံးပြုရမှာလည်းဆိုတာကို ပါ တခါတည်းလေ့လာသွားကြရအောင်ဗျ အရင်ဆုံး ကျတော်တို့ class component တစ်ခုကို ဖန်တီးကြရအောင်ဗျ

အရင်ဆုံး ကျတော်တို့ code တွေကို ဒီလိုလေးရေးလိုက်ပါမယ်

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

```
class ClassComponent extends Component {
```

```
  constructor() {
```

```
    super()
```

```
  }
```

```
  render(){
```

```
    return(
```

```
      < h1 > Class Component < /h1 >
```

```
    )
```

```
  }
```

```
}
```

```
export default ClassComponent
```

အဲလိုမျိုးရေးပြီးတော့ ကျတော်တို့ ကုဒ်တွေကိုတချက် လေ့လာကြည့်လိုက်ရအောင်ဗျ

ဒီထဲမှာ ကျတော်တို့က constructor ကို ထည့်ရေးထားတာကို မြင်ရပါလိမ့်မယ်ဗျ

ကျတော်တို့ class component မှာက ကျတော့ functional လိုမျိုး props ကိုလက်ခံဖို့အတွက်

Parenthesis မပါဘူး ဗျ အဲအတွက် ကျတော်တို့က constructor ကနေပြီးတော့ props

ကိုလက်ခံပေးရမှာပါ

အဲလို လက်ခံလိုက်တော့မှ ကျတော်တို့ render လုပ်တဲ့အခါ props ရဲ့ data တွေကို

အသုံးပြုလို့ရသွားတာပါ

```
import {Component} from 'react'
class Child extends Component {
  constructor(props) {
    super(props)
  }
  render(){
    return(
      <h1> Class Component </h1>
    )
  }
}
export default Child
```

ပုံ - props in class component

အဲလိုမျိုးရေးပြီးတော့ ကျတော်တို့က passing လုပ်လာတဲ့ data ကို ပြန်အသုံးပြုချင်တဲ့အခါကျတော့ render method ထဲမှာ သွားပြီးတော့အသုံးပြုပေးရပါတယ်

```
render(){
  console.log(this.props)
  return(
    <h1> Class Component </h1>
  )
}
export default Child
```

ပုံ - console log props

အဲမှာ ကြည့်လိုက်တဲ့အခါမှာ ကျတော်တို့ သတိထားမိမှာက this keyword

ကြီးပါနေတာကိုပါမြင်ရပါလိမ့်မယ်

ဘာလို့လည်းဆိုတော့ အဲဒီ props က class Child ထဲမှာ ရှိနေတာမလို့ သူ့ကို ကျတော်တို့က

ပြန်ညွှန်ပေးရပါတယ်ဗျ

အဲလိုမျိုး ရေးပြီးတော့ ကျတော်တို့က browser မှာပေါ်စေချင်ရင် ဒီလိုရေးပေးရပါမယ်ဗျ

```
import {Component} from 'react'
class Child extends Component {
  constructor(props) {
    super(props)
  }
  render(){
    console.log(this.props)
    return(
      <h1>{this.props.name}</h1>
    )
  }
}
export default Child
```

ပုံ - use props in render



Bate Thar

ပုံ - result in browser

အဲလောက်ဆိုရင်တော့ ကျတော်တို့ class component ထဲမှာလည်း props တွေကို ဘယ်လိုမျိုး ယူသုံးလို့ရတာလည်း ကို မြင်သာမယ်ထင်ပါတယ်
သူ့ကို လည်း object destructuring လုပ်ချင်ရင်လည်းလုပ်လို့ရပါတယ် ဗျ
ဘယ်လိုမျိုးဖြစ်သွားမလဲဆိုတော့

```
render(){
  console.log(this.props)
  const {name} = this.props
  return(
    <h1>{name}</h1>
  )
}
```

ပုံ - props destructuring

React Learning

အဲလိုလေးရေးပြီးတော့အသုံးပြုလို့ရသွားပါပီဗျ
ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ props တွေကို ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်း
ဆိုတာကိုမြင်သာသွားမယ်လို့ထင်ပါတယ်ဗျ

Children Props

ကျတော်တို့ ဆက်လေ့လာသွားကြမှာက ဘာလည်းဆိုတော့ ကျတော်တို့ props တွေထဲမှာ attribute
အနေနဲ့ပေးလို့ရသလို html tag (component) တခုလုံးကိုလည်းပေးလို့ရပါတယ်
တချို့အခြေအနေတွေမှာ ကျတော်တို့က props တွေကို children အနေနဲ့ passing လုပ်ပြီးတော့
ရေးကြတာမျိုးတွေရှိပါတယ်
နောက်ပိုင်းကျတော် ရှင်းပြတဲ့အခါကျရင် မြင်လာပါလိမ့်မယ်
သူ့ကို က ကျတော့ ကျတော်တို့က ဘယ်လိုမျိုး passing ပေးကြတာလည်းဆိုတော့ enclosing tag ထဲမှာ
ရေးပြီးတော့ data passing လုပ်ကြပါတယ်
ဘယ်လိုမျိုးလည်းဆိုတော့ အရင်ဆုံး ကျတော်တို့ component တခုကိုတည်ဆောက်လိုက်ပါမယ်

```
import React from 'react'

function ChildProps(props) {
  return (
    <div>
      {props.children}
    </div>
  )
}

export default ChildProps
```

ကျတော်တို့ props ကို လက်ခံပြီးတော့ အဲ props ရဲ့ children အနေနဲ့ ကျတော်တို့က ပြန်ပြီးတော့ render
လုပ်ထားလိုက်ပါတယ်
သူ့ကို ကျတော်တို့က app.jsx (app component) ထဲမှာ ဒီလိုလေး သွားရေးလိုက်ပါမယ်
ဒီနေရာမှာ တခုသတိထားမှာက ကျတော်တို့အရင်ကလိုမျိုး attribute အနေနဲ့
သွားတာမျိုးမဟုတ်တော့ဘဲနဲ့
သူ့ရဲ့ အထဲမှာ html tag ရဲ့အထဲမှာစာရေးသလိုမျိုး ရေးပြီးတော့ သွားပါတယ်

```
<ChildProps>
  Hello
  <p>Hello</p>
</ChildProps>
```

React Learning

ဒီလိုလေးရေးလိုက်ပါမယ် အဲဒါကို ကျတော်တို့ browser မှာသွားကြည့်တဲ့အခါကျရင် ဒီလိုလေးတွေ့ရပါမယ်

Hello

Hello

ဒီလောက်ဆိုရင်တော့ props တွေအကြောင်းကို တော်တော်လေး နားလည်သွားမယ်လို့ထင်ပါတယ်

States

States in Class Component

ကျတော်တို့ ခုနက props ကိုလေ့လာပြီးပြီဆိုတော့ ကျတော်တို့ အမြဲလိုလို ကြားကြားနေရမယ့်အရာတခုကို လေ့လာလိုက်ကြရအောင်ဗျ အဲဒါကဘာလည်းဆိုတော့ ခေါင်းစဉ်မှာပြောထားတဲ့အတိုင်းပါပဲ State တွေ ပါပဲ

သူက react application တခုမှာ တော်တော်လေး အရေးကြီးတဲ့ အခန်း ကဏ္ဍ တခုအနေနဲ့ပါတယ်ဗျ ဘာလို့ဆို ကျတော်တို့က state တွေကို အသုံးပြုပြီးတော့ react application တစ်ခုလုံးကို စိတ်ကြိုက် လိုသလို လုပ်နိုင် လို့ပါ သူ့မှာက ကျတော်တို့ က state တွေကို မြင်သာအောင်ပြောရင် get set လုပ်တာပါပဲဗျ အဲအတွက် သူက value နဲ့ method ဆိုပြီးတော့ ထုတ်ပြီးတော့ အလုပ်လုပ်စေတာပါ အဲဒါဆို ကျတော်တို့အရင်ဆုံးလေ့လာလိုက်ကြရအောင်ဗျ အရင်ဆုံး ကျတော်က state ကို class component မှာ အသုံးပြုပြပါမယ် code လေးတွေကို တချက်ရေးလိုက်ကြရအောင်ဗျ

```
import {Component} from 'react'
class Child extends Component {
  constructor(props) {
    super(props)
    this.state= {
      count : 0
    }
  }
  render(){
    return(
      <h1>{this.state.count}</h1>
    )
  }
}
export default Child
```

ပုံ - state in class component

React Learning

ကျတော်တို့ ဒီမှာ သတိပြုရမှာက ကျတော်တို့က class component ထဲမှာဆိုရင် state တစ်ခုက object type အနေနဲ့ရှိနေတာပါပဲ

အဲလိုမျိုးရေးပြီးတော့ ကျတော်တို့က state ကို browser ပေါ်မှာ သူက render ချပြီးတော့ အလုပ်လုပ် ခိုင်းလို့ရပါပြီ ဗျ

ပြီးရင်တော့ ကျတော်တို့ နောက်တဆင့် ထပ်ပြီးတော့ တက်လိုက်ရအောင်ဗျ အဲဒါက ဘာလည်းဆိုတော့ ကျတော်တို့ state ကို changes လေး လုပ်ကြရအောင်ဗျ အဲလိုမျိုး changes လုပ်ဖို့အတွက်ဆိုရင် သူက ဘယ်လိုမျိုး ရေးရမလဲဆိုတော့ ကျတော်တို့က button လေး နိပ်လိုက်တဲ့အချိန်မှာ state ထဲက count ကို ၁ တိုးသွားအောင်ရေးကြမှာပေါ့ အဲလိုဆိုရင်ကျတော်တို့ က h1 tag ရဲ့ အောက်မှာ button ဆိုတဲ့ tag လေးကို ရေးလိုက်ကြရအောင်ဗျ

```
import {Component} from 'react'
class Child extends Component {
  constructor(props) {
    super(props)
    this.state= {
      count : 0
    }
  }
  render(){
    return(
      <h1>{this.state.count}</h1>
      <button>Add One</button>
    )
  }
}
export default Child
```

ပုံ - add btn added

ကျတော်တို့ အဲလိုရေးလိုက်တဲ့အခါမှာ error stage လေး ဖြစ်သွားတာကိုမြင်ရလိမ့်မယ်ဗျ

ဘာလို့အဲလိုဖြစ်ရတာလည်းဆိုတာကို ကျတော်တို့ browser ကို

တချက်လောက်သွားကြည့်လိုက်ရအောင်ဗျ

သွားကြည့်တော့ browser မှာ ဘာတွေတွေ့ရမလဲဆိုတော့

```
[plugin:vite:react-babel] C:\Users\USER PC\Desktop\react learning\react-
lesson1\src\Child.jsx: Adjacent JSX elements must be wrapped in an enclosing tag. Did you
want a JSX fragment <>...</>? (21:2)
  24 | }
```

ပုံ - error

ဒီလိုလေးတွေ့ရပါလိမ့်မယ်

React Learning

အဲမှာ သူက ရေးထားတာက ဘာလည်းဆိုတော့ JSX element တစ်ခု must be wrapped in enclosing tag တဲ့ ပြောချင်တာက ကျတော်တို့ return function ထဲမှာ ပြန်ရမယ့် html တွေက tag

တခုတည်းဖြစ်နေရမှာပါတဲ့

သူက html tag ၂ခု ဖြစ်သွားတာဖြစ်တဲ့အတွက် သူက အဲဒါကို နားမလည်နိုင်တော့ဘူး ဟု

ပြောရရင်သူက node (html tag) တစ်ခုတည်းကိုပဲ သိတယ်ပေါ့ဗျ

အဲလိုမျိုး ဆိုတော့ ကျတော်တို့က ဘယ်လိုမျိုးရေးမှာလည်းပေါ့

div tag နဲ့ အုပ်မယ်ဆိုလည်းရပါတယ်

ပြီးတော့ နောက်တစ်ခုရှိပါသေးတယ် ကျတော်တို့က react မှာ သုံးနေတာဖြစ်တဲ့အတွက် fragment tag (

<> </>) အဲလိုမျိုးလေးနဲ့လည်းရပါတယ် ဟု

div tag နဲ့ fragment tag ၂ခု က ဘာကွာလည်းပေါ့ div က ကျတော့ ကျတော်တို့ browser ပေါ်မှာ render

ချတဲ့အခါမှာ div တွေ ထည့်ပေးသွားမှာပါ fragment က ကျတော့ ဘာမှမပါသွားပါဘူး

အခုနောက်ပိုင်းက fragment ကိုပဲ တော်တော်အသုံးများလာကြပါတယ်ဗျ

ကဲ ဒါဆိုရင်ကျတော်တို့ ခုနက state ကို changes လေး ဆက်လုပ်လိုက်ကြရအောင်ဗျ

အဲလိုမျိုးရေးပြီး သွားရင်ကျတော်တို့က button ထဲမှာ event ကို handle လုပ်ကြရအောင်ဗျ

Event Handling (Changing State)

ကျတော်တို့ ကုဒ်တွေကို အရင်ရေးလိုက်ကြရအောင်ဗျ အရင်ဆုံး ကျတော်တို့ button ထဲမှာ event

လေးတခုကိုရေးကြရအောင်ဗျ

အဲဒါက ဘာလည်းဆိုတော့ click ဆိုတဲ့ဟာလေးဗျ

ကျတော်တို့က html မဟုတ်တဲ့အတွက် ရေးရမယ့်ဟာလေးတွေက နည်းနည်းလေးတွေပြောင်းပါတယ်

ဘယ်လိုမျိုးလည်းဆိုတော့ onClick ဆိုပီးတော့ရေးရမှာပါ

```
return(  
  <>  
    <h1>{this.state.count}</h1>  
    <button onClick={this.clickHandler}>Add One</button>  
  </>  
)
```

ပုံ - add an event

အဲလိုလေးရေးပြီးတော့ ကျတော်တို့က clickHandler function တစ်ခုကို create လုပ်ကြရအောင်ဗျ

render ရဲ့ အထက်မှာ create လုပ်လိုက်ပါမယ်

ပြီးတော့ ကျတော်တို့က state ကို စတင်ပီးတော့ ပြောင်းလဲဖို့ အတွက် သူ့ကိုပြောင်းလဲဖို့အတွက် method

လေးတစ်ခုရှိပါတယ် အဲဒါက ဘာလည်းဆိုတော့ setState ဆိုတဲ့ ဟာလေးပါ

အဲဒါကိုအသုံးပြုပြီးတော့ ကျတော်တို့ state ကို ပြောင်းလဲပေးရမှာပါ

React Learning

ဟာ အဲဒါဆို ဒီလိုရေးရင်ကောမရဘူးလား ဘာလို့ setStateကို အသုံးပြုမှရတာလည်း လို့ မေးစရာရှိလာပါလိမ့်မယ်

```
}  
clickHandler = () => {  
  this.state = {  
    count : this.state.count + 1  
  }  
  console.log(this.state.count)  
}
```

ပုံ - Question for example

အဲလိုမျိုး မေးခွန်းအတွက် ရလားဆိုတော့ ရပါတယ်

ဒါပေမယ့် သူက state ကိုပဲ ပြောင်းလဲပေးတာမျှ

ကျတော်တို့က ဘာကိုပါ ထပ်ပြီးတော့အလုပ်လုပ်စေချင်တာလည်းဆိုတော့ ကျတော်တို့က

ပြောင်းလဲလိုက်တဲ့ state ကို browser ပေါ်မှာ ပြန်မြင်စေချင်တာမလို့ setState ကိုအသုံးပြုတာပါ

setState ဆိုတာက ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတော့ ကျတော်တို့ react component

တခုမှာရှိတဲ့ state က ပြောင်းလဲမှု တခုခုဖြစ်သွားရင် အဲကောင်ကို သူက rerender လုပ်ပေးပါတယ်

အဲလိုမျိုး ပြန်လုပ်တဲ့အတွက် ကျတော်တို့ ပြောင်းလဲလိုက်တဲ့ ပြောင်းလဲမှုတွေကို browser ပေါ်မှာ ပါ

ပြောင်းလဲပြစေချင်တဲ့အတွက် ကျတော်တို့ က setState ကိုအသုံးပြုပေးရတာပါ

ကဲ ဒါဆိုရင်ကျတော်တို့ ကုဒ်တွေကို ဆက်ရေးလိုက်ကြရအောင်ဗျ

```
}  
clickHandler = () => {  
  this.setState({  
    count : this.state.count + 1  
  })  
}  
render(){
```

ပုံ - using setState method

အဲလိုမျိုး အသုံးပြုပြီးတော့ ကျတော်တို့ browser မှာ သွားပြီးတော့ စမ်းကြည့်မယ်ဆိုရင် browser

မှာပေါ်တဲ့ ဒေတာပါ လိုက်ပြောင်းလဲနေတာကိုမြင်ရပါလိမ့်မယ်ဗျ

ဒီလောက်ဆိုရင်ကျတော်တို့ state နဲ့ setState အကြောင်းကို တော်တော်လေး

နားလည်သွားမယ်လို့ထင်ပါတယ်

Asynchronous setState

ကျတော်တို့ state တွေ setState တွေ နဲ့အလုပ်လုပ်ပုံတွေကို နားလည်သွားကြမယ်လို့မျှော်လင့်ပါတယ်

နောက်ထပ်ပြီးတော့ကျတော်တို့ ဆက်ပြီးတော့လေ့လာကြရအောင်ဗျ ဘယ်လိုမျိုးလည်းဆိုတော့

asynchronous setState ပါ

React Learning

ကျတော်တို့ `setState` တခုက `state` ကို `update (changes)` လုပ်တာလို့ကျတော်တို့ သိခဲ့တယ်မလား
အဲဒါကို ကျတော်တို့က နည်းနည်းလေး တမျိုး လုပ်ကြည့်ကြရအောင်ဗျ
ခုနက ရေးခဲ့တဲ့ `increment` ဆိုတဲ့ `function` လေးကို ကျတော်တို့နောက်တမျိုး ထပ်ပြီးတော့
ရေးကြရအောင်ဗျ
ဘယ်လိုမျိုးလည်းဆိုတော့ ကျတော်တို့က တခါ တိုးတိုင်းမှာ ၅ ပေါင်းသွားစေချင်တာမျိုးပါ
ပုံမှန်ဆိုရင် ကျတော်တို့ `count + 5` လို့ရေးလိုက်ရင် ရပီ ဆိုပေမယ့် ဒီတခါ ကျတော်က
ဘယ်လိုမျိုးလုပ်မှာလည်းဆိုတော့ `for loop` နဲ့ပေါင်းမှာပါ ကဲ ရေးလိုက်ကြရအောင်ဗျ

```
addFiveHandler = () =>{
  for(let i =0; i<5; i++){
    this.setState({
      count: this.state.count + 1
    })
  }
}

render(){
  return(
    <>
      <h1>{this.state.count}</h1>
      <button onClick={this.clickHandler}>Add One</button>
      <button onClick={this.addFiveHandler}>Add five</button>
    </>
  )
}
```

ပုံ - adding five

ကျတော်တို့အဲလိုမျိုးရေးပြီးတော့ browser မှာသွားစစ်ကြည့်ရင် ကျတော်တို့က
ဘယ်လိုမျိုးတွေ့ရမလဲဆိုတော့
၅ မတိုးသွားဘဲနဲ့ ၁ ပဲတိုးသွားတာ ကိုတွေ့ရပါလိမ့်မယ်ဗျ
အဲဒါကို ကျတော်တို့က `for loop` က ငါးကြိမ်မပတ်ဘူးလား ဆိုပြီးတော့ မေးစရာရှိလာတာပေါ့ဗျ
ဟုတ်မဟုတ်ကို ကျတော်တို့ `for loop` မှာ `console` ထုတ်ကြည့်လိုက်ကြရအောင်

```
,
addFiveHandler = () =>{
  for(let i =0; i<5; i++){
    console.log("adding" , i)
    this.setState({
```

ပုံ - console log

ဒီလိုထုတ်ပြီး ကြည့်လိုက်ရင် ကျတော်တို့က 5 ကြိမ်အလုပ်လုပ်သွားတာကိုမြင်ရလိမ့်မယ်ဗျ
ဒါဆို ဘာလို့ `setState` ကဘာလို့အလုပ်မလုပ်ရတာလည်း ပေါ့

React Learning

တကယ်က အလုပ်မလုပ်တာ မဟုတ်ပါဘူး သူက asynchronous ဖြစ်နေလို့ပါ

Asynchronous အကြောင်းကို မသိသေးရင် ဒီဟာကို သွားဖတ်ပါ [Link](#)

Asynchronous ဖြစ်တော့ သူက state ကို သွားယူတဲ့အချိန်မှာ အရင်က ရှိပြီးသား initial state ရဲ့ count ကို 0 ဖြစ်တော့ အဲဒါကိုပဲ အစကတည်းက ငါးကြိမ်လုံး သွားယူပြီးတော့ အဲဒါကိုပဲ update လုပ်နေတာဖြစ်တဲ့အတွက်

သူက 1 ပဲ တိုးတိုးသွားတာပါ အဲလိုမျိုးဖြစ်နေတာကို ကျတော်တို့က ဘယ်လိုမျိုး ပြန်ပြီးတော့ fix မလဲပေါ့

ဟုတ်ပြီး အဲဒါကို ကျတော်တို့က function တခုနဲ့ ပြန်ပြီးတော့ သူ့ရဲ့အရင်က previous state ဆို ပြီးတော့ ပြန်ယူသုံးလို့ရပါတယ်ဗျ

အဲဒါကို ကျတော်တို့ ပြင်မယ်ဆိုရင် ဒီလိုလေးဖြစ်သွားလိမ့်မယ်

```
addFiveHandler = () => {  
  for(let i = 0; i < 5; i++) {  
    console.log("adding", i)  
    this.setState((prev) => {  
      return {  
        count: prev.count + 1  
      }  
    })  
  }  
}
```

ပုံ - adding five fixing

Function ရဲ့ parameter ကနေပြီးတော့ ကျတော်တို့က state ကို ပြန်ပြီးတော့လက်ခံလိုက်တဲ့အခါမှာ

သူက သူ့ရဲ့အရင်တခါက update လုပ်ခဲ့တဲ့ state ကိုပဲ လာထည့်ပီးတော့

အလုပ်လုပ်ပေးတာဖြစ်တဲ့အတွက် ကျတော်တို့က 5 တိုးသွားတာကို မြင်ရတာပါ

အဲလိုလေး ပြင်သွားပီးဆိုရင်ကျတော်တို့ပြန်ပီးတော့ သွားစမ်းကြည့်မယ်ဆိုရင် addFiveHandler လေးက

၅ တိုးသွားတာကိုမြင်ရပါလိမ့်မယ်အဲလောက်ဆိုရင် setState က

ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတာ ကို မြင်သွားမယ်လို့ထင်ပါတယ်

States in Functional Components

ဒီတစ်ခါ ကျတော်တို့ ဆက်ပြီးလေ့လာကြမှာက Functional component တွေထဲမှာ state

တစ်ခုကဘယ်လိုမျိုး အလုပ်လုပ်တာလည်းဆိုပီးတော့ပါ အရင်က ကျတော်တို့က functional

component တွေထဲမှာ state တွေကို management လုပ်လို့မရဘူးဗျ react 17.8 မှာ

စတင်မိတ်ဆက်လိုက်တဲ့ React Hook တွေကြောင့် နောက်ပိုင်းကျရင် ကျတော်တို့ state တွေကို

functional မှာ အသုံးပြုလို့ရလာတယ်ဗျ

ကဲ စလိုက်ကြရအောင်ဗျ

React Learning

အရင်ဆုံး ကျတော်တို့ functional component လေးတစ်ခုကိုရေးလိုက်ပါမယ်

```
import React from 'react'

function Functional() {
  return (
    <div>Functional</div>
  )
}

export default Functional
```

