React JS

By

Tin Mai Zaw

Northern City

1. Introduction
2. React JS ဆိုတာဘာလဲ?

React JS သည် web application user interface တွေကို ဖန်တီးရန်အတွက် တီထွင်ဖန်တီးထားတဲ့ JavaScript library ဖြစ်ပါတယ်။ React ကို Facebook မှ ဖန်တီးထုတ်လုပ်ထားတာ ဖြစ်ပြီး React library ကို စဉ်ဆက်မပြတ် update လုပ်ဆောင်နေပြီးတော့ error bug များကို ပြင်ဆင်ရင်းနဲ့ component အသစ်များကို ထပ်ထည့်ထပ်ထည့် React JS ကို ပိုမိုကောင်းမွန်အောင် ကျယ်ကျယ် ပြန့်ပြန်အသုံး ပြုလာနိုင်အောင် ပြင်ဆင်လုပ်ဆောင်နေခဲ့တာ အခုဆို react version 20 အထိရောက်ရှိလာခဲ့ပြီ ဖြစ်ပါတယ်။ သည်စာအုပ်ထဲမှာ React ရဲ့ project setup installation ပြုလုပ်ခြင်း ၊ React အခြေခံရေးသားခြင်း၊ component ဖန်တီးခြင်း၊ JSX ကနေ စတင်ပြီး state management ၊ form programming ၊ routing ကဲ့သို့ React JS ရဲ့ အဆင့်မြင့် သဘောတရားများကို example တွေနဲ့ step by step လေ့လာရမှာ ဖြစ်ပါတယ်။ သည် စာအုပ်ကို အတွေ့အကြုံရှိပြီးသား web developer တွေလည်း ကိုကား အသုံးပြုနိုင်သလို အခုမှ စပြီး React ကို လေ့လာနေတဲ့ beginner တွေလည်း အသုံးပြုနိုင်ပါတယ်။ React JS ကို လေ့လာမယ်ဆိုရင် HTML၊ CSS၊ Vanilla JavaScript ဒါမှမဟုတ် Pure JavaScript နဲ့ programming OOP လောက်အထိ လေ့လာထားဖို့လိုပါမယ်။ Absolute beginner တွေအတွက် မသင့်တော်ပါဘူး။ React JS ကို အသုံးပြုထားတဲ့ popular web application တွေကတော့ Facebook ၊ Instagram ၊ Netflix နဲ့ reddit အစရှိတဲ့ popular platform တွေပဲဖြစ်ပါတယ်။

1. React JS ကို ဘာကြောင့် အသုံးပြုသင့်သလဲ?

React JS ကို ဘာကြောင့်သုံးသင့်သလဲဆိုတဲ့ အကြောင်းအရင်းကတော့ −

1. Component based ဖြစ်ခြင်း
2. App performance speed ပိုမိုမြန်ဆန်လာစေခြင်း
3. Code maintenance လုပ်ဖို့ ပိုမိုလွယ်ကူစေခြင်း

တို့ကြောင့်ဖြစ်ပါတယ်။

1. Component based ဖြစ်ခြင်း

Web App တစ်ခု develop လုပ်တော့မယ်ဆိုရင် component အများကြီး ဆောက်ရပါမယ်။ Component တစ်ခုချင်းစီဟာ သူ့အစိတ်အပိုင်းနဲ့သူ independent ဖြစ်နေကြပြီး coding logic တွေကိုလည်း သူ့ component နဲ့သူ သီးသန့်ရေးရပါတယ်။ ဉပမာ Header component မှာ ရေးထားတဲ့ code logic တွေဟာ side menu component ထဲမှာ ရှိတဲ့ code နဲ့ design တွေအပေါ် effect သက်ရောက်မှုမရှိပဲ သီးသန့် လွတ်လွတ်လပ်လပ် ရေးသားနိုင်မှာ ဖြစ်ပါတယ်။

Components :

Header

navbar

Side menu

card

footer

Page1

Web pages:

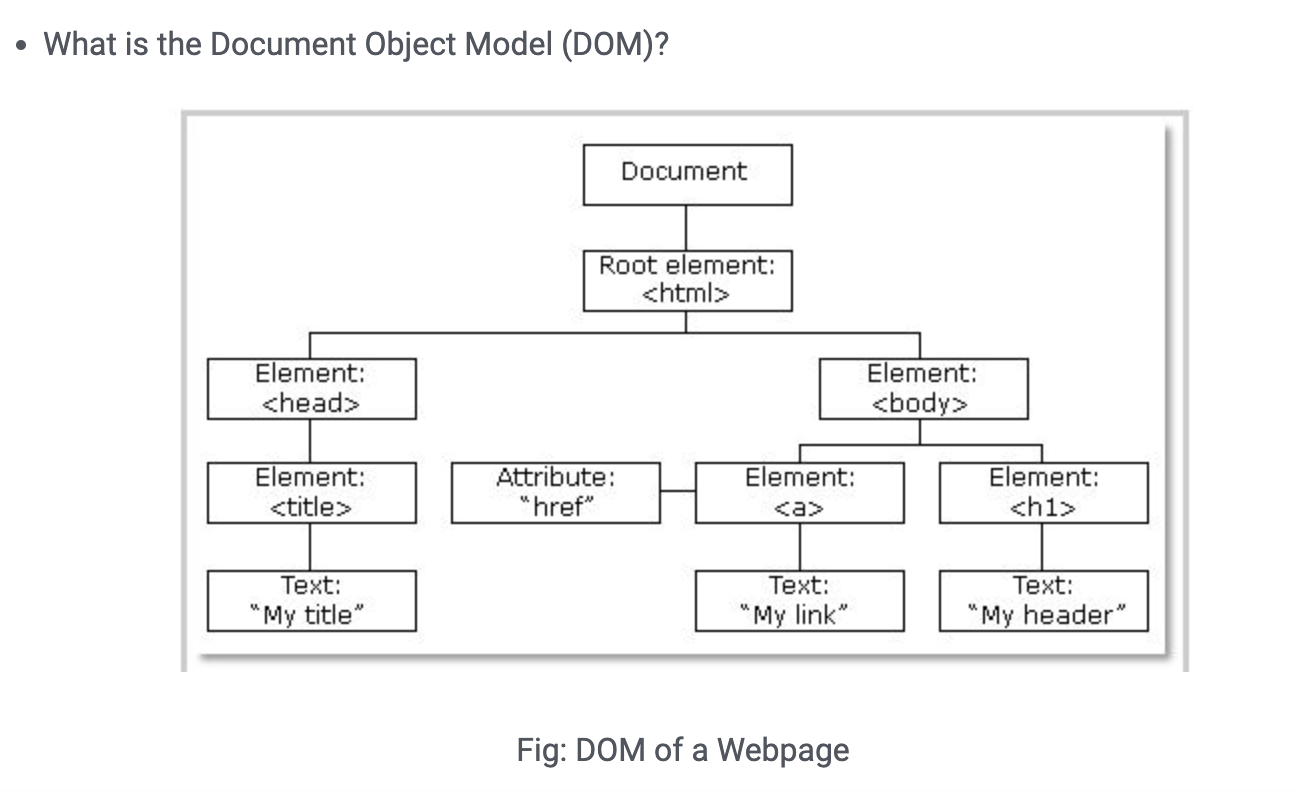
Page2

1. App Performance ပိုမိုမြန်ဆန်ခြင်း

Virtual DOM ကတော့ React ရဲ့ အရေးပါဆုံး concept တစ်ခုလို့ ပြောလို့ရပါတယ်။ DOM ရဲ့အရှည်ကောက်က Document Object Model ဖြစ်ပါတယ်။ Real DOM က ပုံမှန်အားဖြင့် web page တစ်ခုရဲ့ actual HTML structure ကို ကိုယ်စားပြုထားပါတယ်။ Developer ကရေးလိုက်တဲ့ code တွေကို web browser ပေါ်မှာ ထွက်လာအောင် render လုပ်တဲ့ အခါအချိန်ယူရပါတယ်။