ပုံ - functional component

ပြီးသွားရင်တော့ ကျတော်တို့က state ကို အသုံးပြုဖို့အတွက် react hook တစ်ခုဖြစ်တဲ့ useState hook ကို ကျတော်တို့ import လုပ်ရပါမယ် ဘယ်ကနေပြီးတော့ import လုပ်ရမလဲဆိုတော့ react ကနေပြီးတော့ import လုပ်ပေးရမှာပါ

```
import {useState} from 'react'
```

ပုံ - importing useState

အဲလိုမျိုး ကျတော်တို့ hook ကို import လုပ်ပြီးသွားရင် သူ့ကို အသုံးပြုရပါမယ် အသုံးမပြုသေးခင်မှာ ကျတော်တို့က useState hook ဆိုတာက ဘာလည်းဆိုတာကို တချက်လေ့လာကြည့်ကြရအောင်ဗျ

သူ့ကိုကျတော်တို့ ရေးတဲ့အခါကျရင် method လိုလေးမျိုး ပြန်ခေါ်ပေးရပါတယ်

useState() အဲလိုလေး ပြန်အသုံးချရပါတယ်

သူကနေပြီးတော့ ကျတော်တို့ကို state နဲ့ setState method ဆိုတဲ့ value ၂ ခုကို array အနေနဲ့ return ပြန်ပေးပါတယ် ဗျ အဲဒါကို ကျတော်တို့က ဘယ်လိုမျိုး အသုံးပြုရမလဲဆိုတော့

```
function Functional() {
  const [count, setCount] = useState(0)
```

ပုံ - useState

အဲလိုမျိုး အသုံးပြုတဲ့အခါမှာ ကျတော်ရေးထားတာက useState ရဲ့ အနောက်မှာပါတဲ့ parenthesis ထဲမှာ ကျတော်တို့ 0 လေးကို ထည့်ပေးထားတာကို တွေ့ရမယ်ဗျ အဲဒါက ကျတော်တို့က initial value လေး ကိုသတ်မှတ်ပေးလိုက်တာပါ

သူ့အနောက်မှာပါတဲ့ count နဲ့ setCount ဆိုတာက ကျတော်တို့ state နဲ့ setState တွေပါပဲဗျ

တခုထူးဆန်းတာက ကျတော်တို့က class component ထဲမှာဆိုရင် state က object type အနေနဲ့

သတ်မှတ်ပေးရတာဖြစ်ပီးတော့ functional component ထဲမှာဆိုရင် သူက ကြိုက်တဲ့ data type အနေနဲ့

သတ်မှတ်ပေးလို့ရသွားတာပါပဲ ဗျ

အဲဒါဆိုရင်ကျတော်တို့က counter လေးကိုအရင်ဆုံးရေးလိုက်ကြရအောင်ဗျ

```
import {useState} from 'react'

function Functional() {
  const [count, setCount] = useState(0)
  function addHandler () {
    setCount(count+1)
  }
  return (
    <>
    <h1>{count}</h1>
    <button onClick={addHandler}>Add 1</button>
    </>
  )
}

export default Functional
```

ပုံ - counter

ကျတော်တို့က class component ထဲမှာသုံးသွားတဲ့အတိုင်းပါပဲ count value ကို ကျတော်တို့ တိုက်ရိုက် ရေးပြီးတော့ addHandler ဆိုတဲ့ function ထဲမှာ setCount ဆိုတဲ့ state change method လေးကို အသုံး ပြု သွားတာပါ

ကုန်ကြည့်လိုက်ရင် ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတာကိုမြင်သာမယ်ထင်ပါတယ် တစ်ခုတော့မမေ့နဲ့ပေါ့ဗျာ ကျတော်တို့က component တွေကို ခွဲပြီးရေး ခဲ့ရင် အဲဒါကို ကျတော်တို့ app component ထဲမှာ import လုပ်ပြီးတော့ အဲဒါကိုပြန်အသုံးပြုဖို့ မမေ့ကြပါနဲ့ဗျ

This keyword binding in Class Component

အပေါ်မှာ ကျတော်တို့ရေးခဲ့တဲ့ functional နဲ့ class component ကို ပြန်ကြည့်လိုက်ရင် function တွေရေး တဲ့အခါမှာ ၂ မျိုးကွဲနေတာကို မြင်ရလိမ့်မယ်ဗျ

ဟုတ်ပါတယ် ကျတော် တမင် ခွဲရေးခဲ့တာပါ ဘာလို့လည်းဆိုတော့ this keyword ကြောင့်လေးပါ အဲဒါကို သိရအောင်လို့ ကျတော် ဒီလိုခွဲရေးပေးခဲ့တာပါ

တကယ်လို့ ကျတော်တို့က class component ထဲမှာ regular function expression ကို အသုံးပြုချင်တဲ့ အခါကျရင် ကျတော်တို့က this keyword ကို class က ပိုင်တာဖြစ်ကြောင်းသိစေဖို့အတွက်

ကျတော်တို့က binding လုပ်ပေးရပါမယ် ဘာလို့ ဒီလိုလုပ်ရတာလည်းဆိုတော့ class ထဲမှာရှိတဲ့ state / setState ကိုအသုံး ပြုချင်လို့ပါ မဟုတ်ရင် သူက undefined ဆိုပြီးတော့ error လာပြပါလိမ့်မယ်

အရင်ဆုံး ကျတော်တို့ function တခုကိုရေးလိုက်ကြရအောင်ဗျ

ကျတော်တို့ class component ရဲ့ clickHandler လေးကို ဒီလိုလေး ပြင်လိုက်ပါမယ်

React Learning

```
clickHandler () {  
  this.setState({  
    count : this.state.count + 1  
  })  
}
```

ပုံ - change clickHandler

အဲလိုလေးပြင်ပြီးတော့ ကျတော်တို့က browser မှာ သွားပြီးတော့ စမ်းကြည့်လိုက်တဲ့အခါမှာ error တက်လာတာကို မြင်ရလိမ့်မယ်ဗျ

```
Uncaught TypeError: Cannot read properties of undefined (reading 'setState')  
    at clickHandler (Child.jsx:19:8)  
    at HTMLUnknownElement.callCallback2 (react-dom.development.js:4164:14)  
    at Object.invokeGuardedCallbackDev (react-dom.development.js:4213:16)  
    at invokeGuardedCallback (react-dom.development.js:4277:31)  
    at invokeGuardedCallbackAndCatchFirstError (react-dom.development.js:4291:25)  
    at executeDispatch (react-dom.development.js:9041:3)  
    at processDispatchQueueItemsInOrder (react-dom.development.js:9073:7)
```

ပုံ - error in console

အဲလိုမျိုး က ဘာကြောင့်တက်ရတာလည်းဆိုတော့ this ဆိုတဲ့ keyword က regular function ထဲမှာဆိုရင်

သူက အဲဒီ function ကိုပဲ ပြန်ညွှန်းပေးတာကြောင့်မလို့ပါ အဲဒါကြောင့် ကျတော်က arrow function လေးကို အသုံးပြုပြီးတော့ရေးပြသွားတာပါ

သူ့ကို class က နေပီးတော့ လာတဲ့ state / setState တို့ကို အသုံးပြုချင်တဲ့အတွက် ကျတော်တို့က ဘယ်လိုမျိုးရေးရမှာလည်းဆိုတော့

constructor function ထဲမှာ ကျတော်တို့က binding လုပ်ပေးရမှာပါ

ဘယ်လိုမျိုးရေးရမှာလည်းဆိုတော့

```
constructor(props) {  
  super(props)  
  this.state = {  
    count : 0  
  }  
  this.clickHandler = this.clickHandler.bind(this)  
}
```

ပုံ - binding this

ဒီလိုလေးရေးပြီးတော့ ကျတော်တို့က binding လုပ်ပေးရမှာပါ

ဒါဆိုရင် ကျတော်တို့ရဲ့ clickHandler ဆိုတဲ့ function လေးက အရင်အတိုင်းအလုပ်လုပ်သွား ပါလိမ့်မယ်ဗျ

ဒီလောက်ဆိုရင်တော့ state တွေရဲ့ကြောင်းကို တော်တော်လေး နားလည်သွားလိမ့်မယ် လို့ထင်ပါတယ်

React Learning

အပေါ်က ကုဒ်တွေကိုကြည့်ခြင်းအားဖြင့် ကျတော်တို့က ဘာကြောင့်နောက်ပိုင်း ကျရင် functional ကိုပဲ အသုံးများလာတာလည်းဆိုတာကိုမြင်သာမယ်ထင်ကြပါတယ်

Events Handling

ကျတော်တို့ အခု react ရဲ့ state တွေကို တော်တော်လေး နားလည်သွားကြမယ်လို့ထင်မိပါတယ် အခုလက်ရှိကတော့ ကျတော်တို့က react ရဲ့ state ကို အသုံးပြုမယ့် event handle တွေအကြောင်းကို ဆက်လေ့လာသွားကြရအောင်ဗျ ဘယ်လိုမျိုးလည်းဆိုတော့ ကျတော်တို့က vanilla js မှာဆိုရင် event တွေကို ကျတော်တို့ addEventListener တွေနဲ့ စောင့်ကြတယ်မလား

အဲလိုမျိုးပါပဲ ကျတော်တို့မှာက JSX ရဲ့အထဲမှာ အသုံးပြုပြီးတော့ event တွေကို handle လုပ်ရမယ့် method (attribute) လေးတွေ ရှိကြပါတယ် ဘာလို့ attribute လို့ပြောတာလည်းဆိုတော့ သူက html tag မှာလိုမျိုး မြင် သာအောင်ပြောပြလိုက်တာပဲ

တော်တော်အသုံးများတဲ့ event တွေက တော့

onClick (ကျတော်တို့ click နှိပ်တာကိုစောင့်ပေးတဲ့ event အများအားဖြင့် button တွေမှာ အသုံးများ)

onChange (ကျတော်တို့ ရဲ့ ပြောင်းလဲမှုတွေကို စောင့်ကြည့်ပေးတဲ့ event အများအားဖြင့် input တွေမှာ အသုံးများ)

onSubmit (ကျတော် တို့ form submission တွေ ကို စောင့်ကြည့်ပေးတဲ့ event အများအားဖြင့် form တွေမှာ အသုံးများ)

ဒီသုံးခုက တော့ တော်တော်လေး အသုံးများကြပါတယ်

အခုက လက်ရှိကျတော်တို့ ဆက်လေ့လာကြရမှာက onChange အကြောင်းပါပဲ ဘာလို့ onChange ကြီးကို ကျတော်က လာပြောပြတာလည်းဆိုတော့ ကျတော် ရှင်းပြဖို့တွေးထားတာလေး တစ်ခုရှိလို့ပါ

ကဲ စလိုက်ကြရအောင်ဗျ အရင်ဆုံး ကျတော်တို့ input လေးတခု ဖန်တီးလိုက်ပါမယ်

```
function Input() {  
  return (  
    <input type="text" />  
  )  
}  
  
export default Input
```

ပုံ - input component

ကျတော်တို့ အခုလိုမျိုး ဖန်တီးပြီးတော့ state နဲ့ စတင်ပီးတော့ control လုပ်လိုက်တော့မှာပါ ဘယ်လိုမျိုးလည်းဆိုတော့ ကျတော်တို့ ကုဒ်ကို ဒီလိုလေး ပြင်လိုက်ပါမယ်


```
import { useState } from "react";

function Input() {
  const [username, setUsername] = useState("");
  return (
    <input
      type="text"
      name="username"
      value={username}
      placeholder="Enter username"
    />
  );
}

export default Input;
```

ပုံ - input update

ကျတော်တို့ ဒီလိုလေး ရေးလိုက်ပီးရင် browser မှာ သွားပြီးတော့ ကျတော်တို့ စာရိုက်ကြည့်ရင် input မှာက ကျတော်တို့ စာရိုက်လို့မရတာ ကိုမြင်ရပါလိမ့်မယ်ဗျာ ဘာလို့လည်းဆိုတော့ ကျတော်တို့က input ရဲ့ value ထဲကို state ကိုထည့်ပြီးတော့ control လုပ်လိုက်တာဖြစ်တဲ့အတွက် ကျတော်တို့ရဲ့ state က ပြောင်းလဲမှုမလုပ်မချင်း သူ့မှာက စာတွေလာပေါ်မှာမဟုတ်ပါဘူး စာတွေပဲမပေါ်ဖြစ်တာပါ ကျတော်တို့ရိုက်လိုက်တဲ့ value တွေက အကုန်လုံး ရှိနေပါလိမ့်မယ် ကျတော်တို့ onChange လေး ဆက်ရေးကြည့်လိုက်ကြရအောင်ဗျာ

```
function Input() {
  const [username, setUsername] = useState("");
  const inputHandler = (e) => {
    console.log(e.target.value)
  }
  return (
    <input
      type="text"
      name="username"
      value={username}
      placeholder="Enter username"
      onChange={inputHandler}
    />
  );
}
```

React Learning

ပုံ - input handler added

ကျတော်တို့ အခုလိုမျိုးရေးပြီးတော့ console မှာသွားကြည့်လိုက်မယ်ဆိုရင်

```
a
s
d
f
a
s
d
f
a
s
d
f
a
s
d
f
```

ပုံ - console

ဒီလိုလေးမြင်ရမှာပါ ပြောချင်တာက ဒီ input လေးကို ကျတော်တို့က state နဲ့ control

လုပ်လိုက်ပီးပီပေါ့ဗျ သူ့ရဲ့ data လေးကို ယူချင်တဲ့အခါကျတော့ ကျတော်တို့က ဘယ်လိုမျိုးပြန်ပီးတော့

update လုပ်ရမလဲပေါ့ အဲလိုမျိုး ကို ကျတော်တို့က setUsername ဆိုတဲ့ အထဲကို event ရဲ့ value

လေးကို ထည့်ပေးလိုက်ရုံပါပဲ

```
function Input() {
  const [username, setUsername] = useState("");
  const inputHandler = (e) => {
    console.log(e.target.value)
    setUsername(e.target.value)
  }
  return (
    <input
      type="text"
      name="username"
      value={username}
      placeholder="Enter username"
      onChange={inputHandler}
    />
  );
}

export default Input;
```

ပုံ - update function

ကျတော်တို့ အဲလိုလေးရေးပြီးတဲ့အခါမှာ browser မှာ ပြန်သွားကြည့်လိုက်ရင် ကျတော်တို့ စာရိုက်လို့ရသွားတာကိုမြင်ရပါလိမ့်မယ်ဗျာ ဒီအချိန်မှာ ဘာတခုထူးဆန်းတာလည်းဆိုတော့ ကျတော်တို့က setState နဲ့မလုပ်ခင်တုန်းက စာရိုက်ရင်း အခုလိုမျိုး စာလုံးဆက်ပီး တော့မထွက်လာဘဲနဲ့ အခုမှ စာလုံးဆက်ပီးတော့ ထွက်လာတာကိုမြင်ရပါလိမ့်မယ်ဗျာ

အဲဒါက ဘာကြောင့်လည်းဆိုတော့ ကျတော်တို့က react ကနေပီးတော့ control မလုပ်သေးခင် setState ကို မထည့်ပေးသေးခင်မှာသူက ဘာကိုပဲသိတာလည်းဆိုတော့အခုလက်ရှိ change ဖြစ်သွားတဲ့ event ကိုပဲ သိတာပါ အဲဒါကို ကျတော်တို့က setState နဲ့ change လုပ်လိုက်တဲ့အခါမှာ input ရဲ့ value ထဲကို state က ဝင်သွားတဲ့အတွက် ကျတော်တို့က အရင်ရှိခဲ့တဲ့ value နဲ့ update value က string concatenation ဖြစ်သွားတာပါ အဲအတွက်ကြောင့် ကျတော်တို့က စာရိုက်လိုက်တဲ့အခါမှာ ဒီလိုလေး မြင်ရတာ ဖြစ်တာပါ မဟုတ်ရင်စာတလုံးချင်း ပေါ်နေရင်တော့ဘာမှမထူးသွားပါဘူးဗျာ ဒီလောက်ဆိုရင် ကျတော်တို့ onChange event အကြောင်းကို တော်တော်လေး နားလည်မယ်လို့ ထင်ပါတယ်

ဒါဆိုရင်ကျတော်တို့ နောက်ထပ် တမျိုးကိုဆက်ပြီးတော့လေ့လာလိုက်ကြရအောင်ဗျာ အဲဒါက ဘာလည်းဆိုတော့ onSubmit ဆိုတဲ့ဟာလေးပါ အဲကောင်က ဘယ်လိုမျိုး

မှာအသုံးများတာလည်းဆိုတော့ ကျတော်တို့က form တွေကို submit လုပ်တဲ့အခါမှာအသုံးပြုကြတာပါ သူ့မှာက ထူးခြားတာက တခြားကြီးကြီးမားမားမရှိပါဘူးဗျာ vanilla js မှာ သုံးတဲ့အတိုင်း အတူတူပါပဲဗျာ အဲလောက်ဆိုရင်တော့ ကျတော်တို့ event handling တွေကို နားလည်သွားမယ်လို့ထင်ပါတယ်

List Rendering

ကျတော်တို့ programmer တွေ developer တွေတိုင်းလိုလို DRY principle ကို ကြားဖူးကြမယ်လို့ထင်ပါတယ်

React application တစ်ခုမှာလည်းကျတော်တို့က အဲလို DRY principle ကို လိုက်နာပြီးတော့ ရေးကြတဲ့အခါမှာ ကျတော်တို့က render တွေလုပ်ကြတယ်မလား

ဒီထဲမှာကျတော်တို့ react မှာအသုံးပြုတာများတဲ့ list render

အကြောင်းကိုဆက်ပြီးတော့လေ့လာသွားကြ ရအောင်ဗျ

အရင်ဆုံး ကျတော်တို့ code လေးတွေကို ဒီလိုလေးရေးလိုက်ပါမယ်

```
import React from 'react'

function ListRendering() {
  const list = [
    "bate thar", "Kaung", "Myat" , "Hun"
  ]
  return (
    <div>ListRendering</div>
  )
}

export default ListRendering
```

ပုံ - list render data

ကျတော်တို့က နောက်ပိုင်းကျရင် ဒေတာတွေ array တွေကို ဒေတာပြန်ယူပြီးတော့ render လုပ်တဲ့အခါမှာ

ကျတော်တို့က ဒီဟာတွေကိုတခုချင်း လိုက်ပြီးတော့မရေးသွားပါဘူး အဲဒါကို ကျတော်တို့က ဘယ်လိုမျိုးရေးသွားမှာလည်းဆိုတော့

```
return (
  <div>
    {
      list.map((item, index)=> (
        <h3>{item}</h3>
      ))
    }
  </div>
)
```

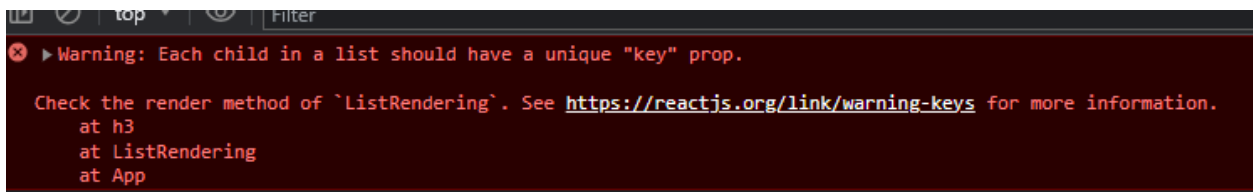
ပုံ - list rendering

ဒီလိုလေးရေးလိုက်ပြီးရင်ကျတော် တချက်ပြန်ရှင်းပြပေးပါမယ်

အဲဒါကဘာလည်းဆိုတော့ ကျတော်တို့က array data တခုကို render ချချင်တဲ့အခါမှာ array.map ကိုအသုံးပြုကြပါတယ်

React Learning

ဘာလို့ ဒီဟာကို ပဲ အသုံးပြုရတာလည်း တခြား ဟာတွေဖြစ်တဲ့ for တို့ for each တို့က သုံးလို့မရဘူးလားလို့မေးစရာရှိလာပါတယ်
သုံးလို့မရဘူးလားဆိုတော့ ရပါတယ် ဒါပေမယ့် map လို့မျိုး
တိုက်ရိုက်ကြီးရေးလို့ရတော့မှာမဟုတ်ဘဲနဲ့ သူ့ကို return ပြန်အောင်လုပ်ပေးပြီးတော့ အဲဒါကို ပြန်ပြီးတော့ render ချပေးရမှာပါ array.map မှာက ကျတော့သူက array ကို return လုပ်ပေးသွားတာဖြစ်တဲ့အတွက် ကျတော်တို့က ဒီဟာကို အသုံးပြုကြတာပါ
ကဲ ဆက်လိုက်ကြရအောင်ဗျ
အခု အဲလိုလေးရေးလိုက်တဲ့အခါမှာ ကျတော်တို့ browser မှာပြန်သွားကြည့်တဲ့အခါမှာ result ကို ကျတော်တို့ မြင်ရပါလိမ့်မယ်ဗျ ဒါပေမယ့်သူ့ကို ကျတော်တို့ console မှာ သွားကြည့်တဲ့အခါမှာ warning ပြနေတာကိုမြင်ရလိမ့်မယ်ဗျ အဲဒါက ဘာလည်းဆိုတော့



ပုံ - warning in console

ဘာလို့ အဲလိုပြောတာလည်းဆိုတော့ ကျတော်တို့ က react မှာ array data ကို list render လုပ်တဲ့အခါမှာ ကျတော်တို့က သူနဲ့ သက်ဆိုင်တဲ့ node ကို unique ဖြစ်တဲ့ key props လေးတခု ကိုထည့်ပေးပါလို့ပြောပါတယ်
ဘာကြောင့် အဲလိုတောင်းတာလည်းဆိုတော့ သူက v-dom (virtual dom) ထဲမှာ သွားပြီးတော့ render သွားချပေးတဲ့အခါမှာ ကျတော်တို့ အခု list render လုပ်လိုက်တဲ့ node တွေက အကုန်လုံးတူနေနိုင်တဲ့အနေအထား တခုရှိနေလို့ပါ အဲအတွက် ကြောင့် သူက v-dom (virtual dom) မှာပြန်စစ်တဲ့အခါမှာ အလွယ်တကူ ဖြစ်အောင်လို့ သူက အဲဒါကိုတောင်းတာပါ ပြီးတော့နောက်တချက်က အဲဒီ unique ဖြစ်တဲ့ key လေးကိုအသုံးပြုပြီးတော့ react က ပြောင်းလဲမှုတွေကို တိုက်စစ်ပြီးတော့မှ browser ကို render ချပေးတာပါ
ကျတော်တို့ ကုဒ်လေးတွေကိုနည်းနည်းပြင်လိုက်ရအောင်ဗျ

```
return (  
  <div>  
    {  
      list.map((item, index)=> (  
        <h3 key={index}>{item}</h3>  
      ))  
    }  
  </div>  
)  
}
```

ပုံ - adding key props

React Learning

ဒီမှာ ကျတော်တို့က index ဆိုတာကို ယူထားတာတွေရလိမ့်မယ်ဗျာ အဲဒီ index ဆိုတာက ဘာလည်း ကို တွေးမိချင်တွေးမိလိမ့်မယ်ဗျာ

တကယ်က ကျတော်တို့က render လုပ်တဲ့ array ရဲ့ index တွေကို ယူပြီးတော့ထည့်ပေးလိုက်တာပါ ကျတော်တို့ array တခုမှာ index တွေက တခုနဲ့တခုက မတူဘူးလေ အဲအတွက်ကြောင့် သူ့ကို unique ဖြစ်တဲ့အရာဆိုတော့ အဲဒါလေးကို ကျတော်တို့က အလွယ်လေးသုံးလိုက်တာပါ အဲလောက်ဆိုရင်တော့ list render ကို နားလည်သွားမယ်လို့ထင်ပါတယ်

Conditional Rendering

ကျတော်တို့ react ရဲ့ rendering ထဲမှာ နောက်ထပ် ထပ်လေ့လာရမယ့် rendering တမျိုးက တော့ Conditional rendering ပါပဲ ဘာလို့အဲဒါကို ကျတော်တို့ လေ့လာဖို့လိုတာလည်းဆိုတော့ ကျတော်တို့က logic တွေကို ရေးတဲ့အခါမှာ condition တွေက အရေးပါပါတယ် အဲအတွက်ကြောင့် ကျတော်တို့က condition အပေါ်မူတည်ပြီးတော့ render ပြပေးရတဲ့ သဘောမျိုး ရေးပေးကြရပါတယ်

ဥပမာ user တယောက်က login မဝင်ထားဘူးဆိုရင် ကျတော်တို့ ကသူနဲ့ပတ်သက်တဲ့အကြောင်းအရာတွေ ကို ဝင်ကြည့်ခွင့်မရဘူး တခြားအကြောင်းအရာတွေကို ပြင်ဆင်ခွင့်မရဘူး အဲလိုမျိုး သဘောတရားတွေကို ကျတော်တို့က condition တွေနဲ့ စစ်ပြီးတော့ လိုအပ်တဲ့ ဒေတာတွေကို render ချပေးတဲ့အရာမျိုးကို ပြောချင်တာပါ ကျတော်တို့ ကုဒ်လေးတွေကိုမရေးသေးခင်မှာ ဘာတွေသိထားဖို့လိုမလဲဆိုတော့ Javascript ရဲ့ condition စစ်တဲ့ ဟာလေးတွေကို သိဖို့လိုပါမယ်ဗျာ ကျတော်တို့ code ရေးလိုက်ကြရအောင်ဗျာ

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

class ConditionalRendering extends Component {
  constructor(props) {
    super(props)

    this.state = {
      sayHello: false
    }
  }

  render() {
    return (
      <div>ConditionalRendering</div>
    )
  }
}

export default ConditionalRendering
```

ပုံ - conditional rendering

React Learning

ကျတော်တို့ အခုလက်ရှိမြင်ရမှာက state တခုကို ဖန်တီးထားတာကိုမြင်ရပါလိမ့်မယ်ဗျ
ကျတော်တို့က အဲ state (boolean) အပေါ်မူတည်ပြီးတော့ ကျတော်တို့က render လုပ်သင့်မသင့်ကို
ဆုံးဖြတ်ပေးမှာပါ ကုဒ်လေးကို နောက်ထပ် ရေးလိုက်ကြရအောင်ဗျ

```
render() {  
  return (  
    <div>  
      {this.state.sayHello ? <h1>Hello</h1>: <h1>HI</h1>}  
    </div>  
  )  
}
```

ပုံ - update code

အဲလိုလေးရေးလိုက်တဲ့အခါမှာ ကျတော်တို့က ဘယ်လိုမျိုးမြင်ရမှာလည်းဆိုတော့ hi ဆိုတာလေးကိုမြင်
နေရမှာပါ ဘာလို့လည်းဆိုတော့ ကျတော်တို့က this.state.sayHello ရဲ့ value က false လို့ပေးထားလို့ ပါ
ပဲ

ကုဒ်ကိုကြည့်ရင်(ternary operator ကို သိတဲ့သူဆိုရင်တော့) နားလည်သွားမှာပါ
တကယ်လို့ ternary operator ကို မသိဘူးဆိုရင်တော့

ဒီမှာသွားလေ့လာပါ [Link](#)

ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ conditional render အကြောင်းကို နားလည်သွားလောက်ပါလို့
ထင်ပါတယ်

ကျတော်တို့က conditional render ကိုအသုံးပြုပြီးတော့ login ဝင်ထားတာလား မဝင်ထားဘူးလား
ဆိုတာမျိုးကို စစ်ပြီးတော့ render လုပ်ပေးတဲ့ logic လိုမျိုးကိုရေးလို့ရပါတယ်
ဒီလောက်ဆိုရင်တော့ နားလည်သွားမယ်လို့ထင်ပါတယ်

V-DOM (Virtual DOM)

ကျတော်တို့က react လို့ပြောလိုက်တာနဲ့ သူနဲ့ တွဲပြီး မျက်စိထဲမြင်ကြတာက virtual dom (react dom)
ပါပဲ

အဲလိုမျိုး virtual dom တခုကလည်း ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတာကို ကျတော်တို့က
သိထားဖို့လိုပါတယ်ဗျ

ကျတော်တို့ virtual dom နဲ့ actual dom တွေ အကြောင်းကို နည်းနည်းပါးပါးလေး အလုပ်လုပ်ပုံကို
သိအောင် လေ့လာလိုက်ကြရအောင်ဗျ

React ကဘာလို့ ဒီလောက်မြန်နေရတာလည်းပေါ့ အဲလိုမျိုး မေးခွန်းတွေက တော်တော်များများ
ကြားဖူးကြမယ်လို့ထင်ပါတယ်

React Learning

ဘာကြောင့်မြန်တာလည်းဆိုတော့ သူ့ရဲ့ react dom မှာအလုပ်လုပ်ပြီးတော့မှ browser ပေါ်ကို render လုပ်ပေးလိုက်တာ ကြောင့်မလို့ပါ
ကျတော်တို့က list render လုပ်တဲ့အခါမှာ react ကနေပြီးတော့ unique ဖြစ်တဲ့ key props လေးတောင်းတာကို မှတ်မိကြမယ် ထင်ပါတယ်
အဲဒါ key လေးက ဘယ်လောက်အထိအရေးပါလည်းဆိုတော့ သူက v-dom ထဲမှာ ပြောင်းလဲမှုဖြစ်သွားတဲ့အရာကိုအဲဒါလေးနဲ့ တိုက်စစ်တာပါ
ပြီးတော့မှ ပြောင်းလဲမှုမရှိတော့တဲ့ final root node ကို browser ပေါ်မှာ render လုပ်ပေးလိုက်တာပါ
ကျတော်တို့ရဲ့ js ရဲ့အလုပ်လုပ်ပုံကအရမ်းမြန်တာဖြစ်တဲ့အတွက် သူ့ကို ကျတော်တို့က တခြား process တွေကို အလုပ်လုပ်ခိုင်းပြီးတော့ လိုအပ်တဲ့ result လေးကိုပဲ browser ရဲ့ render engine ကိုလုပ်ခိုင်းတာဖြစ်တဲ့ အတွက် မြန်နေတာပါ
Dom တွေ render အကြောင်းကိုဖတ်ချင်ရင် [link](#)

Component lifecycle in React

ကျတော်တို့ရဲ့ react က component base ဖြစ်တဲ့အလျောက် သူ့မှာ lifecycle လေးတွေရှိတယ် ဗျ
အဲဒါလေးတွေကို လည်းကျတော်တို့က သိထားဖို့လိုပါတယ် သူ့ကိုနားမလည်ထားဘူးဆိုရင်
ကျတော်တို့က ကျတော်တို့ လေ့လာနေတဲ့ react ရဲ့အလုပ်လုပ်ပုံလေးကိုမြင်သာ မှာမဟုတ်ပါဘူး
ကျတော်ကတော့ react ရဲ့ lifecycle ကို အရင်ဆုံး class component နဲ့ရှင်းပြသွားပါမယ်
ဘာလို့လည်းဆိုတော့ သူ့မှာ ကျတော်တို့က ပိုပြီးတော့ အလုပ်လုပ်ပုံလေးကို မြင်သာလို့ပါ
ကဲ ဒါဆို ကျတော်တို့ စလိုက်ကြရအောင်ဗျ

In Class Components

ကျတော်တို့က component lifecycle ကိုလေ့လာတဲ့အခါမှာ lifecycle stage က သုံးဆင့်ရှိပါတယ်
အဲဒါက ဘာတွေလည်းဆိုတော့ mounting stage ရယ် updating stage ရယ် unmounting stage တွေပါ
ပဲ ဗျ အဲဒါတွေအပေါ်မူတည်ပြီးတော့ method က သုံးမျိုးရှိလာပါတယ် (အရင်ကတော့
အများကြီးရှိပါတယ် ပိုအသုံးများတဲ့အရာကိုပဲ ဦးစားပေးပြီးတော့ ရှင်းပြသွားမှာပါ)
အဲဒါတွေက ဘာတွေလည်းဆိုတော့

`componentDidMount()`

`componentDidUpdate()`

`componentWillUnmount()`

တွေပါပဲ ဗျ ကျတော်တို့ method name ကို ကြည့်လိုက်တာနဲ့ကို ဘယ်ဟာက ဘာအတွက်လည်းဆိုတာကို
ချက်ချင်း မြင်သာပါတယ်ဗျ

React Learning

componentDidMount()

အဲကောင်တွေ ဘယ်လိုအလုပ်လုပ်သွားလည်းဆိုတာကို ကျတော်တို့ တချက်ကြည့်လိုက်ကြရအောင်ဗျ ကျတော်တို့အရင်ဆုံး ဒီလိုလေးရေးလိုက်ပါမယ်

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

class Lifecycle extends Component {
  componentDidMount(){
    console.log("component life cycle start")
  }
  render() {
    return (
      <div>Lifecycle</div>
    )
  }
}

export default Lifecycle
```

ပုံ - lifecycle start

ကျတော်တို့ အခုလက်ရှိသုံးထားတဲ့ componentDidMount() ဆိုတဲ့ method လေး က သူ့ name အတိုင်းပါပဲ

ကျတော်တို့ component လေးကို react ကနေပြီးတော့ Dom tree မှာ စတင်ပြီးတော့ create လုပ်လိုက်တာနဲ့ ဒီကောင်လေးက စတင်ပြီးတော့အလုပ်လုပ်ပေးပါတယ်

အဲဒီ method လေးက ဘာလို့အရေးကြီးတာလည်းဆိုတော့ ကျတော်တို့က browser ရဲ့ side effect (api data fetching) တွေကို အသုံးပြုချင်တဲ့အခါမှာ ပထမဦးဆုံးအကြိမ် fetch စေချင်တဲ့ အခါမျိုးမှာအသုံးပြုကြပါတယ်

အဲ method လေးက react ရဲ့ lifecycle ထဲမှာ ပထမဦးဆုံး တကြိမ်သာ အလုပ်လုပ်ပေးတာပါ ကျန်တဲ့ အချိန် အခြေအနေ မှာ ပြန်ပြီးတော့အလုပ်မလုပ်ပေးပါဘူး ဗျ

အဲဒီ method ထဲမှာ ကျတော်တို့ api request လေးတွေကို ခေါ်သုံးလို့ရပါတယ် အဲအတွက် ကျတော်တို့ ဒီလိုလေး ရေးလိုက်ပါမယ်

```
class Lifecycle extends Component {
  getData = async () => {
    const data = await fetch("https://jsonplaceholder.typicode.com/users")
    const res = await data.json()
    console.log(res)
  }
  componentDidMount() {
    this.getData();
    console.log("component life cycle start");
  }
  render() {
    return <div>Lifecycle</div>;
  }
}
```

ပုံ - use side effect

ကျတော်တို့က fetch ကိုအသုံးပြုပြီးတော့ ကျတော်တို့ web page ရဲ့ ပထမဆုံးအကြိမ် render ချလိုက်တဲ့ အချိန်မှာ ဒေတာတွေကို fetch လုပ်လိုက်ပါတယ်
အဲဒါ ကို ကျတော်တို့ browser console မှာကြည့်ရင် ဒီလိုလေးတွေရပါလိမ့်မယ်

```
component life cycle start
component life cycle start
▶ (10) [{-}, {-}, {-}, {-}, {-}, {-}, {-}, {-}, {-}, {-}]
▶ (10) [{-}, {-}, {-}, {-}, {-}, {-}, {-}, {-}, {-}, {-}]
>
```

ပုံ - console log

ကျတော်တို့က componentDidMount က component မှာ ဦးဆုံးအလုပ်လုပ်ပေးတယ်လို့ ထင်နေပေမယ့်
တကယ်က သူက အရင်ဆုံး အလုပ်လုပ်မပေးပါဘူး
သူ့အရှေ့မှာ ဘာတွေအလုပ်လုပ်သေးတာလည်းဆိုတော့ constructor နဲ့ render က
အလုပ်လုပ်ပေးပါတယ်
ပြီးတော့မှ သူက browser dom ပေါ်ကို ရောက်တော့မှ အလုပ်လုပ်ပေးတာပါ

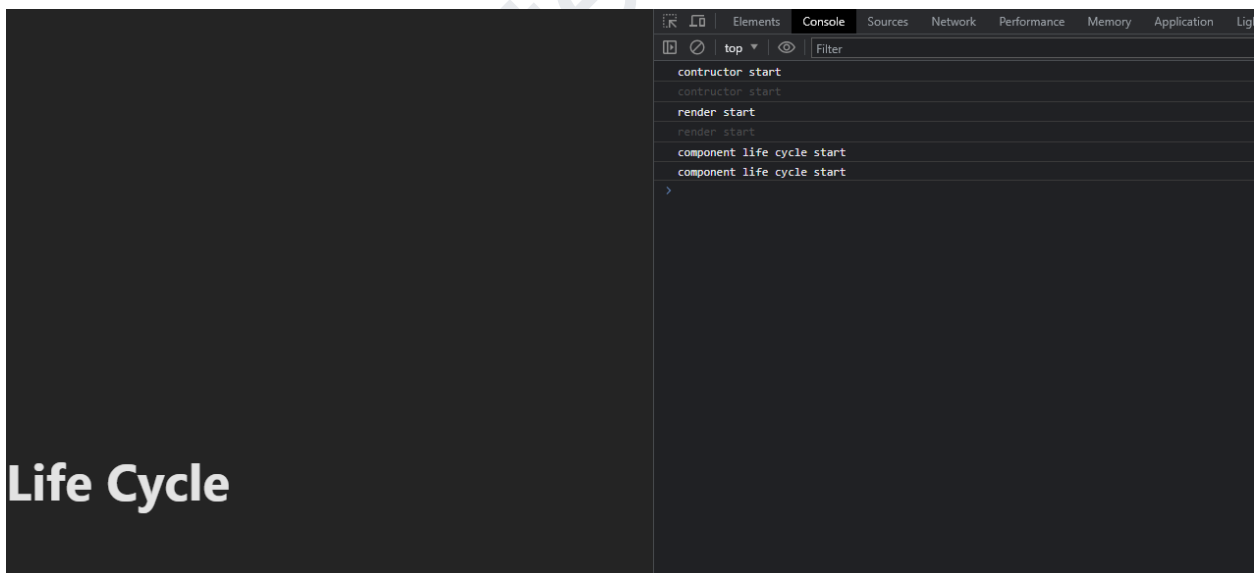
React Learning

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

class Lifecycle extends Component {
  constructor(props) {
    super(props)
    console.log("constructor start")
  }
  componentDidMount() {
    console.log("component life cycle start");
  }
  render() {
    console.log("render start")
    return (
      <>
        <h1>Life Cycle</h1>
      </>
    );
  }
}

export default Lifecycle;
```

ပုံ - console log



ပုံ - result

အဲဒါကို ကြည့်လိုက်ရင် ကျတော်တို့ရဲ့ component lifecycle က ဘယ်အချိန်မှာ စတာလည်းဆိုတာကို မြင်သွား မယ်လို့ ထင်ပါတယ် ဗျ

React Learning

အဲဒီလောက်ဆိုရင်တော့ ကျတော်တို့ရဲ့ lifecycle method တွေထဲ ပထမဆုံး တခုဖြစ်တဲ့ componentDidMount ဆိုတဲ့ဟာလေးကို နားလည်သွားကြမယ်လို့ထင်ပါတယ်
နောက်တခု ကိုဆက်သွားကြရအောင်ဗျ

componentDidUpdate()

ကျတော်တို့ ဒီ method လေးက ကျတော့ ဘယ်လိုမျိုးလည်းဆိုတော့ သူက ကျတော်တို့ရဲ့ component ထဲမှာ ရှိတဲ့ state တစ်ခုခု က ပြောင်းလဲမှုလုပ်လိုက်တိုင်း state update ဖြစ်သွားတဲ့အခါတိုင်းမှာ သူက rerender လုပ်ပေးတာ ပါ

အဲဒါဆို ကျတော်တို့ ကုဒ်လေးကို ဒီလိုလေး ပြင်လိုက်ပါမယ်

```
class Lifecycle extends Component {
  constructor(props) {
    super(props)

    this.state = {
      count: 0
    }
  }
  getData = async () => {
    const data = await fetch("https://jsonplaceholder.typicode.com/users")
    const res = await data.json()
    console.log(res)
  }
  componentDidMount() {
    this.getData();
    console.log("component life cycle start");
  }
  componentDidUpdate(){
    console.log("component state is update")
  }
  increaseHandler = () => {
    this.setState({
      count: this.state.count + 1
    })
  }
  render() {
    return (
      <>
        <h1>{this.state.count}</h1>
        <button onClick={this.increaseHandler}>Increase</button>
      </>
    );
  }
}

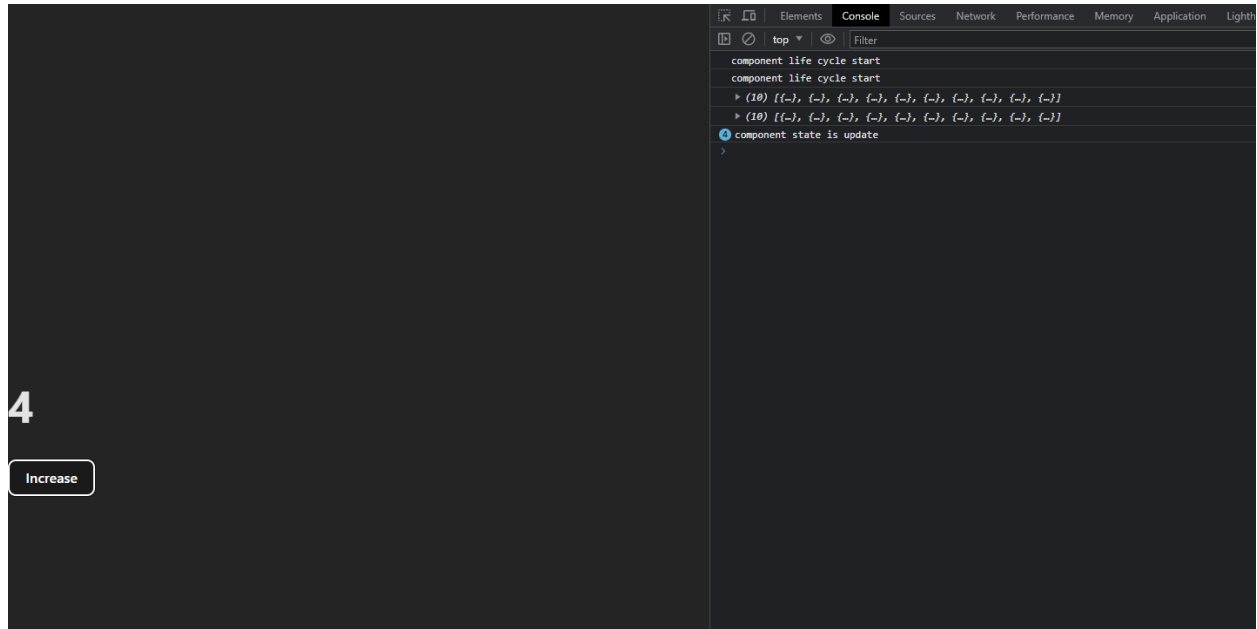
export default Lifecycle;
```

ပုံ - adding componentDidUpdate method

ကျတော်တို့ ဒီအချိန်မှာ ဘာလို့ mounting method ကို မဖျက်သေးတာလည်းဆိုတော့ သူက ပထမဆုံးအကြိမ်မှာအလုပ်လုပ်သွားတာကိုပဲ မြင်သာစေချင်လို့ပါ

React Learning

ကဲ ကျတော်တို့ button လေးကို click လိုက်ကြည့်ကြရအောင်ဗျ အဲလိုနီပဲလိုက်တဲ့အချိန်မှာ browser console မှာ ဘယ်လိုမျိုးမြင်ရမှာလည်းဆိုတော့ ဒီလိုလေး မြင်ရပါလိမ့်မယ် ဗျ



ပုံ - browser console

ကျတော်တို့ရဲ့ componentDidMount လေးက ဦးဆုံး တကြိမ် အလုပ်လုပ်ပြီးသွားတာနဲ့ သူက ထပ် အလုပ်မလုပ်တော့ပါဘူး

ကျတော်တို့ state ကို update လုပ်လိုက်တဲ့အချိန်မှာ componentDidUpdate ဆိုတဲ့ဟာလေးက အလုပ်လုပ်ပေးတာကိုမြင်ရပါလိမ့်မယ်ဗျ အဲလောက်ဆိုရင်တော့ ကျတော်တို့ရဲ့ component တစ်ခုမှာ mounting တွေ updating တွေကို မြင်သာသွားမယ်လို့ ထင်ပါတယ်

Prevent infinite api request in componentDidMount Method

ကျတော်တို့ နောက်တစ်ခုထပ်ပြီးတော့လေ့လာလိုက်ကြရအောင်ဗျ componentDidUpdate ရဲ့ method ထဲမှာ

Parameter ၂ ခုကိုလက်ခံပါတယ်ဗျ တခုက previousProps နဲ့ နောက်တစ်ခုက တော့ previousState ဖြစ်ပါတယ်

အဲဒါကို ကျတော်တို့ တချက်လောက်လေ့လာလိုက်ကြရအောင် ပါ ဘာလို့အဲဒါကို

ကျတော်ရှင်းပြတာလည်းဆိုတော့ သူက အရေးကြီးတာမလို့ပါ

ကျတော်တို့ ကုဒ်လေးတွေကို ဒီလိုလေးရေးလိုက်ပါမယ်

```
import React, { Component } from "react";

class Lifecycle extends Component {
  constructor(props) {
    super(props)

    this.state = {
      count: 0,
      data : []
    }
  }
  getData = async () => {
    const data = await fetch("https://jsonplaceholder.typicode.com/users")
    const res = await data.json()
    console.log(res)
    this.setState({
      data: res
    })
  }
  componentDidMount() {
    console.log("component life cycle start");
  }
  componentDidUpdate(){
    this.getData();
    console.log("component state is update")
  }
  increaseHandler = () => {
    this.setState({
      count: this.state.count +1
    })
  }
  render() {
    return (
      <>
        <h1>{this.state.count}</h1>
        <button onClick={this.increaseHandler}>Increase</button>
        {this.state.data.map((item,index) =>(
          <p key={index}>{item.name}</p>
        ))}
      </>
    );
  }
}