React သည် update လုပ်ထားတဲ့ virtual DOM ကို real DOM နှင့် နှိုင်းယှဉ်ပြီး real DOM ကို လျော်ညီစွာ update ပြန်ပြန်လုပ်ပေးပါတယ်။ သည်လို update လုပ်ပေးတဲ့ ဖြစ်စဉ်ကို reconciliation လို့ ခေါ်ပါတယ်။ ဒီလို လုပ်ဆောင်ခြင်း နည်းစနစ်ကို Diffing Algorithm ဒါမှမဟုတ် heuristic algorithm လို့ခေါ်ပါတယ်။

Document Object Model (DOM)



ပိုမိုမြင်သာအောင်ဖော်ပြရမယ်ဆိုရင် Developer က code ကို update လုပ်ပြီး Page တစ်ခုလုံးကို render ခဏခဏ ပြုလုပ်ပြီး updated output တွေကို ကြည့်မယ်ဆိုရင် စိတ်တိုင်းကျ result တစ်ခုရဖို့ဆိုရင် DOM manipulation က အရမ်းကို slow ဖြစ်သွားမယ်။ Memory လည်း waste ဖြစ်ပါမယ်။ ဒီ problem တွေကို ဖြေရှင်းဖို့ React က virtual DOM ဆိုတဲ့ နည်းစနစ်တစ်ခု ဖန်တီးအသုံးပြုလာတာ ဖြစ်ပါတယ်။ Virtual DOM ကို အသုံးပြုခြင်းအားဖြင့် DOM manipulation က လွယ်သွားမယ်။ Page တစ်ခုလုံးကို ခဏခဏ render လုပ်စရာမလိုပဲ လိုအပ်တဲ့ component ကို ပဲ update လုပ်မယ်။ ပြီးတော့ render လုပ်မယ်ဆိုရင် memory waste မဖြစ်တော့ဘူးပေါ့။ Update process လည်းအရမ်းမြန်သွားပါမယ်။

ဉပမာ တစ်ခု ထပ် ပြောကြည့်ရအောင်။ သာမန် web app တွေမှာဆို web page တစ်ခုရဲ့ အစိတ်အပိုင်း တစ်ခု show details button ကိုနိုပ်လိုက်တယ်ဆိုပါစို့။ Detail Page ကို load လုပ်တဲ့အခါ Detail Page မှာ ရှိသမျှ component တွေအားလုံးကို reload လုပ်ပေးပါမယ်။ အဲဒါကြောင့် process က နဲနဲ slow ဖြစ်သွားပါတယ်။ React နဲ့ရေးထားတဲ့ app တွေမှာတော့ real DOM မှာရှိပြီးသား component အားလုံးကို reload လုပ်မှာ မဟုတ်ပါဘူး။ Updated component တစ်ခုထဲကို ပဲ reload လုပ်ပေးမှာမို့ application ကို သုံးတဲ့ user တွေအနေနဲ့ application performance speed ကို ကောင်းကောင်း ထိတွေ့ခံစားရမှာ ဖြစ်ပါတယ်။

Normal Rendering

Detail page

Master page

1

2

1

Reload

All

The

Components

click

4

3

React Rendering

Detail page

Master page

Reload

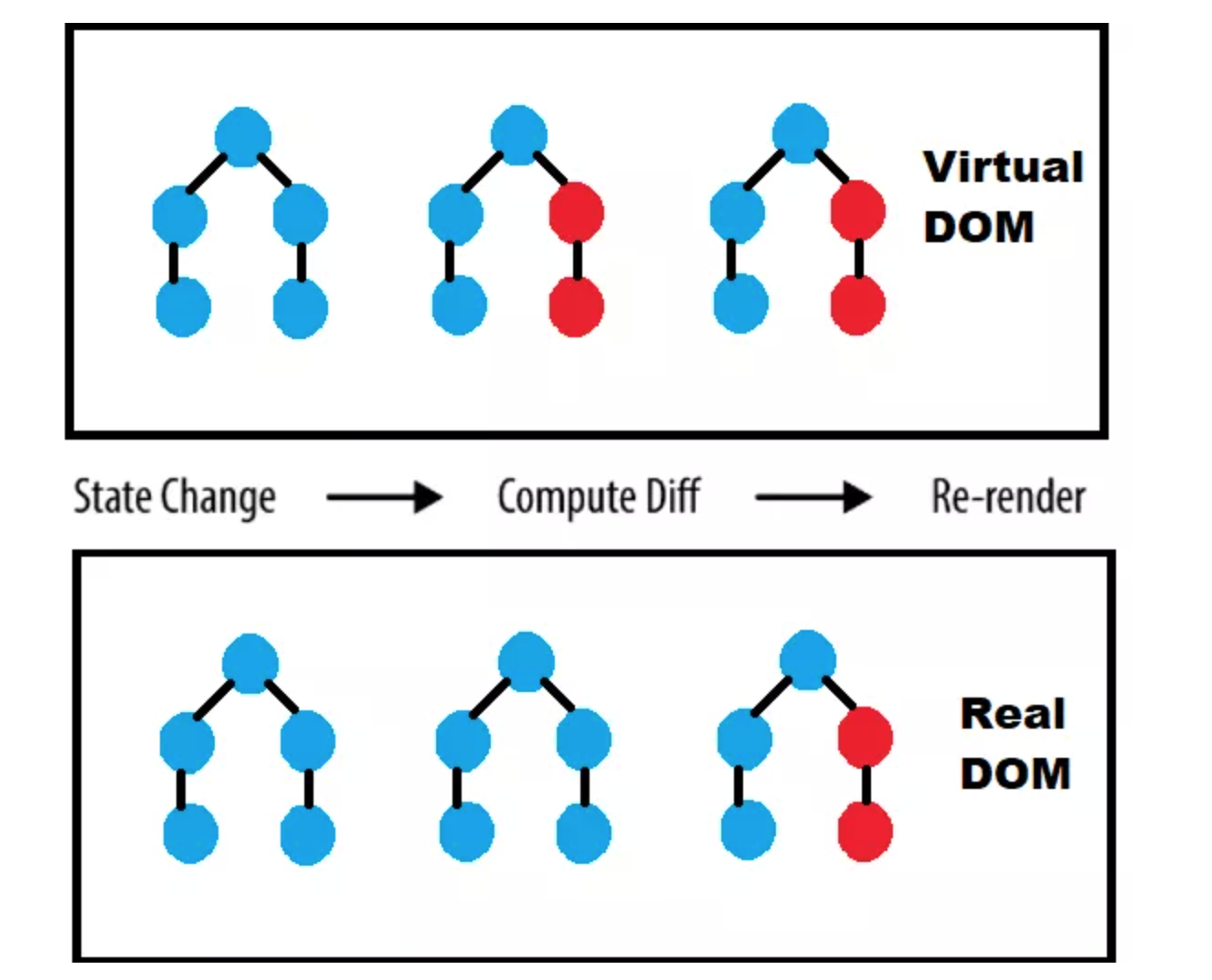
Only One

Components

click

1

Virtual DOM & Real DOM



React Details Process

Previous Virtual DOM

New Virtual DOM

Node1

Node2

Node3

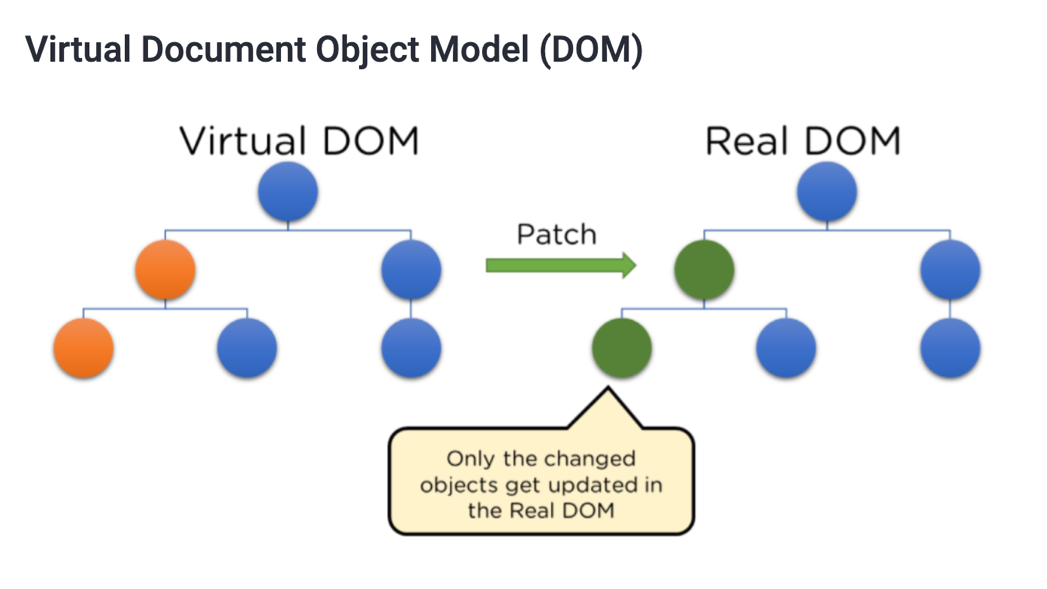
Actual DOM

Changes

React Create Elements

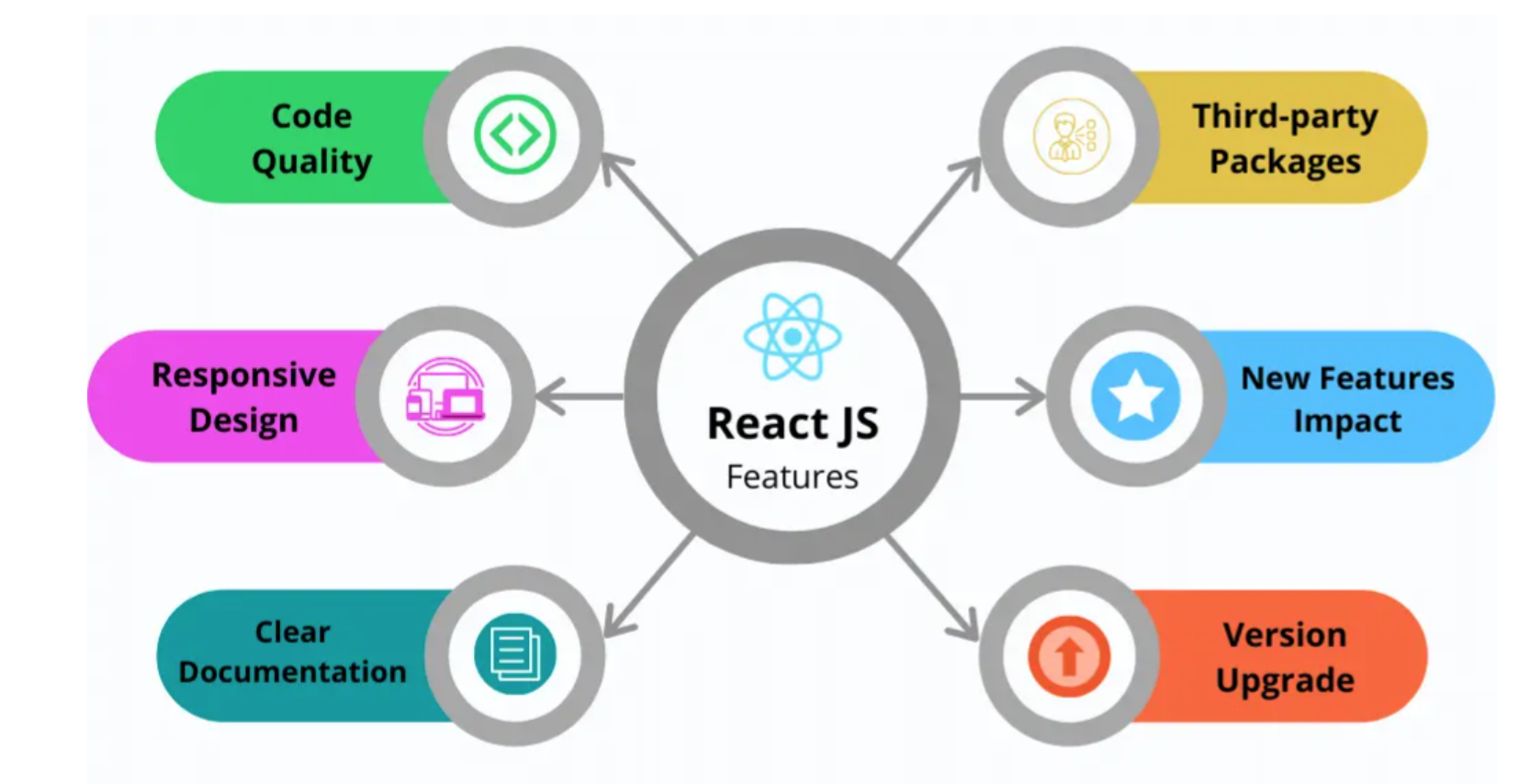
Commit Phase

Render Phase



1. Code Maintenance ပိုမိုလွယ်ကူလာစေခြင်း

ရေးပြီးသား Component code တွေအားလုံးကို ကြိုက်တဲ့ application projects ၊ ကြိုက်တဲ့ project စာမျက်နှာတွေမှာ ထပ်ခါထပ်ခါ ပြန်လည် အသုံးပြုနိုင်တာ ဖြစ်တဲ့ အတွက် Quality Code တွေကို maintain လုပ်ရတာ အင်မတန်မှ အဆင်ပြေပါတယ်။ အချိန်နဲ့အမျှ Quality ရှိတဲ့ Code တွေကြောင့် application performance နဲ့ application quality တွေကို တိုက်ရိုက် အကျိုးပြုမှာ ဖြစ်ပါတယ်။ App Component တွေအားလုံးက independently ရေးရတာ ဖြစ်တဲ့အတွက် ပြန်လည် refurbish လုပ်ရမယ့် အစိတ်အပိုင်းတွေ update လုပ်ရမယ့် အပိုင်းတွေကို အခြားသော application component အစိတ်အပိုင်းတွေကို ထိခိုက်မှုမရှိစေပဲ သီးသန့် ပြင်ဆင် ဆောင်ရွက်နိုင်ပါတယ်။ပြီးတော့ new feature component ကို လည်း လွယ်ကူစွာ ထပ်ထည့် နိုင်ပါတယ်။