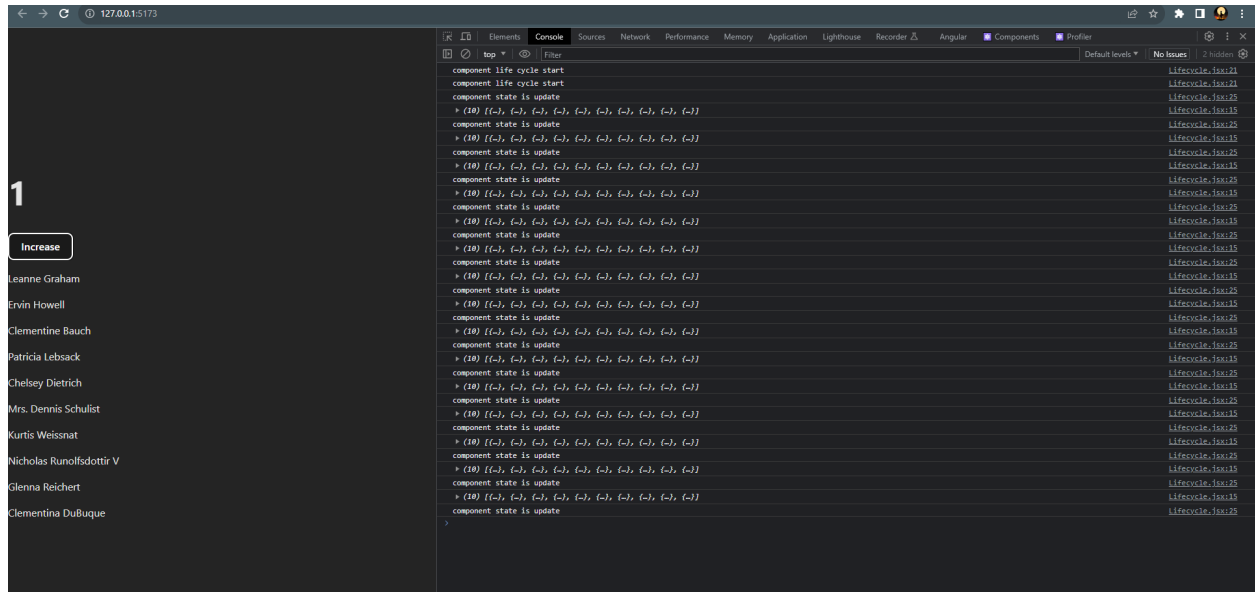
export default Lifecycle;
```

ပုံ - lifecycle method update

ကျတော်တို့က ဒီတစ်ခါမှာ ကျတော်တို့က increment ဆိုတဲ့ button လေးကို click လုပ်တဲ့အချိန်မှာ ကျတော်တို့က api လေးကို fetch လုပ်ပေးဖို့အတွက်ရေးလိုက်တာပါ

React Learning

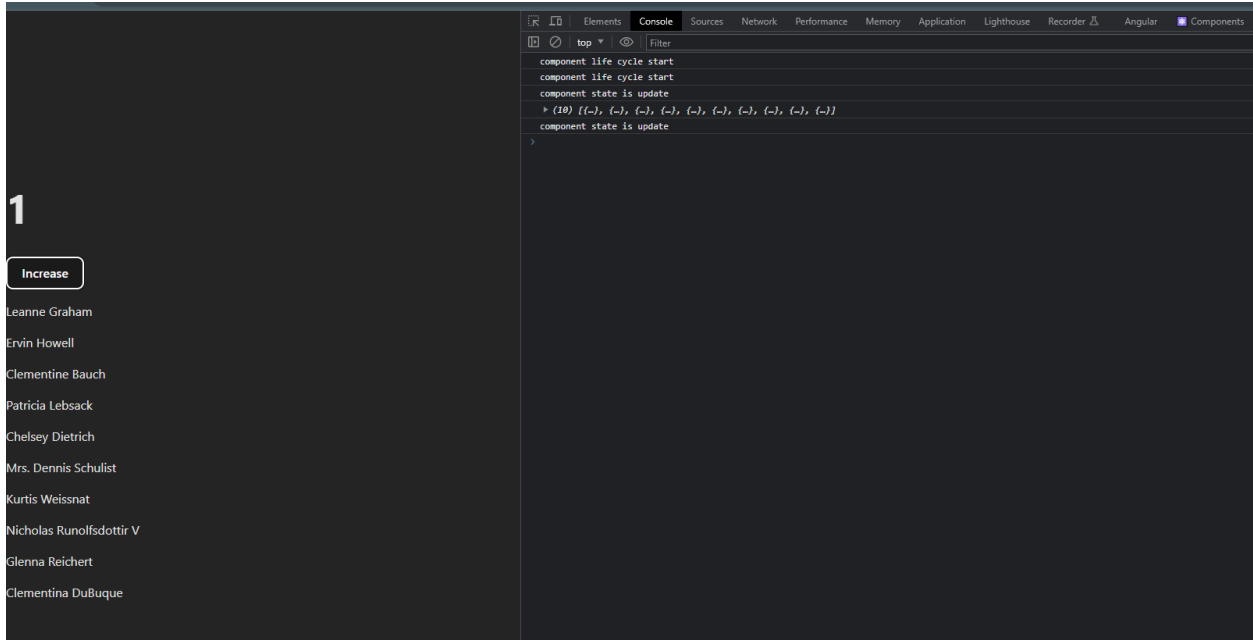
အဲလို fetch ပီးတော့ data ကို render လုပ်လိုက်မှာပါ အဲအချိန်မှာ ကျတော်တို့က button ကို click လုပ်လိုက်တဲ့အချိန်မှာ ဘာဖြစ်သွားတာလည်းဆိုတော့ api request ကို infinite လုပ်နေတာကိုမြင်ရပါလိမ့်မယ်
အဲဒါကို ကျတော်တို့က သတိထားပေးရမှာပါ အဲလိုမျိုးအလုပ်လုပ်ချင်တဲ့အခါမှာသူ့ကို prevent လုပ်ဖို့ အတွက် ကျတော်တို့က previousState ကို ပြန်ယူပြီးတော့ ပြန်အလုပ်လုပ်စေခိုင်းရပါတယ်



ပုံ - infinite api request
အဲဒါကို ကျတော်တို့က ကုဒ်လေးကို ဒီလိုလေးပြင်ရေးလိုက်ပါမယ်

```
componentDidUpdate(prevProps, prevState){  
  if(prevState.count !== this.state.count){  
    this.getData();  
  }  
  console.log("component state is update")  
}  
  
increaseHandler = () => {  
  this.setState({  
    count: this.state.count + 1  
  })  
}
```

ပုံ - update componentDidUpdate method
အဲလိုလေးရေးပြီးတော့ ကျတော်တို့က infinite api request ကို ကာကွယ်လို့ရပါတယ်ဗျ



ပုံ - api fetch only one time

အဲလောက်ဆိုရင်တော့ ကျတော်တို့ componentDidMount method ကို နားလည်သွားမယ်လို့ မျှော်လင့်ပါ တယ်

နောက်ထပ်တစ်ခုကို ထပ်လေ့လာသွားကြရအောင်ပါ

componentWillUnmount()

ကျတော်တို့ ဒီ method ကိုလေ့လာတဲ့အချိန်မှာ ကျတော်တို့က ဘာကိုသိထားရမှာလည်းဆိုတော့ သူက ဘယ်အချိန်မှာအလုပ်လုပ်ပေးတာလည်းပေါ့ ဒီဟာကလည်း သူ့ရဲ့ name အတိုင်းပါပဲ သူက ကျတော်တို့ရဲ့ browser dom ကနေပြီးတော့ အဲ component ကို remove လုပ်လိုက်တဲ့အချိန်မှာအသုံးပြုတာပါ

နားလည်အောင်ပြောရမယ်ဆိုရင်

ကျတော်တို့ က component ကို browser dom tree ပေါ်ကို render လုပ်လိုက်တဲ့အချိန်မှာ component နဲ့ သက်ဆိုင်တဲ့ mounting နဲ့ updating က အလုပ်လုပ်ပေးသွားတာပေါ့ဗျာ တချို့အခြေအနေတွေမှာ ကျတော်တို့က event တခုခုကို long term အနေနဲ့အလုပ်လုပ်ခိုင်းခဲ့တာမျိုးတွေရှိတယ်ဗျာ မြင်သာအောင်ပြောရရင် ကျတော်တို့ component တခုကို render ချလိုက်တဲ့အခါမှာ သူ့ရဲ့ သက်ဆိုင်တယ် event တွေကို react က dom ပေါ်တင်ပေးတယ်ပေါ့ဗျာ ဒီအထဲမှာ တချို့ event တွေက window မှာ သွားပီးတော့အလုပ်လုပ်တာမျိုးတွေ ရှိတယ်ဗျာ အဲလိုမျိုး event တွေကို ကျတော်တို့အနေနဲ့ component ကို remove လုပ်လိုက်တဲ့အချိန်မှာ ပြန်ပီးတော့ cancel လုပ်ပေးသွားရမယ့်အခြေအနေမျိုးမှာ အသုံးပြုပေးကြ တာပါ ကဲ အဲဒါဆိုရင် ကျတော်တို့

React Learning

ကုဒ်လေးတွေကိုရေးလိုက်ကြရအောင်ဗျ၊ ဒီအချိန်မှာ ကျတော်တို့ က ဘာတွေ သုံးမှာလည်းဆိုတော့
component ၊ ခုကိုအသုံးပြုပါမယ်ဗျ
ComponentA နဲ့ Component B ပေါ့
ကျတော်တို့ ComponentA ထဲမှာ ဒီလိုလေးရေးလိုက်ပါမယ်

```
import React, { Component } from 'react'
import ComponentB from './ComponentB'

class ComponentA extends Component {
  constructor(props) {
    super(props)

    this.state = {
      show: false
    }
  }

  showHandler = () =>{
    this.setState({
      show: !this.state.show
    })
  }

  render() {
    return (
      <>
        {this.state.show && <ComponentB />}
        <button onClick={this.showHandler}>Show</button>
      </>
    )
  }
}

export default ComponentA
```

ပုံ - ComponentA

ComponentB ကိုလည်း ဒီလိုလေး ရေးလိုက်ပါမယ်

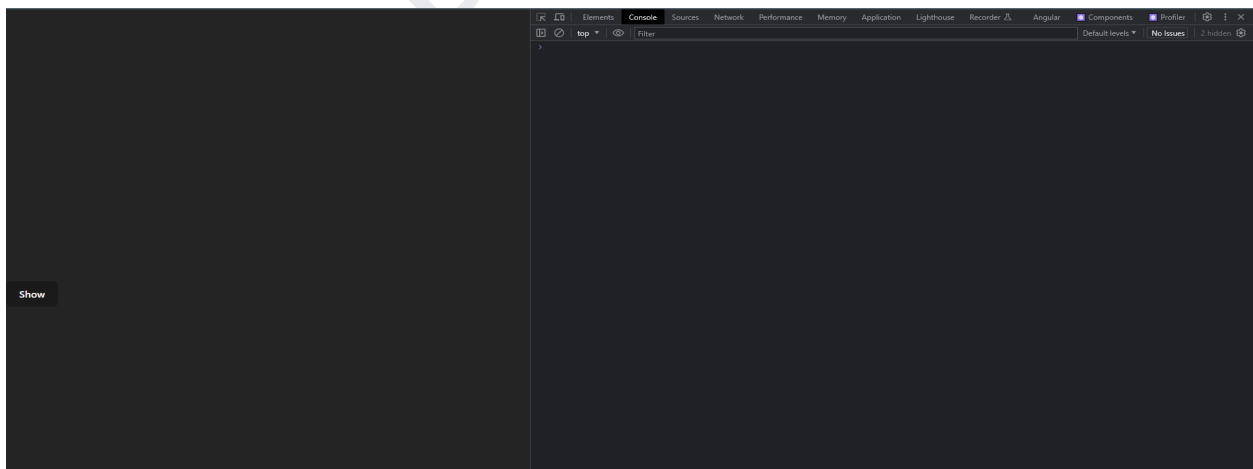
```
import React, { Component } from 'react'

class ComponentB extends Component {
  resizeHandler=() => {
    console.log("resizing")
  }
  componentDidMount() {
    window.addEventListener("resize" , this.resizeHandler )
  }
  render() {
    return (
      <div>ComponentB</div>
    )
  }
}

export default ComponentB
```

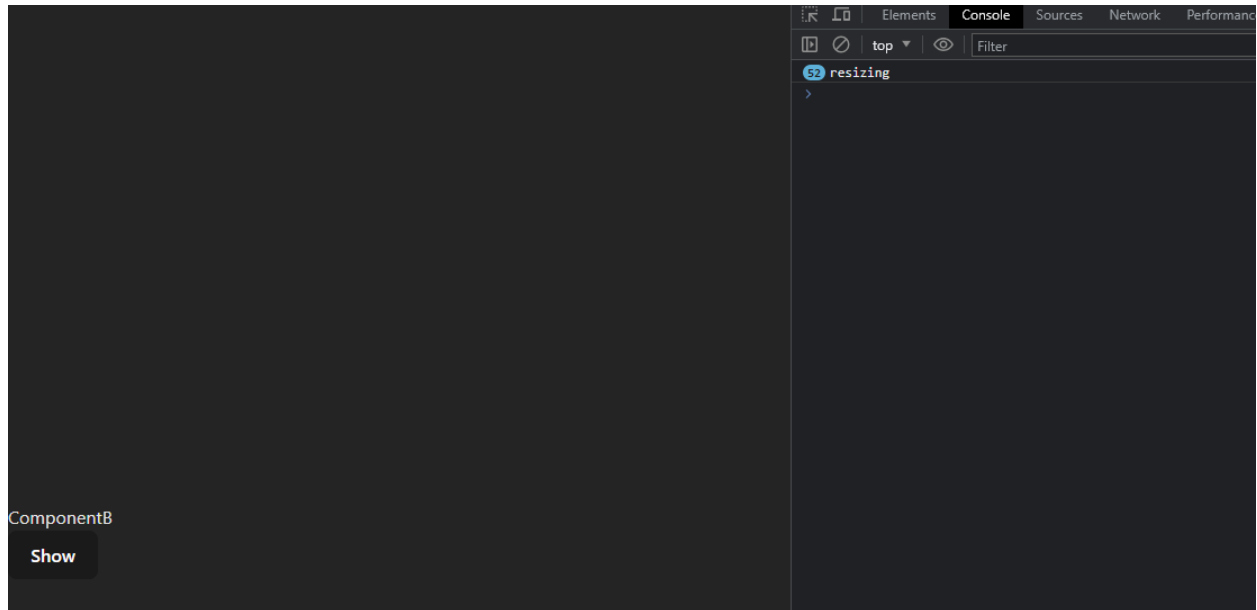
ပုံ - ComponentB

အဲလိုလေးရေးလိုက်ပြီးသွားရင် ကျတော်တချက် ရှင်းပြပေးပါမယ်
ပထမဦးဆုံး ကျတော်တို့က ComponentA ထဲမှာ conditional render ကိုအသုံးပြုပြီးတော့
ComponentB ကို browser dom ပေါ်ကိုတင်လိုက်ပါမယ် ပြီးသွားရင်သူက show : false
ဖြစ်သွားတဲ့အချိန်မှာ ကျတော်တို့က ComponentB ကို dom ကနေပြီးတော့ remove
လိုက်လုပ်ပါမယ်ဆိုပြီးတော့ ကျတော် ရေးထားပါ
ComponentB မှာ က ကျတော့ ကျတော်တို့က ComponentB က browser dom ပေါ်ကို mounting
လုပ်လိုက်တာနဲ့ ကျတော်တို့က window ကို event တခုကို add ခဲ့ပါတယ် အဲကောင်က ဘာလည်းဆိုတော့
ကျတော်တို့က resize လုပ်ခဲ့ရင်ပေါ့ အဲလိုမျိုး ရေးခဲ့တာပါ ကဲ အဲလောက်ဆိုရင်တော့ ကျတော်တို့
စမ်းသပ်ကြည့်လိုက်ကြရအောင်ဗျ



ပုံ - render ComponentA

ကျတော်တို့ show ကိုနှိပ်ကြည့်ပြီးတော့ browser ကို resize လုပ်တဲ့အချိန်မှာ console က
အလုပ်လုပ်နေတာကိုမြင်ရပါလိမ့်မယ်ဗျ



ပုံ - window resize

အဲလိုမျိုး အခြေအနေမှာ ကျတော်တို့က show ကိုထပ်ဖျောက်လိုက်မယ်ဆိုပါစို့ အဲအချိန်မှာ ကျတော်တို့က ဘာတွေထပ်ဖြစ်နေမှာလည်းဆိုတော့ resize က ထပ်ပြီးတော့အလုပ်လုပ်နေဦးမှာပါ တကယ့် logical အရဆိုရင်တော့ အဲဒါ က အဓိပ္ပာယ်မရှိတော့ပါဘူး ဘာလို့လည်းဆိုတော့ ကျတော်တို့က ComponentB ကို browser dom ကနေပြီးတော့ remove လုပ်လိုက်တဲ့အတွက် သူနဲ့သက်ဆိုင်တဲ့ event တွေအကုန်လုံးကို ပါ ပြန်ပြီးတော့ cancel ဖြစ်သွားရမှာပါ အဲလိုမျိုး အလုပ်လုပ်တာကို မြင်သာစေချင်တယ်ဆိုရင် ကျတော်တို့ က componentWillUnmount ကို အသုံးပြုလိုက်ပါမယ် ဘယ်လိုမျိုးလည်းဆိုတော့

```

ComponentB.jsx > ...
import React, { Component } from 'react'

class ComponentB extends Component {
  resizeHandler=() => {
    console.log("resizing")
  }
  componentDidMount() {
    window.addEventListener("resize" , this.resizeHandler )
  }
  componentWillUnmount(){
    window.removeEventListener("resize", this.resizeHandler)
  }
  render() {
    return (
      <div>ComponentB</div>
    )
  }
}

export default ComponentB

```

React Learning

ပုံ - cancel event

ကျတော်တို့ရဲ့ window မှာ add လုပ်ခဲ့တဲ့ event ကို componentWillUnmount ဆိုတဲ့ method ထဲမှာ ပြန်ပြီးတော့ cancel လုပ်သွားတာကို မြင်ရပါမယ်။ အဲဒီအတွက် ကျတော်တို့က ComponentB ကို browser Dom ထဲကနေပြီးတော့ remove လုပ်လိုက်တဲ့အချိန်မှာ သူနဲ့ သက်ဆိုင်တဲ့ event တွေ function တွေကိုအကုန် ပြန်ဖျက်ပေးသွားပြီးတော့ component ကလည်း လုံးဝကို independent ဖြစ်သွားမှာပါ။ အဲလောက်ဆိုရင်တော့ ကျတော်တို့ရဲ့ component lifecycle ကို နားလည်သွားမယ်လို့ မြင်ပါတယ်။

In Functional Components

useEffect hook (lifecycle hook)

ကျတော်တို့က lifecycle method တွေကို ကျတော်တို့က Class Component မှာပဲ အသုံးပြုလို့ရခဲ့ပါတယ်။ အဲဒါကို ကျတော်တို့က react ကနေပြီးတော့ lifecycle method တွေနဲ့ တူတဲ့ react hook လေးတခု ကို release လုပ်ပေးခဲ့ပါ တယ်။ အဲဒါက useEffect hook ပဲဖြစ်ပါတယ်။ အဲကောင်ကို ကျတော်တို့က တော်တော်လေးအသုံး များတဲ့ hook အနေနဲ့လည်း သတ်မှတ်ပေးလို့ရပါတယ်။ ဘာလို့ဆိုတော့ သူက browser side effect တွေကို အသုံးပြုရမှာ တော်တော်လေး အသုံးဝင်တာမလို့ပါ။ ကျတော်တို့ အရင်ဆုံး ကုဒ်လေးတွေကို ရေးလိုက်ကြရအောင်ပါ။

```
import {useEffect} from 'react'

function UseEffectHook() {
  useEffect(() => {
    console.log("useEffect hook is start")
  })

  return (
    <div>
      UseEffectHook
    </div>
  )
}

export default UseEffectHook
```

ပုံ - useEffect hook using

အဲလိုမျိုး ရေးပြီးတော့ ကျတော်တို့သွားကြည့်လိုက်ရင် console မှာ သူက အလုပ်လုပ်ပေးသွားတာကိုမြင်သွားပါလိမ့်မယ်။

React Learning

ကျတော်တို့က useEffect hook လေးကို တချက်ရှင်းပြပေးမယ်ဗျာ အဲကောင်က ဘယ်လိုမျိုးအလုပ်လုပ်ပေးတာလည်းဆိုတော့ သူ့မှာ က parameter ၂ခုကို လက်ခံပေးတယ်ဗျာ ပထမ တစ်ခုက callback function ဖြစ်ပြီးတော့ နောက်တစ်ခုက dependencies array လေးပါပဲ ဗျာ အဲဒါတွေက ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတော့ component တခုက browser dom ပေါ်ကို စပြီးတော့ render ဖြစ်သွားတာနဲ့ callback လေးက အလုပ်လုပ်တာပါ ပြီးတော့ အဲ callback လေးကပဲ state ကို change လုပ်တဲ့အခါမှာ ပြန်အလုပ်လုပ်ပေးသွားတာပါ ကျတော်တို့ ဘယ်လိုမျိုးရေးကြမလဲဆိုတော့

```
import {useEffect, useState} from 'react'

function UseEffectHook() {
  const [count, setCount] = useState(0)
  useEffect(() => {
    console.log("useEffect hook is start")
  })

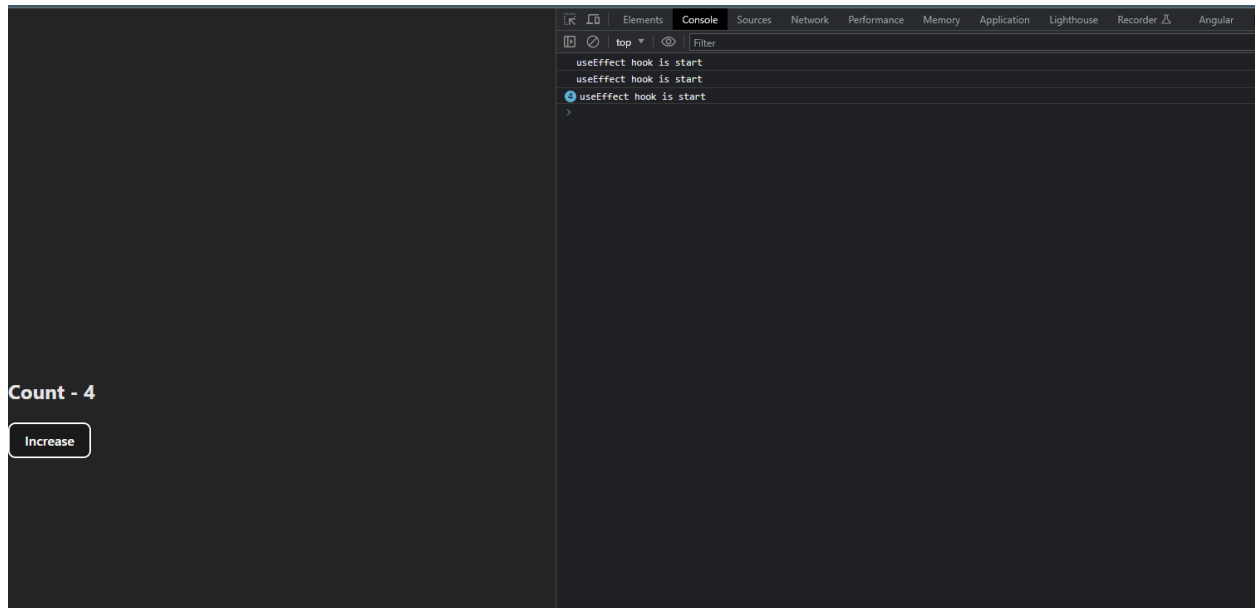
  return (
    <div>
      <h2>Count - {count}</h2>
      <button onClick={() => {
        setCount(prev => prev + 1)
      }}>Increase</button>
    </div>
  )
}

export default UseEffectHook
```

ပုံ - state update in useEffect

အဲလိုလေး ရေးပြီးတော့ ကျတော်တို့ browser မှာသွားစမ်းကြည့်တဲ့အခါမှာ ဒီလိုလေးအလုပ်လုပ်တာကိုမြင်ရပါလိမ့်မယ်

React Learning



ပုံ - result

ပြီးတော့ ကျတော်တို့က update အတွက်လည်းအလုပ်လုပ်ပေးတယ်ဆိုတော့ သူ့မှာ unmounting ကိုလည်းအလုပ်လုပ်ပေးပါတယ် အဲဒါက ဘယ်လိုရေးရတာလည်းဆိုတော့ callback function တခု ကို ကျတော်တို့ က return ပြန်ပေးလိုက်ရတာပါ ဘယ်လိုရေးလို့ရတာလည်းဆိုတော့

React Learning