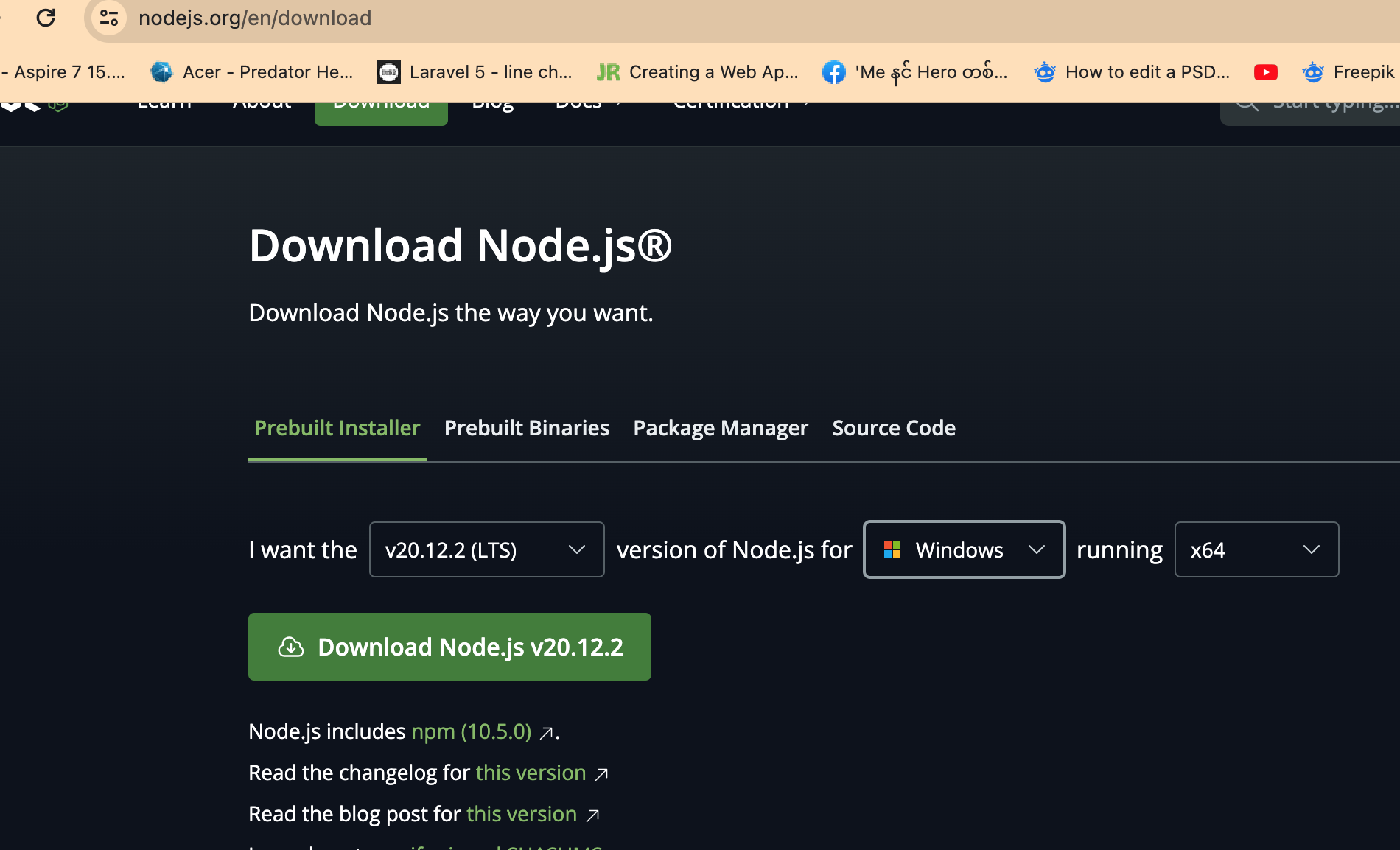


II. Installation and setup

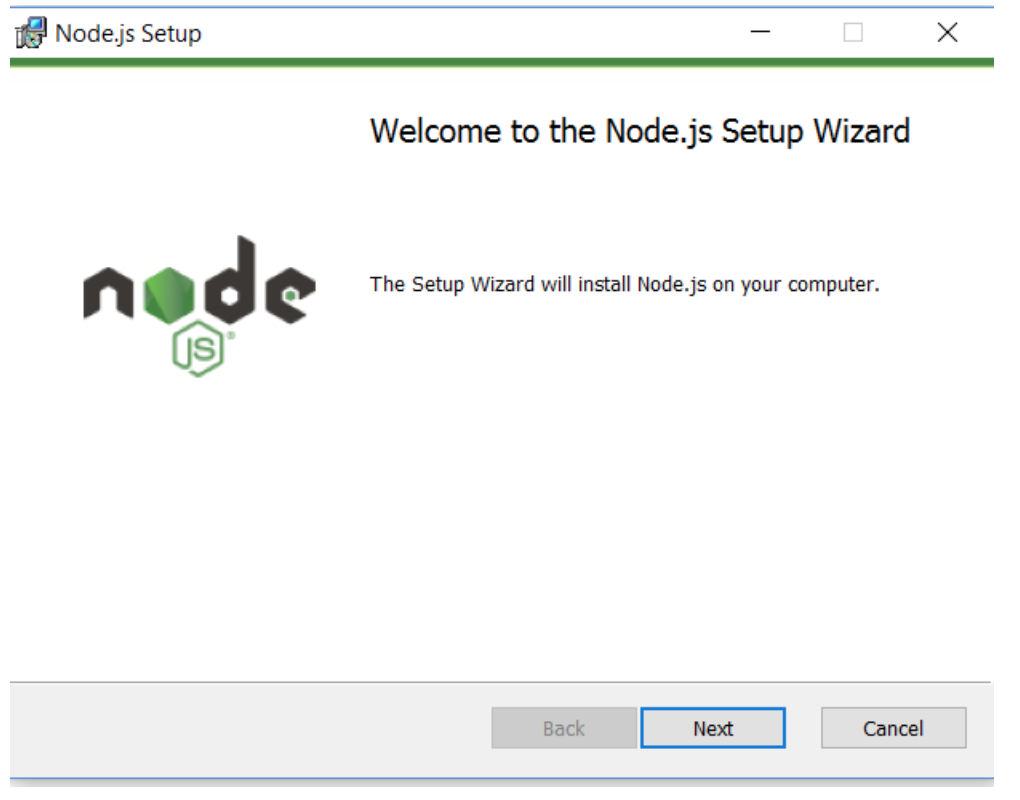
React application ကိုရေးဖို့ရာ ကိုယ့်ရဲ့ ကွန်ပျူတာမှာ Node JS ကို အရင် install လုပ်ရပါမယ်။

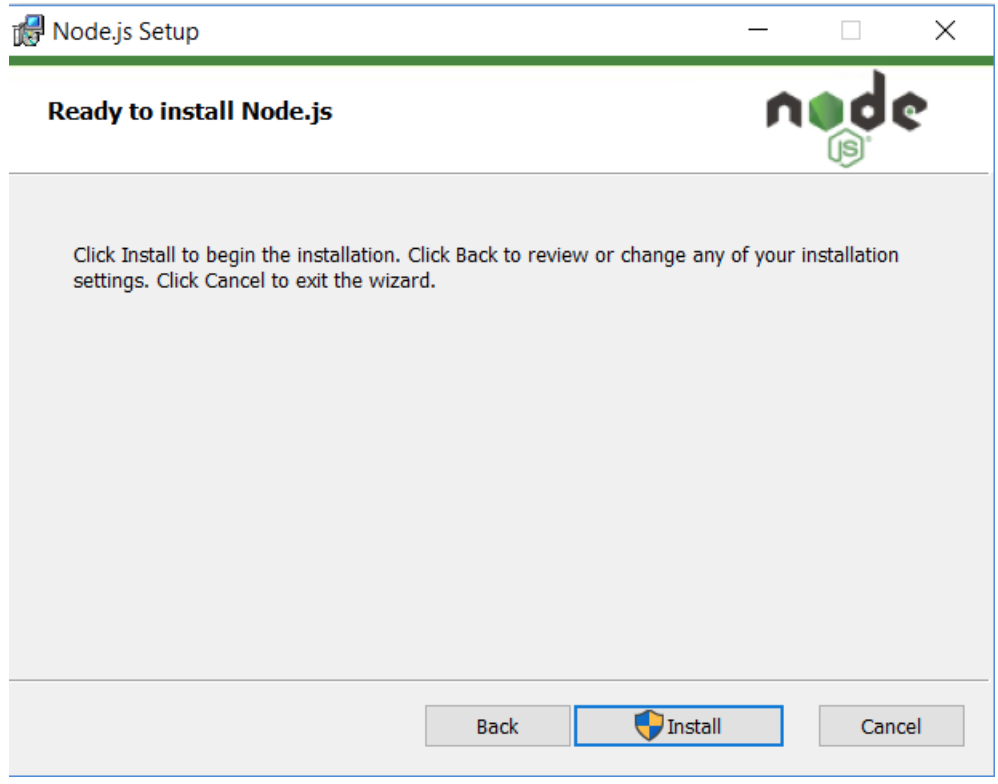
အောက်က download link ကို သွားပြီး Node JS ကို ဒေါင်းလုပ်ဆွဲရပါမယ်။ ကိုယ်အသုံးပြုနေတဲ့ ကွန်ပျူတာမှာ Node JS install လုပ်ပြီးသားဆိုရင် တော့ ဒီ installation step ကိုကျော်လိုက်ပါ။