```
import {useEffect, useState} from 'react'

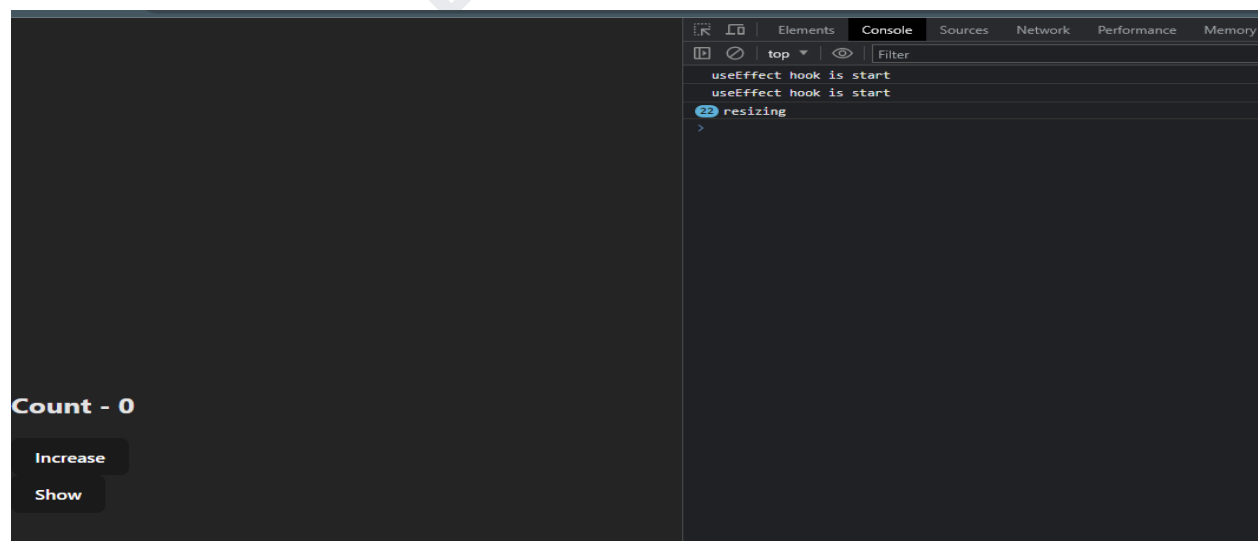
function UseEffectHook() {
  const [count, setCount] = useState(0)
  const resizeHandler = () => {
    console.log("resizing")
  }
  useEffect(() => {
    window.addEventListener("resize", resizeHandler)
    console.log("useEffect hook is start")
    return () => {
      window.removeEventListener("resize", resizeHandler)
    }
  })

  return (
    <div>
      <h2>Count - {count}</h2>
      <button onClick={() => {
        setCount(prev => prev + 1)
      }}>Increase</button>
    </div>
  )
}

export default UseEffectHook
```

ပုံ - cancel addEventListener in useEffect

ကျတော်တို့ browser မှာသွားစမ်းကြည့်ရင် ဒီလိုလေး မြင်ရပါလိမ့်မယ်ဗျ



ပုံ - result

React Learning

Using sideEffect in useEffect hook

ကျတော်တို့က useEffect hook တခုမှာ component lifecycle တွေအကုန်လုံး အလုပ်လုပ်တာကိုနားလည်သွားမယ်လို့ထင်ပါတယ်
အဲဒီအတွက် ကျတော်တို့က first reload data ကို ယူတဲ့အခါကျရင် သူ့အထဲမှာ လာရေးလိုက်ရင် ကျတော်တို့ လိုချင်တဲ့ data ကို ရသွားမှာပါ
ကျတော်တို့ ကုဒ်လေး နည်းနည်းလောက်ရေးလိုက်ကြရအောင်ဗျ

```
import {useEffect, useState} from 'react'

function UseEffectHook() {
  const [count, setCount] = useState(0)
  const [data, setData] = useState([])
  const resizeHandler = () => {
    console.log("resizing")
  }
  const getData = async () => {
    const resData = await fetch("https://jsonplaceholder.typicode.com/users")
    const resdata = await resData.json();
    setData(resdata)
    console.log(resdata)
  }
  useEffect(() => {
    window.addEventListener("resize", resizeHandler)
    console.log("useEffect hook is start")
    getData() // api side effect function
    return () => {
      window.removeEventListener("resize", resizeHandler)
    }
  })

  return (
    <div>
      <h2>Count - {count}</h2>
      <button onClick={() => {
        setCount(prev => prev + 1)
      }}>Increase</button>
    </div>
  )
}

export default UseEffectHook
```

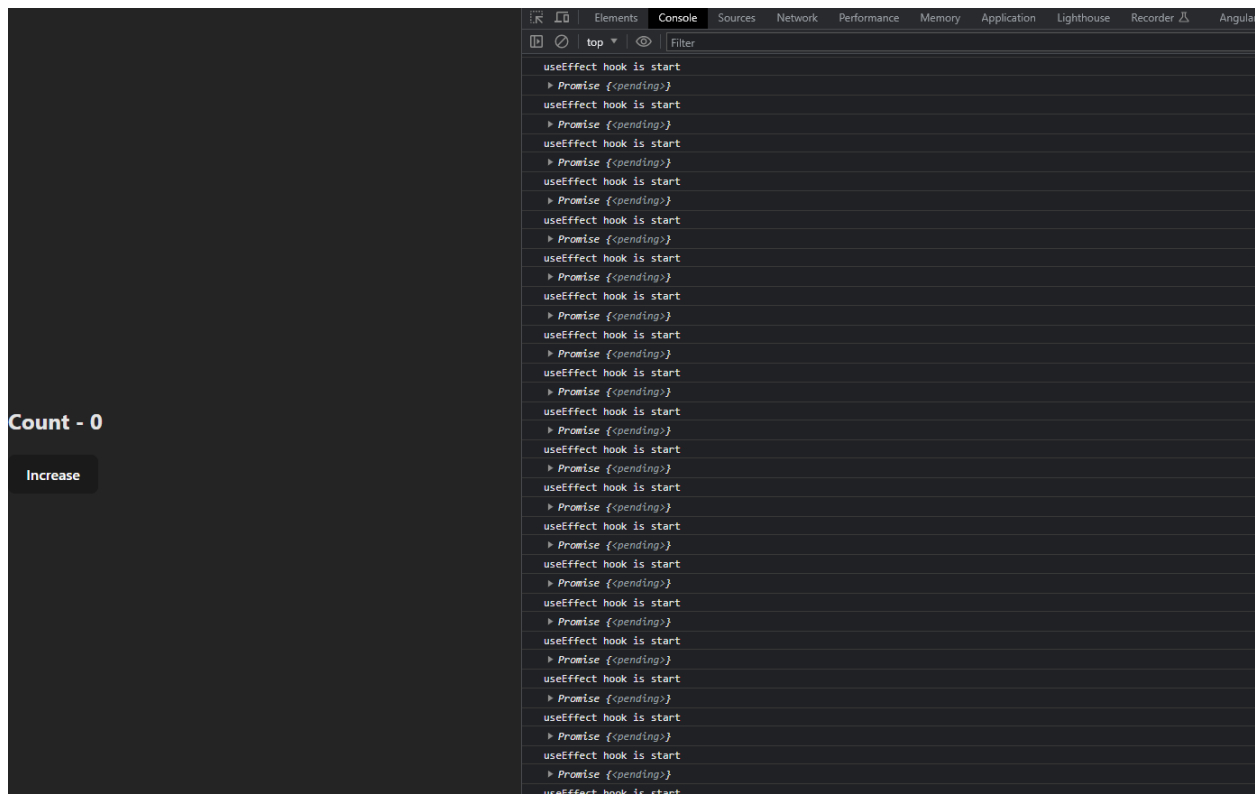
ပုံ - using useEffect in side effect

ကျတော်တို့အဲလိုလေးရေးပြီးတော့ browser မှာသွားကြည့်လိုက်ရင် ကျတော်တို့ ဘာကိုမြင်ရမှာလည်းဆိုတော့ သူက ထုံးစံအတိုင်း infinite api request လုပ်နေတာကိုမြင်ရပါလိမ့်မယ် ဗျ အဲဒါက ဘာဖြစ်လို့လည်းဆိုတော့ကျတော်တို့ useEffect hook မှာက ကျတော် အစကပြောခဲ့ပါတယ်

React Learning

သို့မှာ parameter ၂ခုကို လက်ခံပေးပါတယ်ဆိုပီးတော့ ဗျ တခုက callback ဖြစ်ပြီးတော့ နောက်တစ်ခုက dependency array ဆိုပြီးတော့

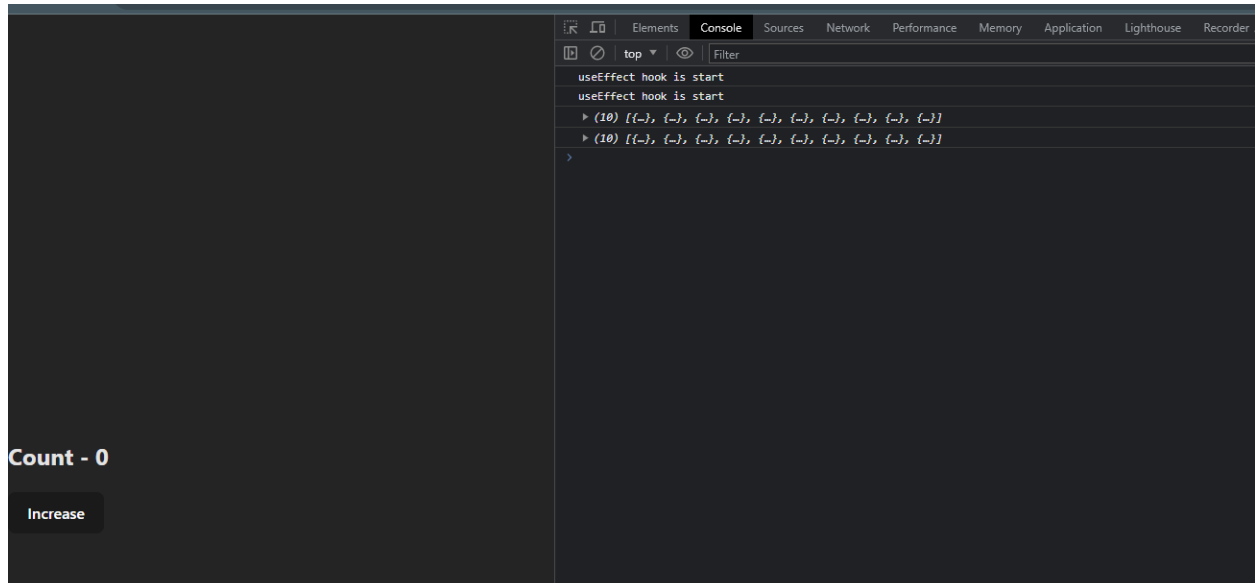
အဲ dependency array က ဘယ်လိုမျိုးအလုပ်လုပ်ပေးတာလည်းဆိုတော့ ကျတော်တို့ အပေါ်မှာရှင်းပြခဲ့သလိုမျိုး state တခု update လုပ်ပြီးသွားတဲ့အခါကျရင်သူက ပြန်စစ်ပေးတယ်ဗျ အဲကောင်က updated ဖြစ်နေရင် ထပ်အလုပ်မလုပ်သွားအောင် prevent အလုပ်လုပ်ပေးပါတယ် ပြီးတော့ သူက နောက်တခု အလုပ်လုပ်ပေးပါတယ် အဲဒါက ဘာလည်းဆိုတော့ ကျတော်တို့ အဲ array ထဲကို ပြောင်းလဲ လာနိုင်ချေရှိတဲ့ state တွေကို input လုပ်ပေးထားရင်လည်း သူက အဲ state update လုပ်တဲ့အချိန် တိုင်းမှာ ပြန်ပြီးတော့ callback ကို အလုပ်လုပ်စေပါတယ် အဲဒီအတွက် ကျတော်တို့က useEffect တခု ဖန်တီးတဲ့အချိန်မှာ အဲ array လေးကို မမေ့ခဲ့ကြဖို့ လိုအပ်ပါတယ်



ပုံ - infinite api request

```
useEffect(() => {  
  window.addEventListener("resize" , resizeHandler)  
  console.log("useEffect hook is start")  
  getData() // api side effect function  
  return() => {  
    window.removeEventListener("resize", resizeHandler)  
  }  
},[])
```

ပုံ - fixing with dependency array



ပုံ - result

ကျတော်တို့ အဲလိုမျိုး ရသွားတဲ့အချိန်မှာ ကျတော်တို့က button နှိပ်လို့ count က ၁ တိုးသွားရင် နောက်တကြိမ် ထပ် ပြီးတော့ api fetch လုပ်စေချင်တယ်ဆိုရင် ဒီလိုလေးရေးလိုက်ပါမယ်

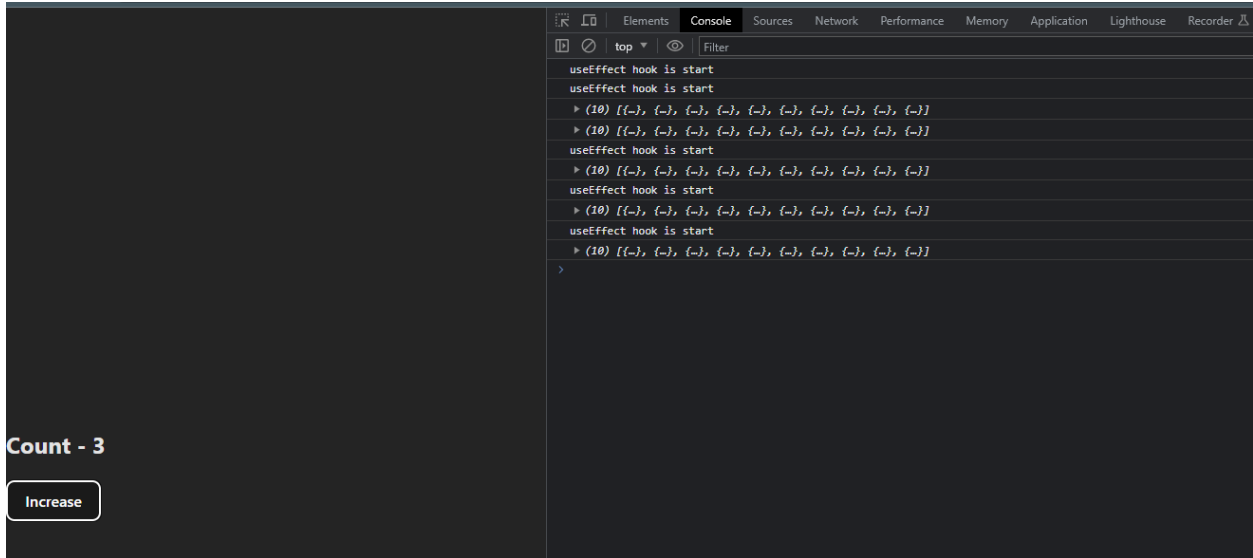
```
useEffect(() => {
  window.addEventListener("resize", resizeHandler)
  console.log("useEffect hook is start")
  getData() // api side effect function
  return () => {
    window.removeEventListener("resize", resizeHandler)
  }
}, [count])