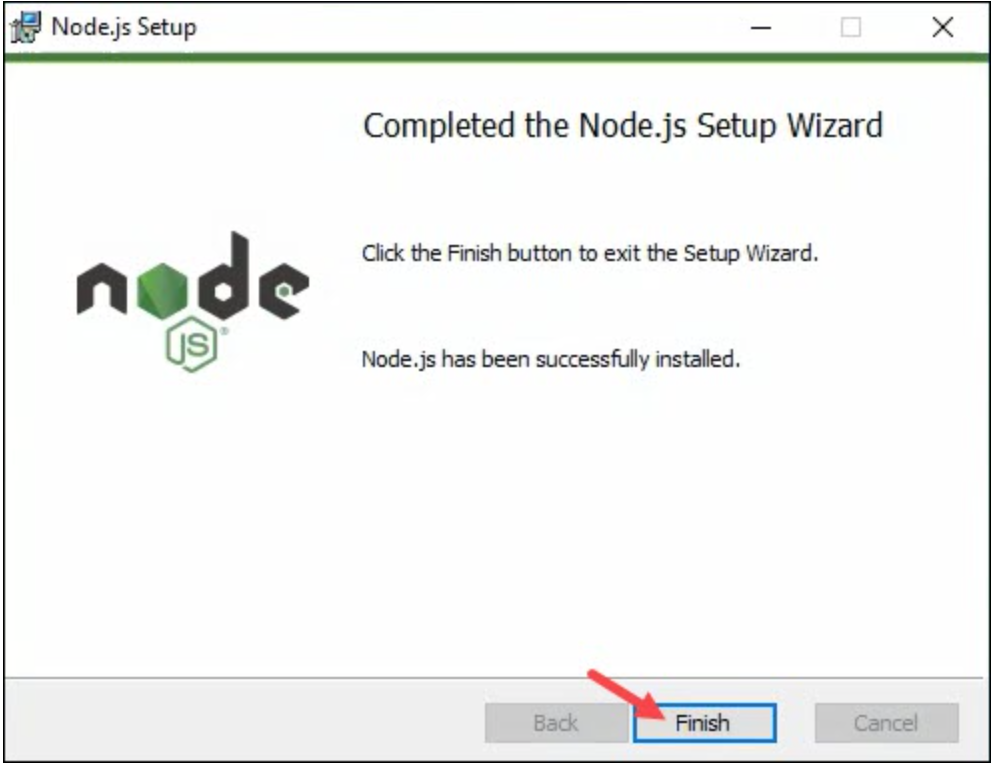
[www.nodejs.org/end/download](http://www.nodejs.org/end/download)



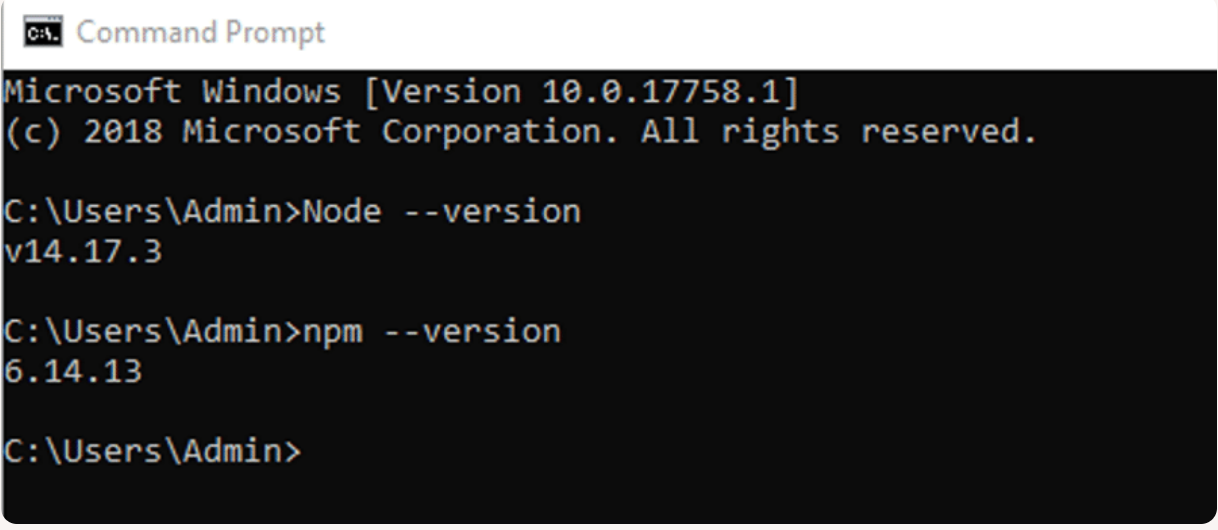
Download ဆွဲပြီးသွားရင် Computer ရဲ့ download folder ကိုသွားပြီး Node JS setup file ကို download click ခေါက်ပြီး အောက်ပါအတိုင်း installation လုပ်ရပါမယ်။