return (
  <div>
    <h2>Count - {count}</h2>
    <button onClick={() => {
      setCount(prev => prev + 1)
    }}>Increase</button>
  </div>
)
```

ပုံ - adding count state in dependency array

အဲလိုလေး add လိုက်ပြီးတဲ့အခါကျရင် ကျတော်တို့ increase button လေးကို click လိုက်တိုင်းမှာ useEffect hook ကနေပြီးတော့ သူ့ရဲ့ callback ထဲမှာရှိတဲ့ api request function ကို အလုပ် လုပ်ပေးနေမှာပါ ဗျ

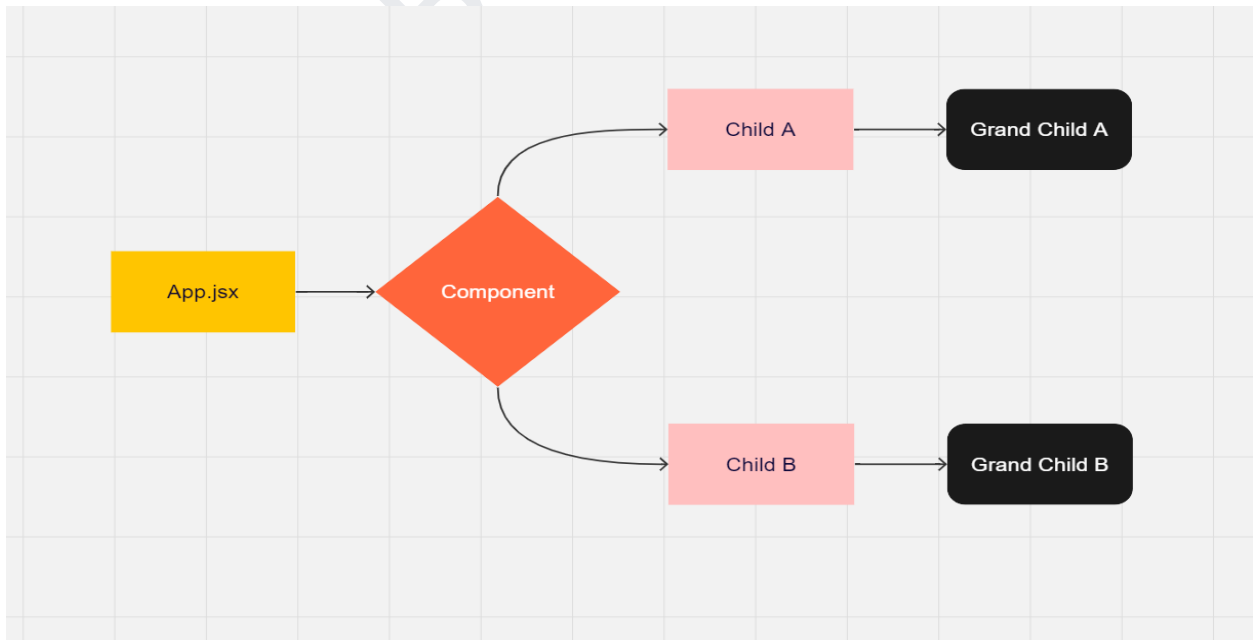
အဲလောက်ဆိုရင်တော့ ကျတော်တို့ useEffect ကို နားလည်သွားမယ်လို့ မျှော်လင့်ပါတယ်



ပုံ - result in browser

Context Api

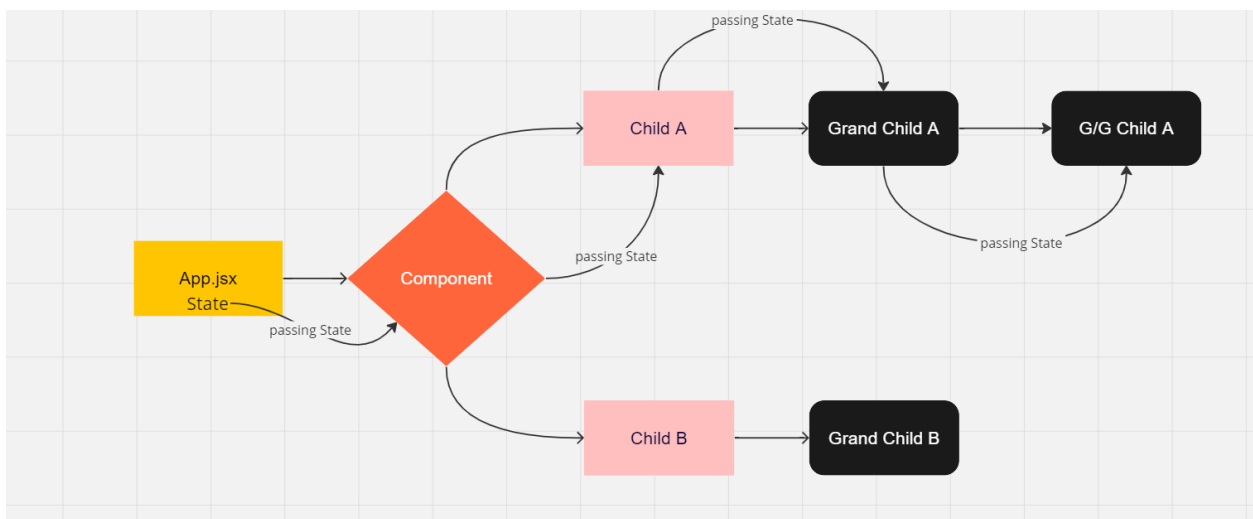
ကျွန်ုပ်တို့ လေ့လာလာလိုက်တာ အခုဆိုရင်တော်တော်လေးကို ခရီးရောက်လာပြီပေါ့ဗျ ကျွန်ုပ်တို့ ဒီထက်နည်းနည်းလေး ပိုပြီးတော့ အလုပ်လုပ်ပုံလေးတွေကို ဆက်လေ့လာလိုက်ကြရအောင် အဲဒါက ဘာလည်းဆိုတော့ context api လို့ ခေါ်တဲ့ (global state management) ပါပဲ အဲဒါကို ကျွန်ုပ်တို့က ဘာလို့ သုံးရတာလည်းဆိုတော့ ကျွန်ုပ်တို့ ပုံလေးတွေနဲ့ ရှင်းပြပေးပါမယ် ဦးဆုံး ဒီပုံလေးကိုအရင်ကြည့်လိုက်ကြရအောင်



React Learning

ပုံ - sample component tree

ဒီပုံမှာကျတော်တို့ ကြည့်လိုက်တဲ့အခါမှာ ဘာတွေရမှာလည်းဆိုတော့ APP component ထဲမှာ component တခု ရှိမှာပေါ့ ဗျ သူ့ ရဲ့ အထဲမှာ နောက်ထပ် child တွေရှိလာပြီးတော့ အဲဒီ child ကနေပြီးတော့ နောက်ထပ် grand child တွေ ခွဲလာတာကိုမြင်ရပါလိမ့်မယ် ကျတော်တို့ ကအရင်တုန်းက component တွေကို data pass လုပ်တုန်းက props နဲ့ပေးကြတယ် မလား အဲဒါက component က ၃ ဆင့်လောက်အထိ ဆိုရင် တော့ ဘာမှ တော့သိပ်မဖြစ်ဘူးဆိုပေမယ့် ကျတော်တို့ရေး တဲ့ application က ကြီးလာတာနဲ့အမျှ data တွေက တဆင့်ပြီး တဆင့် လက်ဆင့်ကမ်း passing လုပ်နေရတဲ့အခါကျရင် ဘာတွေဖြစ်လာတာလည်းဆိုတော့ ကျတော်တို့ application က performance ကျ လာတယ်ဗျ ပြီးတော့ သူ့ကို ဘယ်လိုခေါ်ကြတာလည်းဆိုတော့ props drilling လို့ ခေါ်တယ် ဗျ ဘာလို့ဆိုတော့ ကျတော်တို့ အခုလက်ရှိပြထားတဲ့ပုံအတိုင်းဆိုရင်



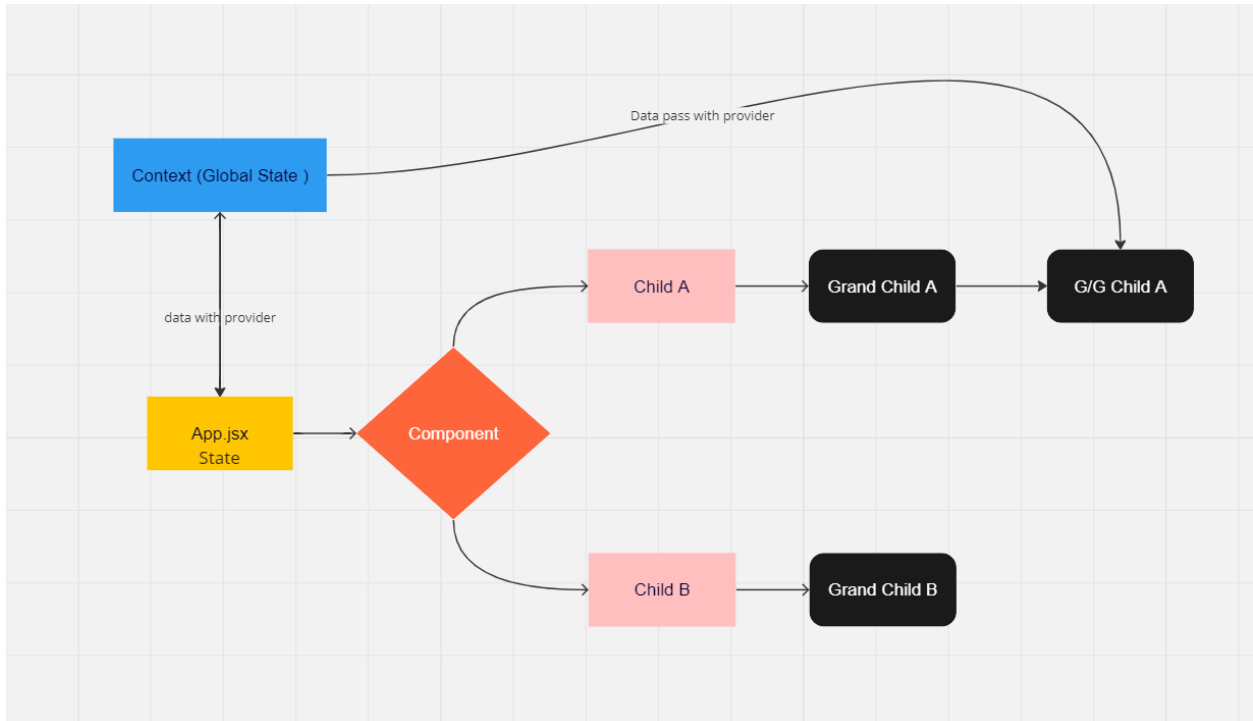
ပုံ - props drilling

ကျတော်တို့က data တခုကို ဒီလိုလေး ပေးလိုက်ပြီးတော့ သူတို့က ဟို အောက်ဆုံးက component ကနေပြီးတော့ အသုံးပြုချင်ရင် သူက အပေါ်က အဆင့်ဆင့် လက်ဆင့်ကမ်းလာတဲ့ ဟာတွေကို စောင့်နေရမှာပါ

အဲလိုမျိုးဖြစ်လာတဲ့အတွက် state တခုက ပြောင်းလဲပြီးတော့ အပေါ်ကို ပြန်တက်လာတဲ့ အခါကျရင် တော်တော် လေးကို အဆင်မပြေဖြစ်လာမှာပါ အဲလိုမျိုး မဖြစ်ရလေအောင် အတွက် ကျတော်တို့က ဘယ်လိုမျိုး အသုံးပြုရတာလည်းဆိုတော့ global state management တခု ဖြစ်တဲ့ context api ကိုအသုံးပြုပေးရပါတယ်

သူ့ကို အသုံးပြုလိုက်တဲ့အခါမှာ ခုနက လိုမျိုး state တွေ data တွေက component အဆင့်ဆင့် ဖြတ်ပြီးတော့ သွားစရာမလိုတော့ဘဲနဲ့ ကျတော်တို့ လိုအပ်တဲ့ component ကနေပြီးတော့ ခေါ်ယူသုံးလို့ရသွားပါတယ်

ဒီလောက်ဆိုရင် ကျတော်တို့ context api တခု အလုပ်လုပ်ပုံကို မြင်သွားမယ်လို့ထင်ပါတယ်



ပုံ - Context API

ကျတော်တို့ ဒီလောက်ဆိုရင် code မှာ ဘယ်လိုအသုံးပြုရလည်းဆိုတာကို

ဆက်လေ့လာလိုက်ကြရအောင်ဗျ

ကျတော်တို့ အဲဒီလေးကို သိသွားပြီဆိုရင် code လိုက်ကြရအောင်ဗျ

အရင်ဆုံး ကျတော်တို့ code မရေးခင်မှာ context တခု မှာ ဘယ်လိုမျိုး

အဆင့်တွေရှိဖို့လိုတာလည်းဆိုတာကို သိထားဖို့လိုတယ်ဗျ

ပထမဆုံး ကျတော်တို့က သူ့ကို အရင်ဆုံး create လုပ်ပေးရပါတယ်

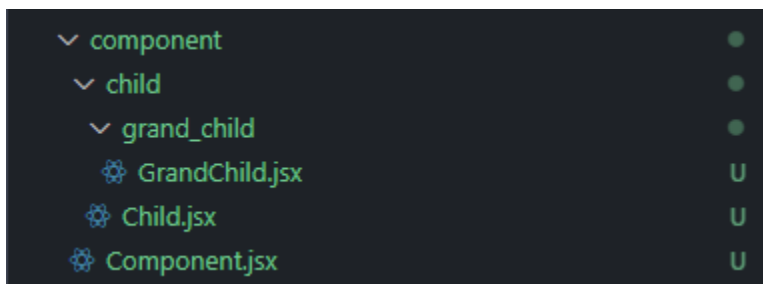
ဒုတိယ တဆင့်က ကျတော်တို့က create လုပ်လိုက်တဲ့ context ကို ဘယ် component

တွေကိုအသုံးပြုစေချင်တာလည်းဆိုပြီးတော့ provide လုပ်ပေးရပါတယ်

နောက်ဆုံး တဆင့်ကတော့ ကျတော်တို့က သူ့ကို ဘယ်လိုမျိုး ပြန်ပြီးတော့ consume (အသုံးပြု)

ရမှာလည်း ပေါ့ ဗျ

ကျတော်တို့ အရင်ဆုံး file လေးတွေကို ဒီလို လေး တည်ဆောက်လိုက်ပါမယ်



ပုံ - component tree

React Learning

အဲလိုလေး ရေးပြီးတော့ ကျတော်တို့ Component ထဲမှာ Child ကို import လုပ်ပြီးတော့ return လုပ်လိုက်ပါမယ်

ပြီးရင်တော့ ကျတော်တို့က Child ထဲမှာ GrandChildကို ထပ်ပြီးတော့ရေးလိုက်ပါမယ် ပြီးသွားရင် ကျတော်တို့က App component မှာ အသုံးပြုလိုက်ပါမယ်

```
import Child from './child/Child'

export default function Component() {
  return <Child />;
}
```

ပုံ - Component.jsx

```
import GrandChild from './grand_child/GrandChild'

export default function Child() {
  return <GrandChild />
}
```

ပုံ - Child.jsx

```
function GrandChild() {
  return (
    <div>GrandChild</div>
  )
}

export default GrandChild
```

ပုံ - GrandChild.jsx

ဒီလိုလေးရေးသွားပြီးရင် ကျတော်တို့ browser မှာ သွားကြည့်လိုက်ကြရအောင် ဗျ

GrandChild

ပုံ - browser result

အဲလိုလေးရေးလို့ပြီးသွားရင် ကျတော်တို့ context တစ်ခုကို create လုပ်ကြရအောင်ဗျ ဘယ်လိုမျိုးလုပ်ရမှာလည်းဆိုတော့

React Learning

ကျတော်တို့က အရင်ဆုံး react ကနေပြီးတော့ createContext ဆိုတာလေးကို import လုပ်ရပါမယ်

Create Context API

```
import { createContext } from "react"
```

ပုံ - import createContext

ဒီလိုလေး import လုပ်ပြီးသွားရင် ကျတော်တို့က context တခုကို စတင်ပြီးတော့ create လုပ်လို့ရသွားပါပြီ

နောက်ထပ် ဘယ်လိုရေးရမှာလည်းဆိုတော့

```
export const NameContext = createContext()
```

ပုံ - create context

ဒီမှာ ကျတော်တို့က တခုမေးစရာရှိသွားတယ်ဗျာ အဲဒါက ဘာလည်းဆိုတော့ ဘာလို့ export လုပ်တာလည်းပေါ့

ကျတော်တို့က ဒီလိုမျိုး export လုပ်ပေးလိုက်တော့မှ ကျတော်တို့က ကြိုက်တဲ့နေရာကနေပြီးတော့ သူ့ကိုအသုံးပြုလို့ရသွားစေချင်လို့ပါ

အဲဒါဆိုရင် ကျတော်တို့က ပထမ တဆင့်ဖြစ်တဲ့ context တခုကို create လုပ်လို့ပြီးသွားပါပြီ

Provide Context API and Pass data with Value props

အဲဒါပြီးသွားရင် ကျတော်တို့က ဘယ်လိုမျိုး provide လုပ်ပေးရမှာလည်းပေါ့

အဲဒါက ဘယ်လိုမျိုးလည်းဆိုတော့ ကျတော်တို့ ခုနက create လုပ်ခဲ့တဲ့ context ထဲမှာ provider ဆိုတာလေး ပါတယ် ဗျာ အဲဒါနဲ့ ဘယ်လိုမျိုးရေးရမှာလည်းဆိုတော့

```
/* <ComponentA /> */  
/* <UseEffectHook /> */  
<NameContext.Provider>  
  <Component />  
</NameContext.Provider>  
</>
```

ပုံ - wrapped with provider

အဲဒီလိုလေး ရေးပေးရမှာပါ

အဲလိုလေး ရေးလိုက်ပြီးသွားရင် ကျတော်တို့က ဘယ်လိုမျိုး data ကို ပေးရတာလည်းဆိုတော့ value ဆိုတဲ့ props လေးနဲ့ data ကို pass ပေးရတာပါ

ဘယ်လိုမျိုးရေးရတာလည်းဆိုတော့

React Learning

```
    { /* <UseEffectHook /> */  
    <NameContext.Provider value={"Bate Thar"}>  
      <Component />  
    </NameContext.Provider>  
  </>  
}
```

ပုံ - passing value

ဒီလိုလေးရေးပြီးတော့ ကျတော်တို့ data ကို passing ပေးလိုက်ပြီးသွားရင် ကျတော်တို့ က Context api ကို provide လုပ်တာပြီးသွားပြီပေါ့ဗျ

နောက်ဆုံး တဆင့်က ကျတော့ ကျတော်တို့ ဘယ်ကနေပြီးတော့ ပြန်အသုံးပြုမှာလည်းပေါ့

အဲဒါကို ကျတော်တို့က ဘယ်လိုမျိုး အသုံးပြုမှာလည်းဆိုတော့

Consume Context API

With Consumer method

ကျတော်တို့က consume လုပ်တဲ့အခါမှာ method က ၂မျိုးရှိတယ်ဗျ ပထမ တခုက တော့ ကျတော်တို့က callback function နဲ့ consume လုပ်တာပါ

ဘယ်လိုမျိုး လုပ်ရတာလည်းဆိုတော့

အရင်ဆုံး ကျတော်တို့က အသုံးပြုမယ့် component ထဲမှာ ခုနက app component ထဲ က export

လုပ်ခဲ့တဲ့ context ကို import လုပ်ပေးရမှာပါ

```
import { NameContext } from "../../App"
```

ပုံ - import context

အဲလိုမျိုး import လုပ်ပြီးရင် ကျတော်တို့ code လေးတွေကို ဒီလိုရေးလိုက်ပါမယ်

```
import { NameContext } from "../../App"  
  
function GrandChild() {  
  return (  
    <NameContext.Consumer>  
      {(name) => {  
        return (  
          <h1>{name}</h1>  
        )}}  
    </NameContext.Consumer>  
  )  
}  
  
export default GrandChild
```


React Learning

ပုံ - consuming

ဒီလိုလေးရေးပြီးသွားရင်ကျတော် code လေးတွေကို အရင်ဆုံးရှင်းပြပေးပါမယ်
အရင်ဆုံး ကျတော်တို့က import လုပ်ထားတဲ့ context ထဲမှာ ပါလာတဲ့ consumer
ကိုအသုံးပြုလိုက်ပါတယ်

သူတို့ရဲ့ child node နေရာမှာ ကျတော်တို့က callback function တခုကိုရေးပြီးတော့ return
ပြန်လိုက်ပါတယ်

ဒီအချိန်မှာ ကျတော်တို့က callback function ရဲ့ parameter နေရာမှာ app component ကနေပြီးတော့
ပေးလိုက်တဲ့ data တွေကို ကျတော်တို့ လက်ခံပြီးတော့ အသုံးပြုရပါတယ်

parameter လက်ခံတာကတော့ ကြိုက်တဲ့ နာမည်နဲ့ လက်ခံလို့ရပါတယ်

ကျတော်က တော့ name ကိုပေးလိုက်တာဖြစ်တော့ name နဲ့ လက်ခံလိုက်တာ ပါ

အဲလိုလက်ခံပြီးတော့ ကျတော်တို့က ပြန်အသုံးပြုလို့ရသွားပါပြီ ဗျာ ဒီလောက်ဆိုရင်တော့ ကျတော်တို့
နားလည်မယ်လို့ထင်ပါတယ်



Bate Thar

ပုံ - browser result

ကျတော်တို့က context ထဲမှာ နောက်ထပ် context တခုရှိ လာတဲ့အခါမှာ ရောဘယ်လိုမျိုး
အသုံးပြုလို့ရမလဲပေါ့ အဲလိုမျိုး ဆိုရင်ကျတော်တို့က ထုံးစံအတိုင်း ဟိုးအစက အဆင့်တွေအတိုင်း
တဆင့်စီပြန်လုပ်သွားရမှာပါ code လေးတွေကို တော့ ကျတော်ရေးပြထားလိုက်ပါမယ်
ပထမအဆင့်

```
export const NameContext = createContext();  
export const AgeContext = createContext();
```

ဒုတိယအဆင့်

React Learning

```
    { /* <UseEffectHook /> */ }
    <NameContext.Provider value={"Bate Thar"}>
      <AgeContext.Provider value={27}>
        <Component />
      </AgeContext.Provider>
    </NameContext.Provider>
  </>
```

Consuming အဆင့်

```
import { NameContext, AgeContext } from "../../App"

function GrandChild() {
  return (
    <NameContext.Consumer>
      {(name) => {
        return (
          <AgeContext.Consumer>
            {
              (age) => (<h1>My name is {name} and {age} years old</h1>)
            }
          </AgeContext.Consumer>
        )
      }}
    </NameContext.Consumer>
  )
}

export default GrandChild
```

Browser result

My name is Bate Thar and 27 years old

ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ context api တွေကို အသုံးပြုတတ်သွားမယ်လို့မျှော်လင့်ပါတယ်
အခုက ကျတော်က ဒီအတိုင်း number နဲ့ string တွေပဲ ပေးလိုက်တာ ပါ

React Learning

ကျတော်တို့က state တွေဖြစ်တဲ့ state နဲ့ setState တွေကိုရော provide လုပ်ပြီးတော့ data passing လုပ်ပေးလို့ရပါတယ် ဗျာ ဒီလောက်ဆိုရင်တော့ နားလည်သွားမယ်ထင်ပါတယ်

With useContext Hook

ကျတော်တို့ ခုနက consumer ဆိုတဲ့ node နဲ့ context api တွေကို အသုံးပြုပြီးတော့ data တွေကို လက်ခံလိုက်ကြတယ်မလား ဒီတစ်ခါ ကျတော့ ကျတော်တို့က useContext ဆိုတဲ့ ဟာလေးကိုအသုံးပြုပြီးတော့ data တွေကို အသုံးပြုကြပါမယ်

ဘာလို့ ကျတော်က consume လုပ်တာကိုပဲပြောရတာလည်းဆိုတော့ အပေါ်က အဆင့်တွေက အကုန်အတူတူ တွေပါပဲ အဓိက ကွာသွားတဲ့ အရာက ကျတော်တို့ consume လုပ်တဲ့နေရာမှာပဲ ကွာသွားတာပါ

Hook ကိုအသုံးပြုလိုက်တဲ့အတွက် ကျတော်တို့အတွက် က တော်တော်လေးကို လွယ်ကူသွားပါတယ် ကဲ ကျတော်တို့ စလိုက်ကြရအောင်ဗျာ

အရင်ဆုံး ကျတော်တို့ useContext ကို import လုပ်လိုက်ပါမယ်

```
import { useContext } from "react"
```

ပုံ - import useContext hook

Import လုပ်ပြီးသွားတဲ့အခါကျရင် ကျတော်တို့က ခုနက context တွေကို consume လုပ်ပါမယ် ဘယ်လိုမျိုးလုပ်ရမှာလည်းဆိုတော့

```
import { useContext } from "react"
import { NameContext, AgeContext } from "../App"

function GrandChild() {
  const name = useContext(NameContext)
  const age = useContext(AgeContext)
  return (
    <h1>My name is {name} and {age} years old</h1>
  )
}

export default GrandChild
```

ပုံ - consuming with useContext

ကျတော်တို့က useContext hook ရဲ့ parenthesis ထဲမှာ ခုနက import လုပ်ခဲ့တဲ့ context တွေကို ထည့်ပေးလိုက်တဲ့အခါကျရင်ကျတော်တို့ ခုနက provide လုပ်ခဲ့တဲ့ value တွေကို သူက return ပြန်ပေးလာပါတယ် အဲဒါကို ကျတော်တို့က variable တွေနဲ့ သိမ်းပြီးတော့ အဲဒါကို render လုပ်ပေးလိုက်ရုံပါပဲ ဗျာ

React Learning

ဒီလောက်ဆိုရင် ကျတော်တို့ context api တွေကိုအသုံးပြုသွားနိုင်မယ်လို့ထင်ပါတယ်

useReducer Hook

ကျတော်တို့ အခုလေ့လာမှာကတော့ useReducer hook အကြောင်းပါပဲ
ကျတော်တို့က react ကို လေ့လာတာက တော်တော်လေးခရီးရောက်ခဲ့ကြပါပြီ
အဲဒီအတွက် ကျတော်တို့က အရေးကြီးတဲ့ အရာတွေကို ဆက်လေ့လာကြရတော့မှာပါ useReducer
hook ဆိုတာက ကျတော်တို့ မြင်သာအောင် အလွယ်ပြောရရင် state management အတွက် ပါပဲ
ကျတော်တို့က နောက်ပိုင်း state တွေက များလာတာနဲ့အမျှ complex တွေဖြစ်လာတဲ့အခါကျရင်
ကျတော်တို့ရဲ့ code တွေက ရှုပ်ပွ လာတော့မှာပါ အဲဒါကို ကျတော်တို့က reducer hook
ကိုအသုံးပြုပြီးတော့ ကုဒ်တွေကိုရှင်းလင်းသွားအောင် ရေးလို့ရပါတယ်
သူ့ရဲ့အလုပ်လုပ်ပုံလေးက တော့ ဘယ်လိုမျိုးလည်းဆိုတော့ ကျတော်တို့ရဲ့ state တွေကို action အပေါ်
မူတည်ပြီးတော့ ပြောင်းလဲ အလုပ်လုပ်သွားတာမျိုးပါ
ကျတော်တို့ စပြီးတော့ လေ့လာလိုက်ကြရအောင်ဗျ
အရင်ဆုံး ဒီလိုလေး ရေးလိုက်ပါမယ်

```
import { useReducer } from 'react'

function ReducerHook() {
  return (
    <div>ReducerHook</div>
  )
}

export default ReducerHook
```

ပုံ - reducer hook

ကျတော်တို့ component တစ်ခုကိုအရင် တည်ဆောက်လိုက်ပါမယ်

ပြီးသွားတဲ့အခါကျရင်ကျတော်တို့က reducer hook ကို ဒီလိုလေး အသုံးပြုလိုက်ပါမယ်

```
import { useReducer } from 'react'

function ReducerHook() {
  const [state, dispatch] = useReducer(first /* this is callback function */ , second /* this is initial state */ , third /* this is optional */ )
  return (
    <div>
      ReducerHook
    </div>
  )
}

export default ReducerHook
```

ပုံ - reducer hook

ကျတော် ဒီကုဒ်လေးတွေကိုတချက်ရှင်းပြပေးပါမယ်

React Learning

ပထမဆုံး ကျတော်တို့က `useReducer` လေးကို ဒီလိုလေးရေးလိုက်ပါမယ်

`useReducer(first , second , third)`

ပြောချင်တာက reducer hook တခုမှာ parameter ကို ၃ခု လက်ခံပါတယ်

အဲဒီ သုံးခုမှာက ကျတော်တို့အသုံးများတာက first နဲ့ second ပါပဲ third တခုက ကျတော့ optional ဆိုတော့ သူ့ကို ကျတော်တို့ နောက်မှ ဆက်လေ့လာလိုက်ကြရအောင်

```
}  
const [count, dispatch] = useReducer(reducer, initailCount)
```

အဲဒီ ၂ခု ကဘာတွေလည်းဆိုတော့ အပေါ်က ကုဒ်မှာ ရေးပြထားတဲ့အတိုင်းပါပဲ reducer ဆိုတဲ့ callback function တခုနဲ့ initial state တစ်ခုပါပဲ

သူက ဒီလို parameter ၂ခုကို တောင်းပြီးတော့ ဘာကို return ပြန်ပေးတာလည်းဆိုတော့ ကျတော်တို့ကို update ဖြစ်လာတဲ့ state နဲ့ dispatch ဆိုတဲ့ reducer callback ကို action လုပ်ပေးတဲ့ method လေးတခု ကို ပြန်ပေးပါတယ်

reducer ဆိုတဲ့ callback function လေးက ဘယ်လိုမျိုးအလုပ်လုပ်ပေးတာလည်းဆိုတော့ `useReducer` ဆိုတဲ့ hook မှာ ဝင်လာတဲ့ initial state လေးကို လက်ခံပြီးတော့ သူ့ရဲ့ action ထဲကို dispatch ကနေပြီး တော့ ဝင်လာမယ့် action type အပေါ်မူတည်ပြီးတော့ state ကို return ပြန်ပေးတာပါ

အရင်ဆုံး ကျတော်တို့ Counter app လေးတခုကို ဒီလိုလေး ရေးလိုက်ပါမယ်
ဘယ်လိုမျိုးလည်းဆိုတော့

```
import { useReducer } from 'react'  
  
function ReducerHook() {  
  const initailCount = 0;  
  const reducer = (state ,action) => {  
    return state  
  }  
  const [count, dispatch] = useReducer(reducer, initailCount)  
  return (  
    <div>  
      <h1>Count - {count}</h1>  
      <button onClick={() => {}}>Increase 1</button>  
    </div>  
  )  
}  
  
export default ReducerHook
```

ပုံ - Counter App with Reducer Hook

ကျတော်တို့က ပထမဦးဆုံး ဘယ်လိုမျိုး state တွေကို ပြောင်းလဲမှာလည်းဆိုတော့ အဓိက ကျတာက dispatch နဲ့ action တွေပါပဲ အဲဒါကို ကျတော်တို့က ဘယ်လိုမျိုး ရေးရမှာလည်းဆိုတော့

React Learning

ဒီလိုလေးရေးလိုက်ပါမယ်

```
function ReducerHook() {
  const initailCount = 0;
  const reducer = (state ,action) => {
    switch(action){
      case "INCREASE": {
        return state + 1
      }
      default : {
        return state
      }
    }
  }
  const [count, dispatch] = useReducer(reducer, initailCount)
  return (
    <div>
      <h1>Count - {count}</h1>
      <button onClick={() => {
        dispatch("INCREASE")
      }}>Increase 1</button>
    </div>
  )
}

export default ReducerHook
```

ပုံ - using dispatch

ကျတော်တို့က dispatch ရဲ့ parameter ထဲကို ကျတော်တို့က ပြောင်းလဲမှုလုပ်စေချင်တဲ့ action လေးကို သတ်မှတ် ပေးထားလိုက်ပါတယ်

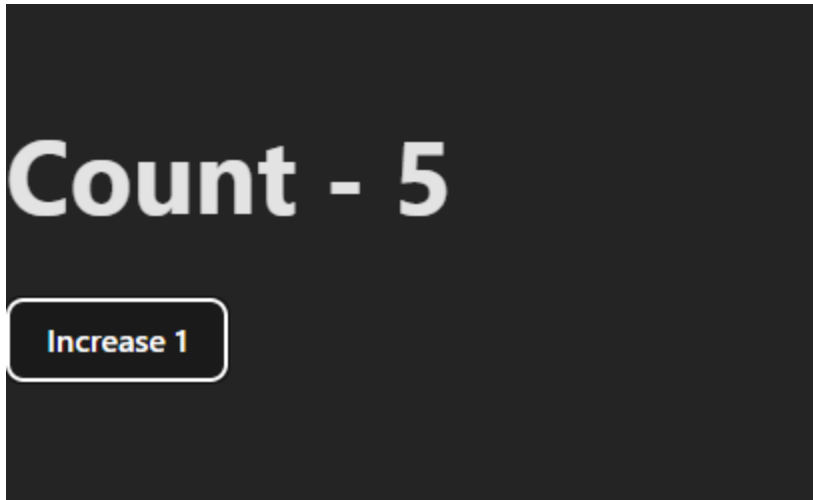
အဲဒါကို ကျတော်တို့ရဲ့ reducer callback ထဲမှာ ဘယ်လိုမျိုး စောင့်ကြည့်ထားတာလည်းဆိုတော့

ကျတော်တို့က switch ဆိုတဲ့ condition စစ်တာလေးနဲ့ စောင့်ကြည့်ထားပါတယ် အဲဒါကို

ကုဒ်လေးကိုကြည့်လိုက်ရင် မြင်သာပါလိမ့်မယ် ဒီနေရာမှာ if else နဲ့သုံးလို့မရဘူးလားလို့

မေးချင်လာပါလိမ့်မယ် ရပါတယ် ကြိုက်တာနဲ့သုံးလို့ရပါတယ်

သူ့ကိုအလုပ်လုပ်တာကို browser မှာသွားကြည့်ရင် ဒီလိုလေး မြင်ရပါလိမ့်မယ်



ပုံ - browser result

ဒီလိုလေး ကျတော်တို့က state ကို လိုချင်တဲ့ action လေးတွေ သတ်မှတ်ပြီးတော့ လုပ်ခိုင်းတာဖြစ်တဲ့အတွက် ကျတော်တို့ရဲ့ state တွေကို management လုပ်တဲ့အခါမှာ တော်တော်လေး ရှင်းလင်းသွားတာကိုမြင်ရပါလိမ့်မယ်

ကျတော်တို့က action ကိုပေးတဲ့အခါမှာ အခုက ကျတော်က string type အနေနဲ့ ပဲ ပေးလိုက်တာပါ တကယ်က ကျတော်တို့က နောက်ထပ် object အနေနဲ့ သတ်မှတ်ပြီးတော့ပေးလို့ရပါတယ် ဘယ်လိုမျိုးလည်းဆိုတော့

```
{ type: "type", payload: "payload" }
```

ဒီလိုလေးပေးပြီးတော့ ကျတော်တို့ ကုဒ်ကို ဒီလိုလေးပြင်လိုက်ပါမယ် ဗျ

React Learning

```
* ReducerHook.jsx > ...
import { useReducer } from 'react'

function ReducerHook() {
  const initailCount = 0;
  const reducer = (state ,action) => {
    switch(action.type){
      case "INCREASE": {
        return state + action.payload
      }
      default : {
        return state
      }
    }
  }
  const [count, dispatch] = useReducer(reducer, initailCount)
  return (
    <div>
      <h1>Count - {count}</h1>
      <button onClick={() => {
        dispatch({type: "INCREASE" , payload:1})
      }}>Increase 1</button>
      <button onClick={() => {
        dispatch({type: "INCREASE", payload: 5})
      }}>Increase By Amount</button>
    </div>
  )
}

export default ReducerHook
```

ပုံ - code update

ဒီလိုလေးရေးလိုက်ရုံပဲဆိုတော့ ကျတော်တို့ရဲ့ code တွေက တော်တော်လေး
ရှင်းသွားတာကိုမြင်သွားမယ်လို့ထင်ပါတယ်

React Styling

ကျတော်တို့ react ကို လေ့လာနေခဲ့တာ တော်တော်လေး ခရီးရောက်ပြီတော့ တော်တော်လေးလည်း
ရေးနိုင်သွားမယ်လို့ ထင်ပါတယ် ကျတော်တို့ beginner တွေအတွက်ကျန်ခဲ့တာလေးတွေကို
ဆက်ပြီးတော့ လေ့လာလိုက်ကြရအောင်အဲဒါက ဘာတွေလည်းဆိုတော့ ကျတော်တို့က component
styling တွေ ပါပဲ

React Learning

ကျတော်တို့က Frontend Web development ကိုလေ့လာနေကြတာဆိုတော့ function တွေ အလုပ်လုပ်ပုံတွေကို သိနေရုံနဲ့တော့ မပြီးသွားဘူး ဗျ၊ အဲအတွက် ကျတော်တို့က styling တွေကို လည်းရေးတတ်ဖို့ လိုပါတယ်

ကျတော်တို့ react မှာ style ရေးပုံရေးနည်းက ၃နည်းရှိတယ်လို့ပြောလို့ရပါတယ်
အဲဒါက ဘာတွေလည်းဆိုတော့

Inline styling

External CSS styling နဲ့

Module styling တွေပါပဲ

Inline Styling

အရင်ဆုံး ကျတော်တို့ inline styling ကို လေ့လာလိုက်ကြရအောင်

အရင်ဆုံး ကျတော်တို့ component တခုကို ဖန်တီးလိုက်ပါမယ်

```
import React from 'react'

function Styling() {
  return (
    <div>
      <h1>Inline Styling</h1>
    </div>
  )
}

export default Styling
```

ပြီရင်တော့ ကျတော်တို့က style ဆိုတဲ့ props လေး ရေးလိုက်ပါမယ်

ဒီအချိန်မှာ ကျတော်တို့ သတိထားရမှာက html css လို့မျိုး string type နဲ့ရေးလို့ မရတာကိုပါပဲ

သူ့မှာက object type ပုံစံနဲ့ရေးပေးရမှာပါ

```
<div>
  <h1 style={{margin: "auto", backgroundColor: "#007bff", color: "#fff"}}>Inline Styling</h1>
</div>
```

ဒီလိုလေး ရေးပေးရမှာပါ

ကျတော်တို့ html css တွေမှာရှိတဲ့ dash (-) တွေအစား camelCase နဲ့ရေးပေးရပါမယ်

ဥပမာ - background-color >> backgroundColor အဲလိုလေး ပြောင်းရေးပေးရပါမယ်

ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ inline stylingကို ရေးတတ်သွားမယ်လို့မြင်ပါတယ်

External CSS Styling

အဲဒါကတော့ ကျတော်တို့ သိပ်ပြီးတော့ ထွေထွေထူးထူးတော့မဟုတ်ပါဘူး html css လို့မျိုးပါပဲ

အဲဒါကိုကျတော်တို့က import လုပ်ပြီးတော့ အသုံးပြုရပါပဲ

React Learning

```
<h1 style={{margin: auto, backgroundColor: '#007bff'}}>  
<h1 className='external'>External CSS Styling</h1>  
</div>
```

ဒီလိုလေးရေးပြီးတော့

ဒီလိုလေး import လုပ်လိုက်ရုံပါပဲ

```
import React from 'react'  
import './style.css'
```

Module Styling

ကျတော်တို့ styling တွေထဲမှာ နောက်ဆုံးတခု ဖြစ်တဲ့ module styling

နည်းကိုလေ့လာလိုက်ကြရအောင်ဗျ

အဲနည်းက တော်တော်လေး အသုံးများကြပါတယ် ဘာလို့ဆိုတော့ သူက ကျတော်တို့ ရဲ့ component တခုထဲမှာ import လုပ်ပြီးတော့ သူ့ကိုအသုံးပြုလိုက်တဲ့ နေရာကလွဲပြီးတော့ တခြား styling တွေ မှာ သက်ရောက်မှုမရှိလို့ပါ ပဲ

အဲအတွက်ကြောင့် သူ့ကို တော်တော်လေး အသုံးပြုကြပါတယ်

သူ့ကိုရေးတဲ့အခါကျတော့ ဘယ်လိုမျိုးရေးရမှာလည်းဆိုတော့

```
<h1 className={styles.moduleStyling}>Module Styling</h1>  
</div>
```

ဒီလိုလေး object.key ပုံစံနဲ့ရေးရပါတယ်

အဲအရှေ့က style ဆိုတဲ့ ဟာက ဘယ်ကနေပြီးလာတာလည်းဆိုတော့

ကျတော်တို့က

```
import './style.css'  
import styles from './style.module.css'
```

ဒီလိုလေး import လုပ်ထားတာပါ

style ဆိုတဲ့ နေရာမှာ ကြိုက်တာထည့်လို့ရပါတယ် ကျတော်ကတော့ styles ကို

သုံးတာအကျင့်ဖြစ်နေလို့ပါ

ပြီးတော့ နောက်ထပ်တခု ထူးခြားတာကိုတွေ့ရပါလိမ့်မယ် အဲဒါက ဘာလည်းဆိုတော့ css file name မှာ module ဆိုတာလေးပါနေတာပါ

ဟုတ်ပါတယ် ကျတော်တို့ file name ကို ဒီလိုလေး ပေးထားလိုက်တော့ ဘယ်ဟာက module file

ဆိုပြီးတော့ သိသာသွားပါတယ် ဗျ

ဒီလောက်ဆိုရင်တော့ တော်တော်လေး နားလည်သွားမယ်လို့ထင်ပါတယ်

Css code တွေက ကျတော်တို့ အရင် css ရေးထုံးရေးနည်းအတိုင်းပဲမလို့ ကျတော် မပြတော့ပါဘူး ဗျ

ဒီလောက်ဆိုရင် ကျတော်တို့ styling အကြောင်းကို နားလည်သွားမယ်လို့ထင်ပါတယ်

Inline Styling

External CSS Styling

Module Styling

ပုံ - result

React Router

ကျတော်တို့ အခုဆိုရင် react ကို styling တွေကိုရောလေ့လာလိုပြီးသွားပြီဆိုတော့ နောက်ထပ်

ကျန်နေသေးတဲ့ အရာလေးတွေကို ဆက်ပြီးတော့ လေ့လာသွားကြရအောင်

အဲဒါက ဘာလည်းဆိုတော့ react router ပါပဲ

အဲဒါက ဘာဖြစ်လို့ လိုအပ်တာလည်းဆိုတော့ ကျတော်တို့ရဲ့ react က SPA ဖြစ်တဲ့ အတွက် ကျတော်တို့

ရဲ့ application တစ်ခုက html page တွေများစွာရှိတာမဟုတ်တဲ့အတွက် page တခုနဲ့တခု

အကူးအပြောင်း ကိုလုပ်စေဖို့အတွက် third-party library ဖြစ်တဲ့ react router ကိုအသုံးပြုရပါတယ်

သူက ဘယ်လိုမျိုးအလုပ်လုပ်ပေးတာလည်းဆိုတော့ ကျတော်တို့ web application တခုရဲ့ page တွေက

url (route) အပေါ်မူတည်ပြီးတော့ သူက page (component) ကို ပြောင်းလဲပေးတာ ပါ

ကျတော်တို့ လေ့လာလိုက်ကြရအောင်

အရင်ဆုံးမလေ့လာခင်မှာ ကျတော်တို့ terminal ကို ဖွင့်ပြီးတော့ ဒီကုဒ်လေးကို run လိုက်ပါမယ်

ဘာလို့လည်းဆိုတော့ ကျတော်တို့ရဲ့ router ကို setup လုပ်တဲ့အခါမှာ ကျတော်တို့ရဲ့ application တခုလုံးကို သူက

မှာ build in ပါလာတာမဟုတ်ပါဘူး ဗျဲ အဲအတွက်ကြောင့်သူ့ကို install လုပ်ပေးရမှာပါ

`npm i react - router - dom`

ဒီလိုလေး run ပြီးသွားရင် ကျတော်တို့ setup လုပ်လိုက်ရအောင်

အရင်ဆုံး ကျတော်တို့ရဲ့ router ကို setup လုပ်တဲ့အခါမှာ ကျတော်တို့ရဲ့ application တခုလုံးကို သူက

အလုပ်လုပ်စေဖို့အတွက် ကျတော်တို့ application ရဲ့ root node ကနေပြီးတော့ provide လုပ်လိုက်ပါမယ်

ဘယ်လိုမျိုးလုပ်မှာလည်းဆိုတော့ ကျတော်တို့ main.jsx ထဲကိုဝင်ပြီးတော့

ဒီလိုလေးရေးလိုက်ပါမယ်

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'
import { BrowserRouter } from 'react-router-dom'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>,
)
```

ကျတော်တို့ရဲ့ react ကနေပြီးတော့ သူ့ရဲ့ application မှာရှိတဲ့ url (address) bar ကို ဒီအတိုင်းဆိုင်ရင်မသိပါဘူး

အဲအတွက် ကျတော်တို့ရဲ့ react ကနေပြီးတော့ သိသွားစေဖို့အတွက် BrowserRouter လေးကို react-router-dom ကနေပြီးတော့ import လုပ်ပြီးတော့ app component တခုလုံးကို အုပ်ပေးလိုက်ပါတယ်

အဲအချိန်မှာ ကျတော်တို့ရဲ့ react application တစ်ခုက url (address) bar ကိုသိပြီးတော့အလုပ်လုပ်သွားပါတယ်

အခုက ကျတော်တို့ ရဲ့ application က url ကိုသိသွားပြီးတော့အလုပ်လုပ်သွားပြီဆိုတော့ ကျတော်တို့ သူ့ကို ဘယ်လိုမျိုး အသုံးပြုရမလဲဆိုတာကို ဆက်လေ့လာသွားကြရအောင်ဗျ
အရင်ဆုံး ကျတော်တို့ app.jsx ထဲကိုသွားပြီးတော့ ဒီလိုလေး ရေးလိုက်ကြရအောင်

```
import { Routes , Route } from 'react-router-dom'
```

အရင်ဆုံး ကျတော်တို့က Routes နဲ့ Route ကို react-router-dom ကနေပြီးတော့ import လုပ်လိုက်ပါမယ်

ဘာလို့ လည်းဆိုတော့ ကျတော်တို့ရဲ့ application တခုက root node တခုကိုပဲ သိပါတယ်

အဲအတိုင်းပါပဲ react-router-dom ကလည်း Routes ဆိုတဲ့ root node တစ်ခုကိုပဲ သိပါတယ်

သူ့ရဲ့အထဲမှာမှ ကျတော်တို့ ပြောင်းလဲသွားစေချင်တဲ့ route တွေ page တွေကို ရေးပေးရပါတယ် ကျတော်တို့ ကုန်လိုက်ကြရအောင်

```
<Routes>
  <Route path="/" element={Styling}></Route>
</Routes>
```

ဒီကုန်လေးတွေကို ကျတော် တချက်ရှင်းပြပေးပါမယ်

React Learning

အပေါ်မှာပြောထားသလိုပဲ Routes ရဲ့အထဲမှာ Route ကိုရေးပေးရပါတယ် သူ့ရဲ့အထဲမှာ ကျတော်တို့ က path နဲ့ element ကိုရေးထားတာကိုမြင်ရပါလိမ့်မယ်

Path ဆိုတာက ကျတော်တို့ browser address bar (url bar) ထဲမှာ ပေးလိုက်တဲ့ route တွေကိုပြောတာပါ

“/” ဆိုတာက ကျတော်တို့ က ဘာ route မှ မရှိတဲ့ နေရာကိုပြောတာပါ အလွယ်ပြောရရင် home pageပေါ့အဲလို သဘောမျိုး နဲ့ ပြောတာပါ

သူ့ရဲ့အနောက်မှာ ပါတဲ့ element ဆိုတာက အရှေ့ရဲ့ path ဆိုတဲ့ route ကိုရောက်တဲ့အချိန်မှာ render လုပ်ပေးရမယ့် component (page) ကို ထည့်ပေးရပါတယ်

အဲလိုလေး ထည့်ပေးလိုက်တော့ သူက သူနဲ့သက်ဆိုင်တဲ့ route ကိုရောက်တဲ့အခါမှာ

ကျတော်တို့ရေးထားတဲ့ page ကို render လုပ်ပေးသွားတာပါ ဒီလောက်ဆိုရင်တော့ တော်တော်လေး နားလည်သွားမယ်လို့ထင်ပါတယ်

Additional React-Router-Dom

ကျတော်တို့ react-router ကို လေ့လာတဲ့အခါမှာ ကျန်တဲ့ဟာလေးတွေအထဲမှာမှ အသုံးဝင်တဲ့ အရာလေးကို

ကျတော်တို့ လေ့လာဖို့လည်းလိုပါတယ်

အဲဒါက ဘာလည်းဆိုတော့ router ကို ချိန်းတဲ့အခါမှာ လိုအပ်တဲ့ tip လေးနဲ့ သူ့ရဲ့ hook တစ်ခုဖြစ်တဲ့ useNavigate hook လေးပါပဲ

အဲဒါကိုလေ့လာသွားကြရအောင်ဗျ

အရင်ဆုံး ကျတော်တို့ ဒီလိုလေးရေးလိုက်ပါမယ်

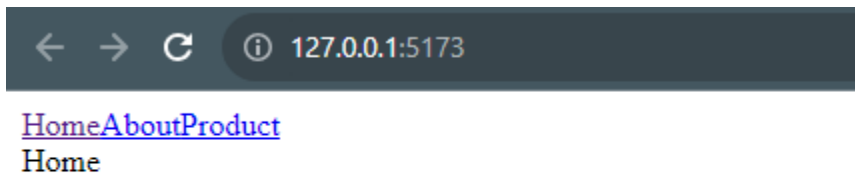
```
<a href="/">Home</a>
<a href="/about">About</a>
<a href="/product">Product</a>
<Routes>
  <Route path="/" element={<Styling />}></Route>
</Routes>
```

ကျတော်တို့ html မှာဆိုရင် page တစ်ခုနဲ့ တစ်ခု အကူးအပြောင်းကို သွားချင်တဲ့အခါမှာ သုံးတဲ့ a tag လေးကို အသုံးပြုပြီးတော့ route တွေကို ပြောင်းလို့ရပါတယ်ဗျ

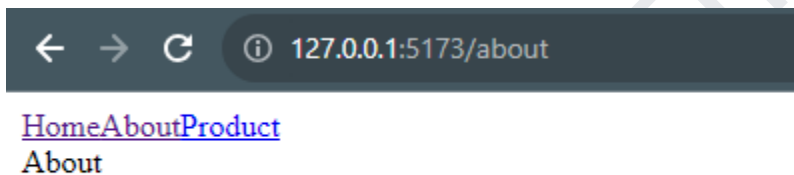
href ထဲမှာ က ကျတော့ ကျတော်တို့ သွားစေချင်တဲ့ route page ကို ရေးထားပြီးတော့ သူ့ကို route ထဲမှာ ဒီလိုလေးရေးလိုက်ပါမယ် ဗျ

```
<Routes>
  <Route path="/" element={<Home />}></Route>
  <Route path="/about" element={<About />}></Route>
  <Route path="/product" element={<Product />}></Route>
</Routes>
```

ဒီလိုလေးရေးလိုက်ပြီးတော့ ကျတော်တို့ browser မှာသွားကြည့်တဲ့အခါမှာ ဒီလိုလေး မြင်ရပါမယ်



ဒီလိုလေး မြင်ရတဲ့အခါမှာ ကျတော်တို့ အပေါ်က address bar ထဲမှာ ရှိတဲ့ localhost:5173 ရဲ့အနောက်မှာ ဘာမှ မရှိတာကို မြင်ရပါလိမ့်မယ်အဲဒါက "/" ဒီဟာကိုပြောချင်တာပါ နောက်ထပ် route ကိုသွားဖို့အတွက် ကျတော် about လေးကိုနှိပ်လိုက်ပါမယ်



ဒီလိုလေး ဖြစ်သွားတာကိုမြင်ရပါမယ်
ဒီလောက်ဆိုရင်ကျတော်တို့ တော်တော်လေး နားလည်သွားမယ်လို့ထင်ပါတယ်
အဲဒီတော့ ကျတော်တို့ နောက်ထပ် ဟာလေးတွေကို ဆက်လေ့လာသွားကြရအောင်
အဲဒါက ဘာလည်းဆိုတော့ NavLink နဲ့ Link ဆိုတဲ့ဟာလေးတွေပါပဲ
သူတို့က ဘယ်လိုမျိုးအလုပ်လုပ်ပေးတာလည်းဆိုတာကို ကြည့်လိုက်ကြရအောင်

```
import {Routes , Route, Link , NavLink} from 'react-router-dom'
```

အရင်ဆုံး ကျတော်တို့ react-router-dom ကနေပြီးတော့ သူတို့ကို import လုပ်လိုက်ပါမယ်
ပြီးသွားရင် ကျတော်တို့က ခုနက a tag တွေအစား ဒီလိုလေးပြောင်းလဲပြီးတော့ရေးလိုက်ပါမယ်

```
<Link to="/">Home</Link>
<Link to="/about">About</Link>
<Link to="/product">Product</Link>
```

React Learning

ဒီလိုလေးရေးပြီးတော့ browser မှာသွားကြည့်လိုက်ရင် ကျတော်တို့ ခုနက အတိုင်းလေး မြင်ရပါလိမ့်မယ် ဗျ

သူ့ရဲ့အလုပ်လုပ်ပုံက a tag နဲ့ သိပ်မကွာသွားပါဘူး a tag ရဲ့ href အစား to ဆိုပြီးတော့ ပြောင်းလဲ သုံးပေးရုံပါပဲ