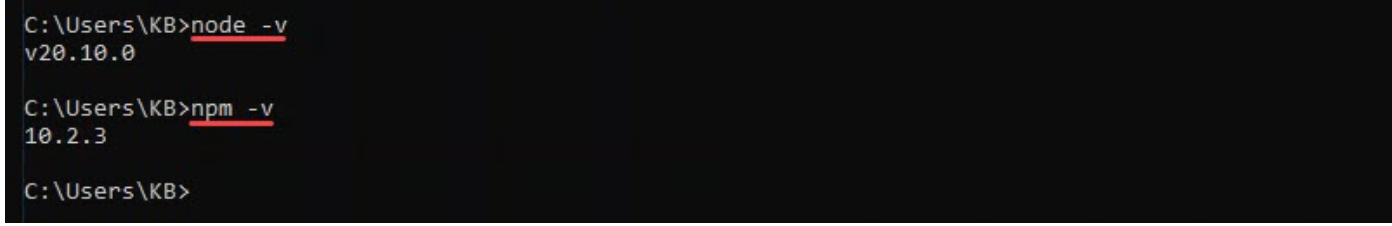






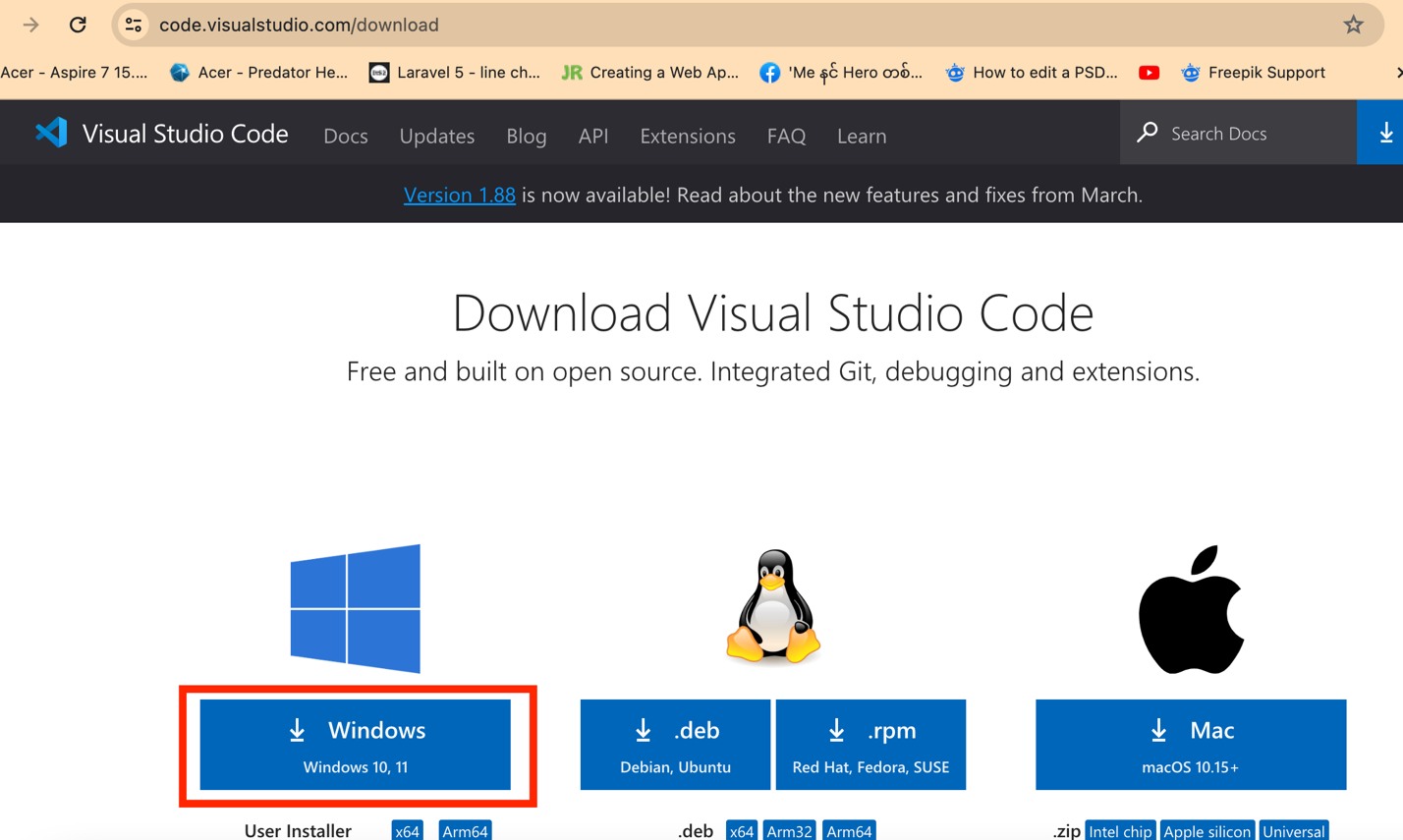
Node JS installation ပြီးသွားရင် Node JS version နဲ့ npm version ကို command mode မှာ အောက်ပါအတိုင်း ကြည့်လို့ရပါတယ်။



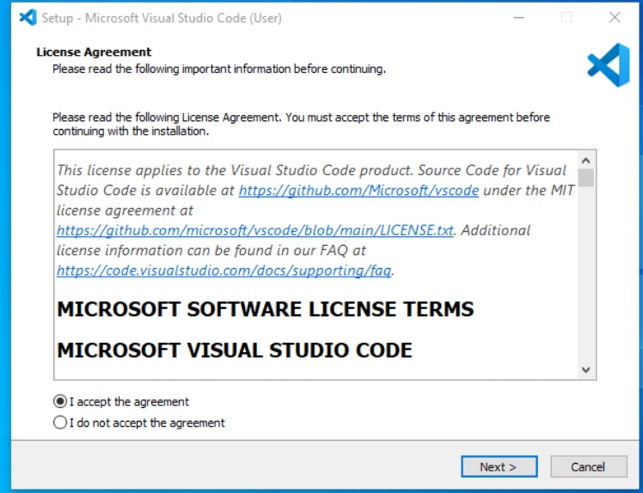


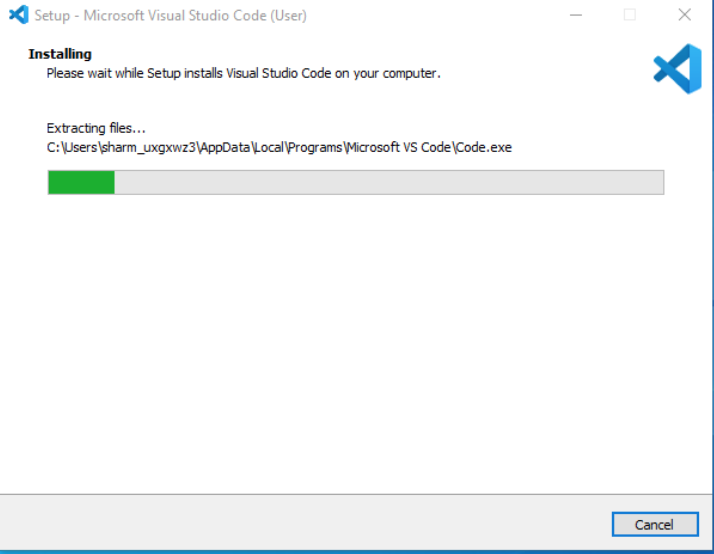
Node JS download လုပ်ပြီး installation ပြီးသွားရင် တော့ vs code ကို install လုပ်ရပါမယ်။

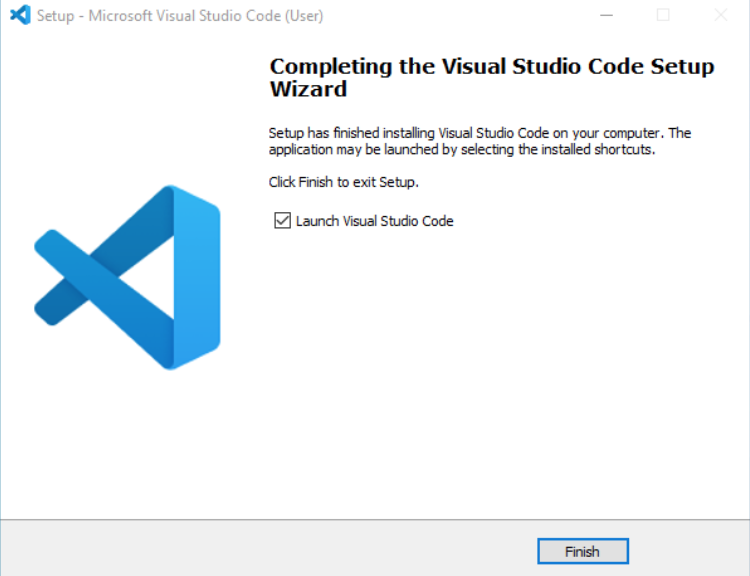
<https://code.visualstudio.com/download>

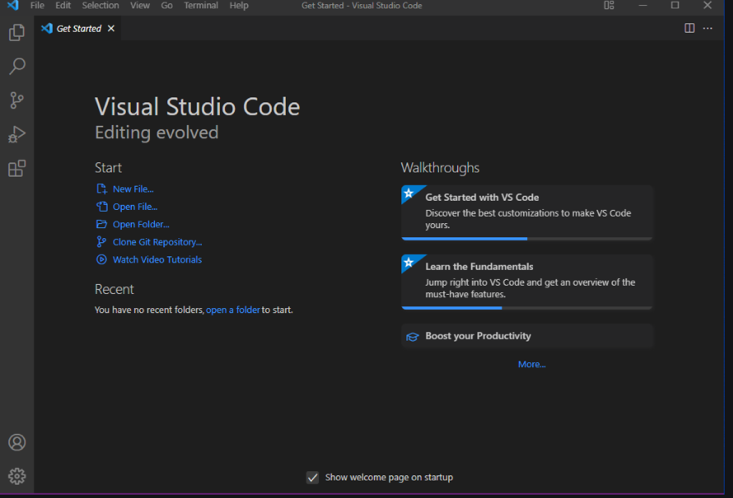


VS Code download ဆွဲပြီးသွားရင် အောက်ပါအတိုင်း install လုပ်ပါ။









လိုအပ်တဲ့ software Installation တွေပြီးသွားပြီဆိုရင်တော့ react application ကို စရေးလို့ ရပါပြီ။

1. Practical Lessons

Lesson 1

 Link :

Running the first application

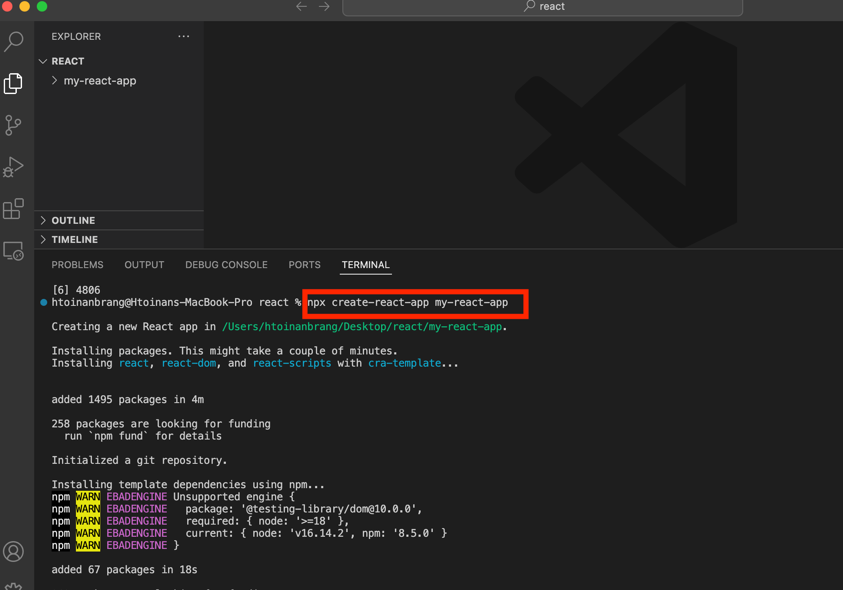
React Environment setup ကို အောက်ပါအတိုင်း ၂ မျိုး တည်ဆောက်နိုင်ပါတယ်။

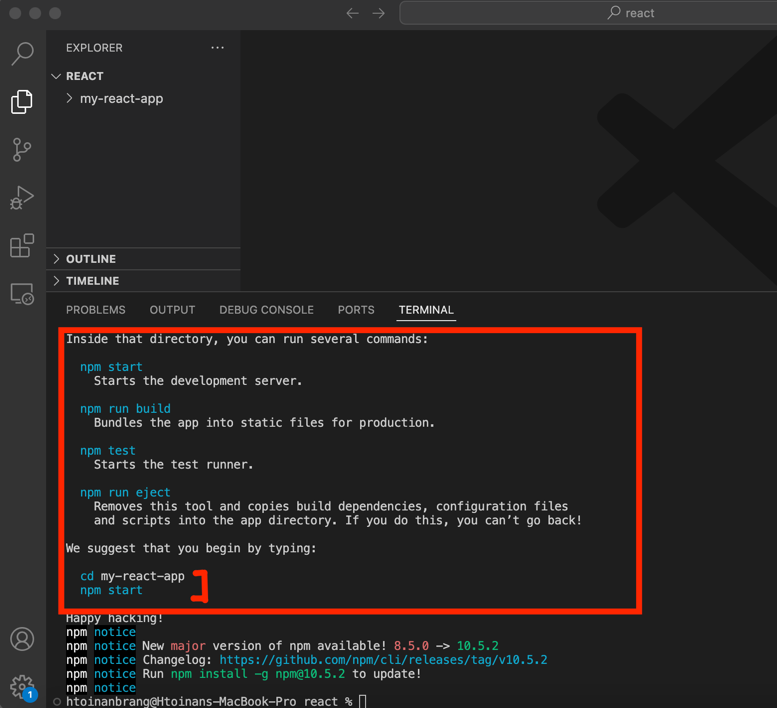
1. Webpack နဲ့ babel သုံးပြီး Manual setup ပြုလုပ်နိုင်ပါတယ်။
2. create-react-app command နဲ့ auto setup ပြုလုပ်နိုင်ပါတယ်။

ဒီ စာအုပ်ထဲက lesson တွေမှာ တော့ create-react-app command ကို အသုံးပြုသွားမှာ ဖြစ်ပါတယ်။ auto setup ဖြစ်တဲ့အလျှောက် အချိန်ကုန်သက်သာပြီး configuration လုပ်ရတာ လွယ်ကူသွားပါမယ်။ ဒီ command ကို သုံးတော့မယ်ဆို Node JS ကို install လုပ်ထားဖို့တော့ လိုပါတယ်။

Command:

npx create-react-app my-react-app



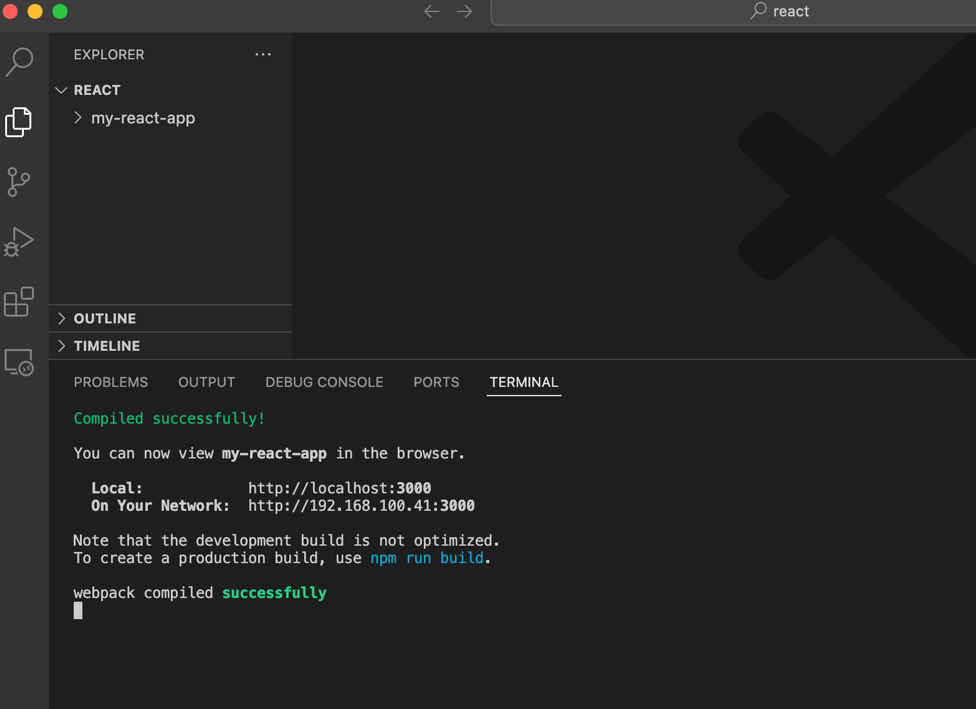


Command:

cd my-react-app

Command:

my-react-app> npm start





Congratulation ပါ။ Running the first React Application ကို အောင်မြင်သွားပါပြီ။ အခုကနေစပြီး React Lesson တွေကို တစ်ခုပြီးတစ်ခု ဆက်လေ့လာသွားကြပါမယ်။