နောက်တစ်ခုဖြစ်တဲ့ NavLink ကို ဆက်ကြည့်လိုက်ကြရအောင်ဗျ

```
/* <Styling></Styling> */  
<NavLink to="/">Home</NavLink>  
<NavLink to="/about">About</NavLink>  
<NavLink to="/product">Product</NavLink>
```

ဒီလိုလေးရေးပြီးတော့ ကျတော်တို့ browser မှာသွားကြည့်တဲ့အခါမှာ ခုနက လိုပဲ အတူတူပဲ သွားပါတယ် ဒါပေမယ့် သူ့မှာက ထူးခြားတာ တခုက ဘာလည်းဆိုတော့

ကျတော်တို့က လက်ရှိရောက်နေတဲ့ route ရဲ့ nav link (a tag) ထဲကို class 'active' လေး ကို ထည့်ပေးသွားတာကို မြင်ရပါလိမ့်မယ်

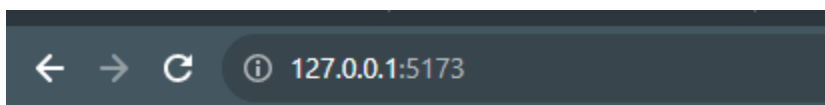
```
<div id="root" >==>  
  <a aria-current="page" class="active" href="/">Home</a>  
  <a class href="/about">About</a>  
  <a class href="/product">Product</a>  
</div>Home</div>
```

Inspect ထောက်ပြီး ကြည့်ရင် ဒီလိုလေး မြင်ရပါလိမ့်မယ် အဲလိုမျိုး မြင်ရတော့ ကျတော်တို့က nav link ကို active ဖြစ်အောင် ပြဖို့အတွက်က လွယ်ကူသွားပါတယ်

သူ့ရဲ့ class active ကို ကျတော်တို့ index.css မှာ လိုအပ်တဲ့ css code လေးတွေ

သွားရေးလိုက်တဲ့အခါမှာ ကျတော်တို့အတွက် UX (user experience) ပိုင်းမှာ တော်တော်လေး အဆင်ပြေသွားစေတယ်ဗျ

```
.active {  
  background-color: black;  
  color: #fff;  
}
```



HomeAboutProduct
Home

React Learning

ဒီလိုလေး မြင်ရပါလိမ့်မယ်ဗျ

ဒီလောက်ဆိုရင် ကျတော်တို့ react-router ကို တော်တော်လေး နားလည်သွားမယ်လို့မြင်ပါတယ်

useNavigate Hook

ကျတော်တို့ react router ရဲ့ hook တစ်ခုဖြစ်တဲ့ useNavigate ကလည်းတော်တော်လေး

အသုံးဝင်ပါတယ်

သူ့ကိုကျတော်တို့က ဘယ်လိုမျိုးအသုံးပြုလို့ရတာလည်းဆိုတော့ ကျတော်တို့ dynamic တွေကို redirect

လုပ်ရတဲ့အခါမှာ လည်း သူက တော်တော်လေးအသုံးဝင်ပါတယ်

ကျတော်တို့ လေ့လာလိုက်ကြရအောင်

အရင်ဆုံး ဒီလိုလေး ရေးလိုက်ပါမယ်

```
import { useNavigate } from "react-router-dom"
```

ဒီလိုလေး import လုပ်လိုက်ပြီးရင် ကျတော်တို့က သူ့ကိုအသုံးပြုမှာပါ

သူ့ကိုအသုံးပြုရင် သူက method တခုကို return ပြန်ပေးပါတယ်

အဲဒါကို ကျတော်က ဒီလိုလေး သိမ်းလိုက်ပါမယ်

```
const navigate = useNavigate()
```

အဲဒီ method ထဲမှာ သူက parameter တွေကို string value အနေနဲ့ လက်ခံပါတယ်

အဲဒါကို ကျတော်တို့က ဘယ်လိုမျိုး အသုံးပြုလို့ရသွားမှာလည်းဆိုတော့ ကျတော်တို့ function တွေထဲမှာ

သူ့ကိုအသုံးပြုပြီးတော့ သွားစေချင်တဲ့ route ကို redirect လုပ်ခိုင်းလို့ရသွားပါတယ်

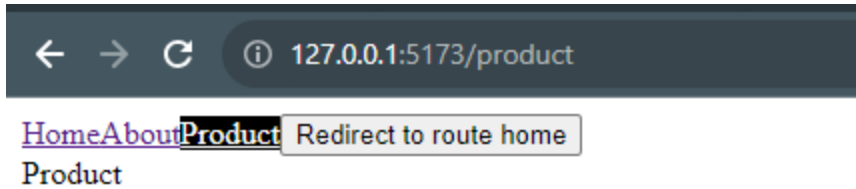
ကျတော်တို့ function လေးကို ဒီလိုလေးရေးလိုက်ပါမယ်

```
const navigate = useNavigate()
const clickHandler = () => {
  // rest of code
  navigate("/home")
}
```

ပြီးသွားရင်တော့ ကျတော်တို့ က ဒီလိုလေး အသုံးပြုလိုက်ပါမယ်

```
<button onClick={clickHandler}>Redirect to route home</button>
```

ကျတော်တို့ browser မှာ ကြည့်လိုက်တဲ့အခါမှာ ဒီလိုလေးတွေ့ရပါလိမ့်မယ်



ဒီလောက်ဆိုရင် သူ့ကိုအသုံးပြုတတ်သွားမယ် ထင်ပါတယ်

useParams Hook

ကျတော်တို့ react-router ကို အသုံးပြုပြီးတော့ route တွေကို ကျတော်တို့ page ချိန်းတာတို့ လုပ်တာကို လေ့လာခဲ့ပြီးပါပြီ နောက်ထပ်တစ်ခုကို ကျတော်တို့ ဆက်လေ့လာသွားကြရအောင် အဲဒါက ဘာလည်း ဆိုတော့ ကျတော်တို့ က လက်ရှိ ဘယ် route ကို ရောက်နေတာလည်းဆိုပြီးတော့ ပြန်ပြီးတော့ သိဖို့လိုအပ်တဲ့အခါမျိုးတွေ ရှိပါတယ်

မြင်သာအောင်ပြောရရင် ဘယ်လိုအခြေအနေမျိုးတွေမှာဆိုရင်တော့ ကျတော်တို့ e-commerce page တခုကို မြင်ကြည့်ရအောင်

အဲဒီမှာ ကျတော်တို့ item တွေအများကြီးရှိတဲ့အခါမှာ ကျတော်တို့ လိုချင်တဲ့ item ကိုနိပ်လိုက်တဲ့အချိန်မှာ အပေါ်က route မှာ ဒီလိုလေးဖြစ်သွားတာကိုမြင်ရပါလိမ့်မယ် item/1

ဒီလိုလေးတွေမျိုးပေါ့ အဲဒါက ဘာဖြစ်သွားတာလည်းဆိုတော့ ကျတော်တို့က id လေးတွေ တနည်းပြောရရင်

dynamic route တွေပေါ့ဗျာ အဲလိုမျိုးဖြစ်သွားတဲ့ id တွေ dynamic တွေဖြစ်တဲ့ route ကို ကျတော်တို့က ဘယ် id ဘယ် name နဲ့ရောက်နေတာပါဆိုပြီးတော့ ပြန်သိစေချင်တဲ့အခါမှာ useParams hook ကိုအသုံးပြုပေးရပါတယ်

ကဲ ကျတော်တို့ ကုဒ်လေးတွေကို ဒီလိုလေးအရင်ရေးလိုက်ကြရအောင်

```
import React from 'react'

function DynamicRoute() {
  return (
    <div>DynamicRoute</div>
  )
}

export default DynamicRoute
```

React Learning

အရင်ဆုံး ကျတော်တို့ component တစ်ခုကို တည်ဆောက်လိုက်ပါမယ်

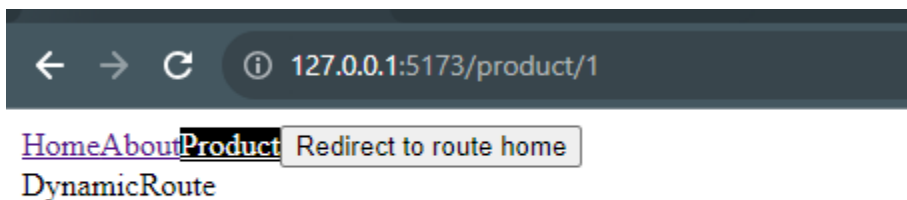
အဲဒီအတွက် ကျတော်တို့က Route တွေမှာ သူ့ကို ဒီလိုလေး အသုံးပြုထားလိုက်ပါမယ်

```
<Route path="/product" element={<Product />}></Route>
<Route path="/product/:id" element={<DynamicRoute />}></Route>
</Routes>
```

အဲဒီမှာ ကျတော်တို့ စတင်ပြီးတော့ မြင်ရမှာပါ product ရဲ့ အနောက်မှာ full colon နဲ့ id လေးကိုရေးထားတာပါပဲ

အဲလိုမျိုးရေးထားလိုက်ရင် react router က ဘယ်လိုမျိုး မြင်တာလည်းဆိုတော့ ဒီ route က dynamic ပြောင်းလဲ ပြီးတော့လာနိုင်တယ်ဆိုပြီးတော့ သူက သိပါတယ်

အဲဒါကို ကျတော်တို့က browser မှာ သွားစမ်းကြည့်ရင် ဒီလိုလေးမြင်ရပါလိမ့်မယ်



ဒီလိုလေးဆိုရင် ကျတော်တို့ နားလည်သွားမယ်ထင်ပါတယ်

ဆက်ပြီးတော့ ကျတော်တို့ hook လေးကိုအသုံးပြုလိုက်ကြရအောင်

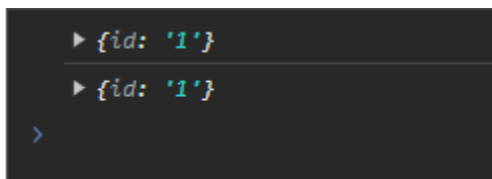
အရင်ဆုံး ဒီလိုလေးရေးလိုက်ပါမယ်

```
import { useParams } from 'react-router-dom'
```

သူ့ကိုအသုံးပြုရင် သူက object value တစ်ခုကို return ပြန်ပေးပါတယ် အဲဒါကို ကျတော်တို့ ဒီလိုလေးရေးလိုက်ပါမယ်

```
const params = useParams()
console.log(params)
```

ဒီလိုလေးရေးပြီးတော့ browser console မှာသွားကြည့်ရင် ဒီလိုလေးမြင်ရပါမယ်



ကျတော်တို့ Route မှာရေးခဲ့တဲ့ :id လေးက ဒီအချိန်မှာ object ရဲ့ key အနေနဲ့ ဝင်လာပြီးတော့ ကျတော်တို့ တကယ့် route ရဲ့ number ကို ကတော့ string type အဖြစ် object value အနေနဲ့ ပြပေးပါတယ်

React Learning

သူ့ကို ကျတော်တို့က ရသွားပြီးဖြစ်တဲ့အတွက် ကျတော်တို့ ကြိုက်တဲ့ id number ကိုပြောင်းပါစေ သူက ဒီလိုလေး ပြောင်းလဲပြီးတော့ ပြသွားပါလိမ့်မယ်
ဒီလောက်ဆိုရင် နားလည်သွားမယ်ထင်ပါတယ်
ကျတော်တို့ သူ့ကို browser မှာ ပြစေချင်ရင်ဒီလိုလေးရေးလိုက်ရင် သူက ပြနေပါလိမ့်မယ်

```
DynamicRoute.jsx > ...  
import React from 'react'  
  
import { useParams } from 'react-router-dom'  
  
function DynamicRoute() {  
  const params = useParams()  
  console.log(params)  
  return (  
    <div>  
      <h1>{params.id}</h1>  
    </div>  
  )  
}  
  
export default DynamicRoute
```

params က object type ဖြစ်တဲ့အတွက် ကျတော်တို့ သူ့ကို destructuring ပုံစံနဲ့ ရေးလို့လည်းရပါတယ်
ဒီလိုလေးဖြစ်သွားမှာပါ

```
function DynamicRoute() {  
  const params = useParams()  
  console.log(params)  
  const {id} = params;  
  return (  
    <div>  
      <h1>{id}</h1>  
    </div>  
  )  
}
```

ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ react-router ကိုတော်တော်လေး နားလည်သွားမယ်ထင်ပါတယ်

Additional Hooks

useRef hook

ကျတော်တို့ ဒီအထိရောက်လာပြီဆိုရင် react ကို beginner level လောက်ကို တော်တော်လေး ရသွားပြီလို့ပြောလို့ရပါပြီ ကျတော်တို့ react ရဲ့ နောက်ထပ် အသုံးဝင်တဲ့ hook လေးတွေကိုလည်း ဆက်ပြီး တွေ့လေ့လာသွားကြရအောင်ပါ အဲဒါက ဘာလည်းဆိုတော့ useRef ဆိုတဲ့ hook လေးပါ အဲဒါက ဘယ်လိုမျိုး အလုပ်လုပ်တာလည်းဆိုတော့ ကျတော်တို့က တချို့ component တွေရဲ့ data တွေကို state နဲ့ ယူတဲ့အခါမှာ ကျတော်တို့ရဲ့ component တွေက ခနခန rerender လုပ်နေတာကိုမြင်ရပါမယ် ဗျ

အဲလိုမျိုး ခနခန ဖြစ်တာက ကျတော်တို့ small application (project) တွေမှာ မသိသာပေမယ့် အရမ်းအရမ်းကြီးတဲ့ project တွေ ဖြစ်လာတဲ့အခါကျရင် သူက တော်တော်လေး performance ကို ထိခိုက်လာစေ ပါတယ်အဲအတွက်ကြောင့်တချို့သောအရာတွေကို ကျတော်တို့ရဲ့ react က နေပြီးတော့ state လိုမျိုး data ကို ယူပေးနိုင်တဲ့ hook လေး တခုကို ထုတ်ပေးလိုက်ပါတယ် အဲဒါက useRef လေးပါပဲ

သူ့ကို ကျတော်တို့က ဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတာကို လေ့လာလိုက်ကြရအောင်ရင် အရင်ဆုံး ကျတော်တို့ ဒီလိုလေးရေးလိုက်ပါမယ်

```
import { useRef } from 'react'

function UseRefHook() {
  const ref = useRef(0)
  return (
    <div>
      <h1>Count - {ref.current}</h1>
      <button>Add</button>
    </div>
  )
}

export default UseRefHook
```

အရင်ဆုံး component တစ်ခုကို ဒီလိုလေး တည်ဆောက်လိုက်ပါမယ်

ဒီအချိန်မှာ ကျတော်တို့သတိထားမိမှာကိ ကျတော်တို့ count value ကို ref.current နဲ့ ခေါ်ထားတာကိုမြင်ရပါ

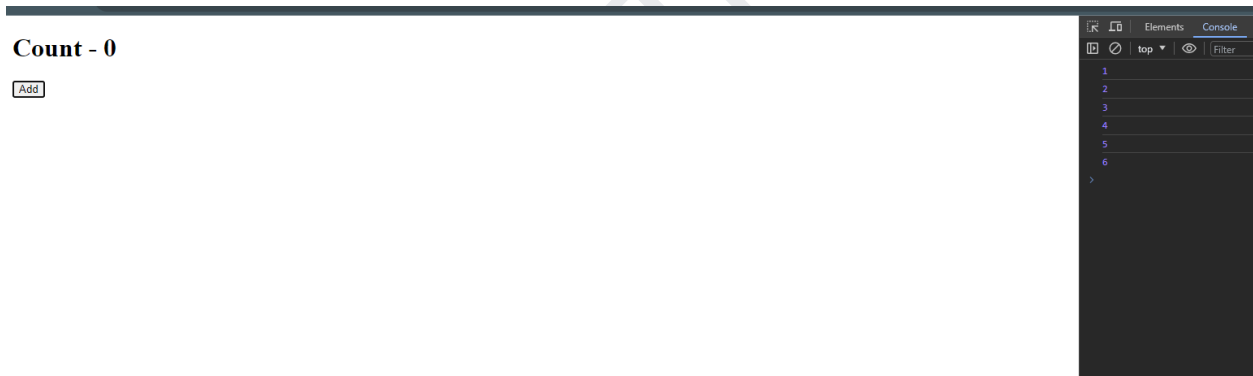
မယ်ဗျ အဲဒါက ဘာလည်းဆိုတော့ ကျတော်တို့ useRef က သူက object တခုကို return ပြန်ပေးပါတယ် သူ့ထဲမှာဘာတွေပါတာလည်းဆိုတော့

current ဆိုပြီးတော့ ကျတော်တို့ useRef ရဲ့ parenthesis ထဲမှာ initial value လေးနဲ့ ပြန်ပေးထားတာပါ အဲဒါကို ကျတော်တို့က button onClick ရဲ့အထဲမှာ function လေးကို ဒီလိုလေးရေးလိုက်ပါမယ်

```
function UseRefHook() {
  const ref = useRef(0)
  const clickHandler = () => {
    ref.current += 1
    console.log(ref.current)
  }
  return (
    <div>
      <h1>Count - {ref.current}</h1>
      <button onClick={clickHandler}>Add</button>
    </div>
  )
}

export default UseRefHook
```

တခုသတိထားရမှာက ကျတော်တို့က useRef လေးကို update လုပ်လိုက်တိုင်းမှာ ကျတော်တို့က current ကိုပဲ သွားသွားပြီးတော့ update လုပ်ပေးရမှာပါ
ဒီလိုလေးရေးပြီးတော့ ကျတော်တို့ browser ကိုသွားကြည့်လိုက်ကြရအောင်



ဒီမှာ ထူးခြားတာ ဘာတွေ့ရမှာလည်းဆိုတော့ ကျတော်တို့ရဲ့ useRef ရဲ့ data က ပြောင်းလဲသွားပေမယ့် Browser မှာ 0 ကနေပြီးတော့ ပြောင်းလဲသွားတာကိုမမြင်ရပါဘူး
ကျတော်state မှာတုန်းကလည်းပြောခဲ့ပါတယ်
ကျတော်တို့ပြောင်းလဲမှုကို မြင်စေချင်ရင် state နဲ့ setState ကို အသုံးပြုပေးရမှာပါဆိုပြီးတော့
အဲသဘောအတိုင်းပါပဲ ref က သူက background မှာပဲအလုပ်လုပ်ပေးသွားတာပါ
ဒါဆိုရင် သူကို ကျတော်တို့ rerender လုပ်စေချင်ရင် ဘယ်လိုရေးရမှာလည်းပေါ့
တစ်ကယ် ကျတော်တို့ရဲ့ အဓိက ရည်ရွယ်ချက်ကိုက rerender ကိုမဖြစ်စေချင်လို့ပါ တကယ်လို့
လုပ်ရမယ့်အနေအထား လေးရှိလာခဲ့ရင် state ကိုအသုံးပြုလိုက်တာက ပိုကောင်းပါတယ်
အခုက ကျတော့ ကျတော်က useRef ကိုဘယ်လိုမျိုးအလုပ်လုပ်တာလည်းဆိုတာကို
ပဲလေ့လာရပါသေးတယ်

React Learning

သူ့ကိုအဓိက ဘယ်မှာအသုံးပြုရတာလည်းဆိုတော့ ကျတော်တို့က input data တွေ မှာ အဓိက အသုံးပြုကြပါတယ်

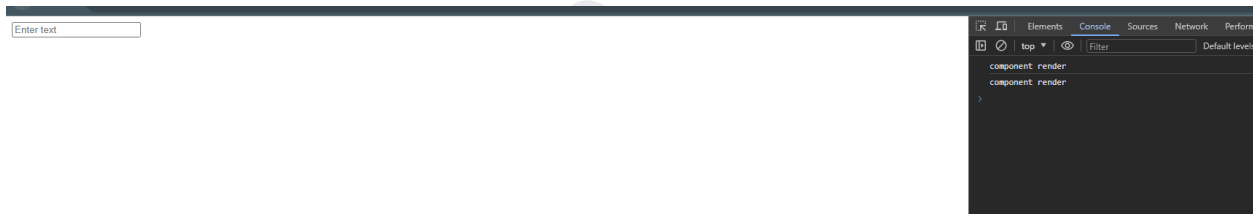
ဘာလို့ဆိုတော့ ကျတော်တို့ရဲ့ input တွေမှာက onChange လုပ်ရတဲ့အခါ သူက component ကို ခန့်ခန့် rerender လုပ်ပေးနေလို့ပါအဲဒါတွေကြောင့် ကျတော်တို့က useRef ကို အသုံးပြုရတာပါ ကျတော်တို့ ဒီလိုလေး ရေးလိုက်ပါမယ်

```
import { useEffect, useRef } from 'react'

function UseRefHook() {
  const inputRef = useRef("")
  const changeHandler = (e) => {
    inputRef.current = e.target.value
    console.log(inputRef.current)
  }
  useEffect(() => {
    console.log("component render")
  }, [])
  return (
    <div>
      <input type="text" ref={inputRef} onChange={changeHandler} placeholder='Enter text' />
    </div>
  )
}

export default UseRefHook
```

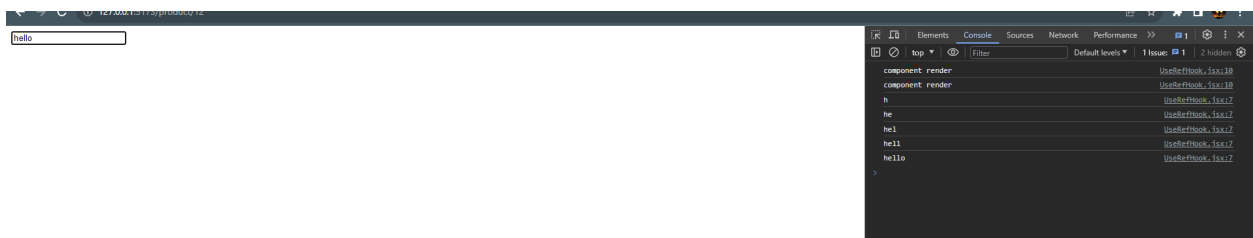
ဒီလိုလေးရေးပြီးတော့ ကျတော်တို့ browser ကို တချက်သွားကြည့်လိုက်ပါမယ်



ဒီလိုလေးမြင်ရပါမယ်

ကျတော်တို့က ဒီထဲမှာစာကိုရိုက်ကြည့်တဲ့အခါမှာ useEffect က အလုပ်မလုပ်တော့ပါဘူး

ဘာဖြစ်လို့လည်းဆိုတော့ useEffect က state update လုပ်တဲ့အခါမှ ပြန်ပြီးတော့အလုပ်လုပ်ပေးတာပါ



ဒီလောက်ဆိုရင်တော့ ကျတော်တို့ useRef hook ကို နားလည်သွားမယ်လို့ထင်ပါတယ်

ကျတော်တို့ useRef ကို လေ့လာပြီးတဲ့အခါမှာ ဘာကို သိသွားရမှာလည်းဆိုတော့ ကျတော်တို့ရဲ့ data တွေကို user တွေ client တွေကို ပြန်ပြီးတော့ render ပြချင်တဲ့အခါမှာ state ကို အသုံးပြု ရပြီးတော့

React Learning

data တွေကို client တွေကိုမပြချင်ပဲ နဲ့ နောက်ကွယ်ကနေပြီးတော့အလုပ်လုပ်သွားစေချင်တဲ့အခါမျိုး
ဥပမာ
form handling တွေမှာဆိုရင် useRef ကိုအသုံးပြုပေးရမှာပါ

Bate Thar