 Lesson 2

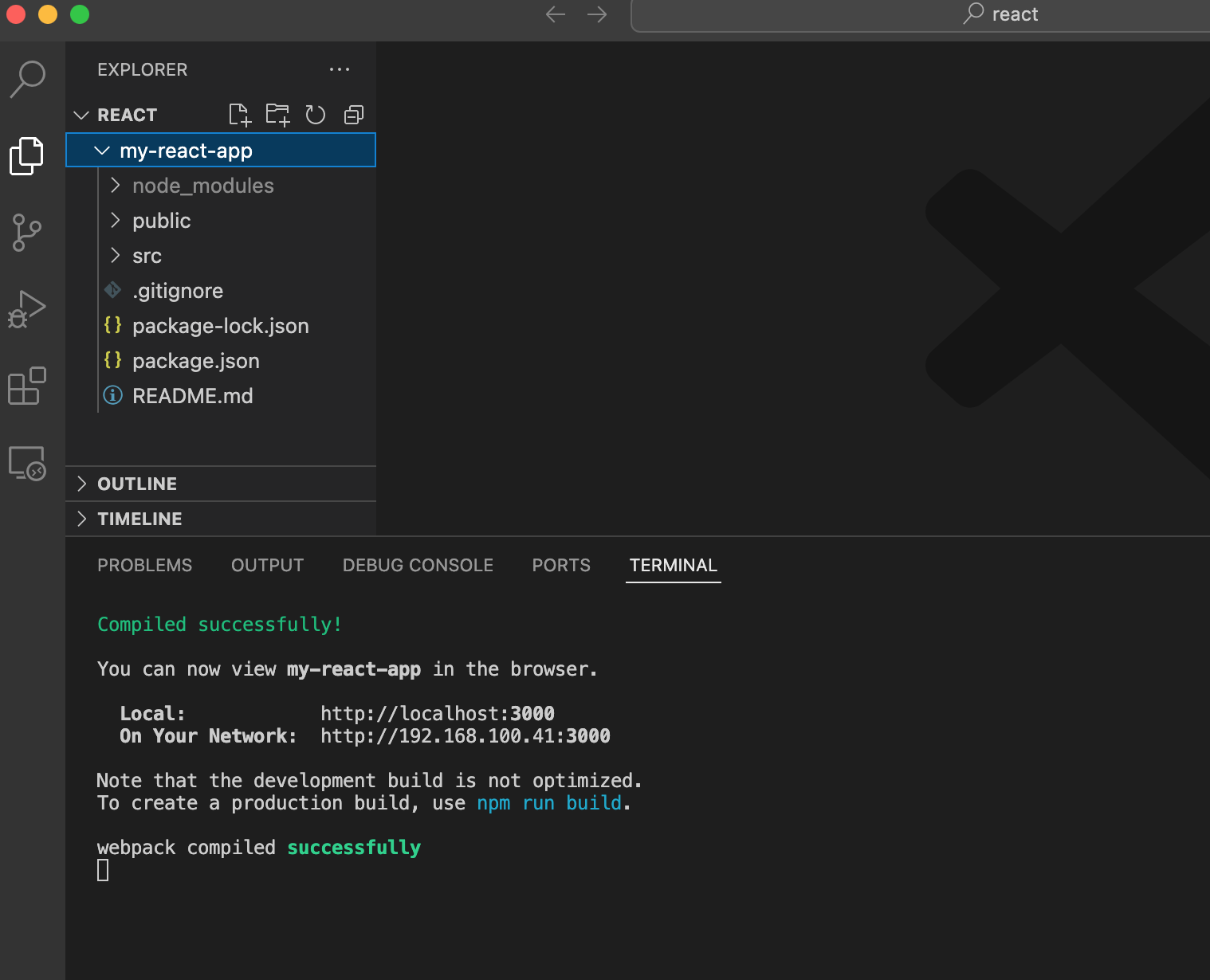


Link:

Discussion on project structure

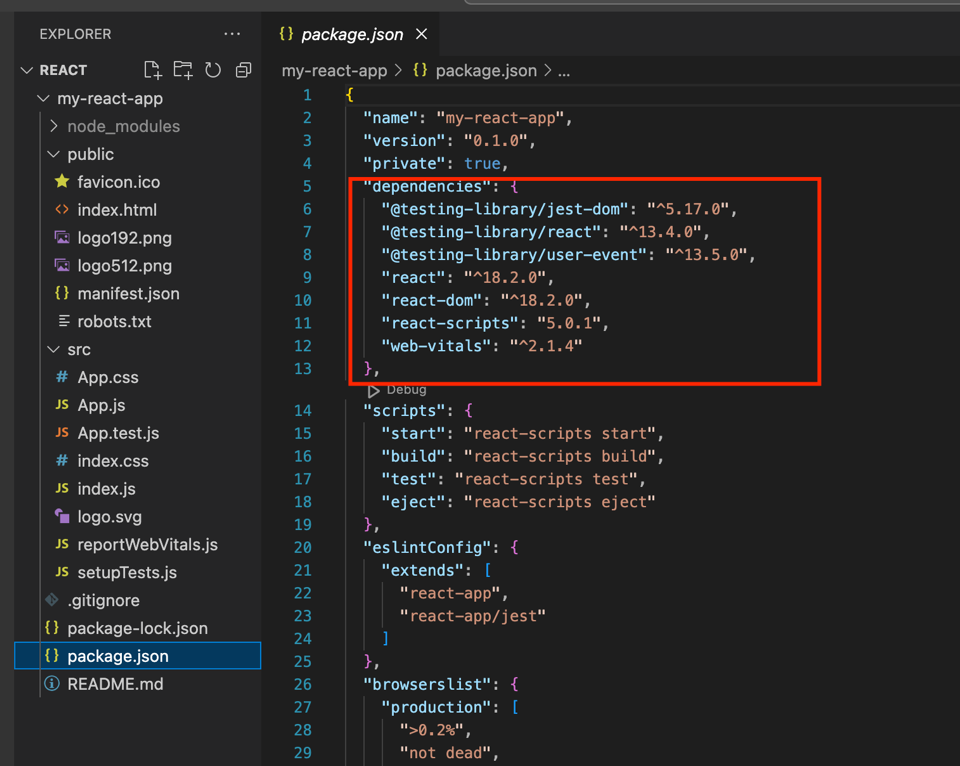
Project structure နဲ့ ပတ်သက်လို့ အောက်ပါအစိတ်အပိုင်းများကို အသေးစိတ်ဆွေးနွေးကြမှာ ဖြစ်ပါတယ်။ အဲဒီအစိတ်အပိုင်းတွေကတော့−

1. package.json file
2. node\_modules folder
3. src folder
4. public folder တို့ ဖြစ်ပါတယ်။



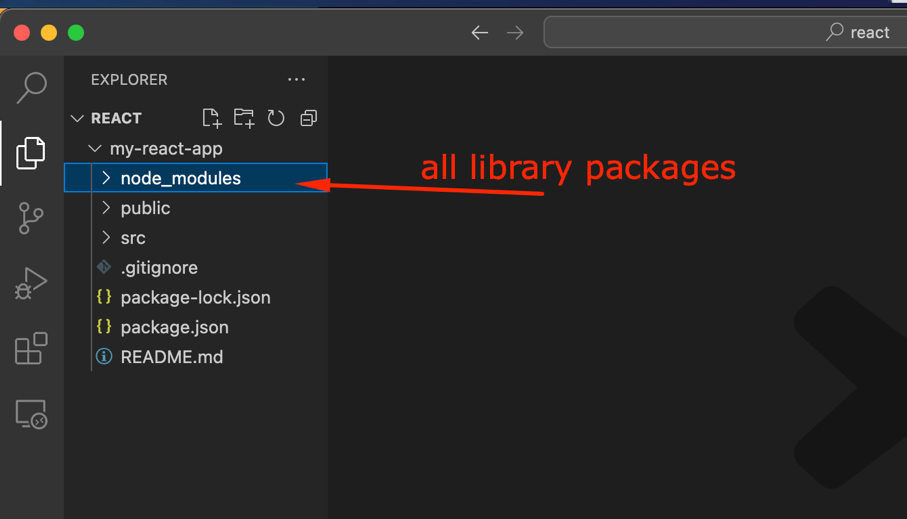
1. package.json file

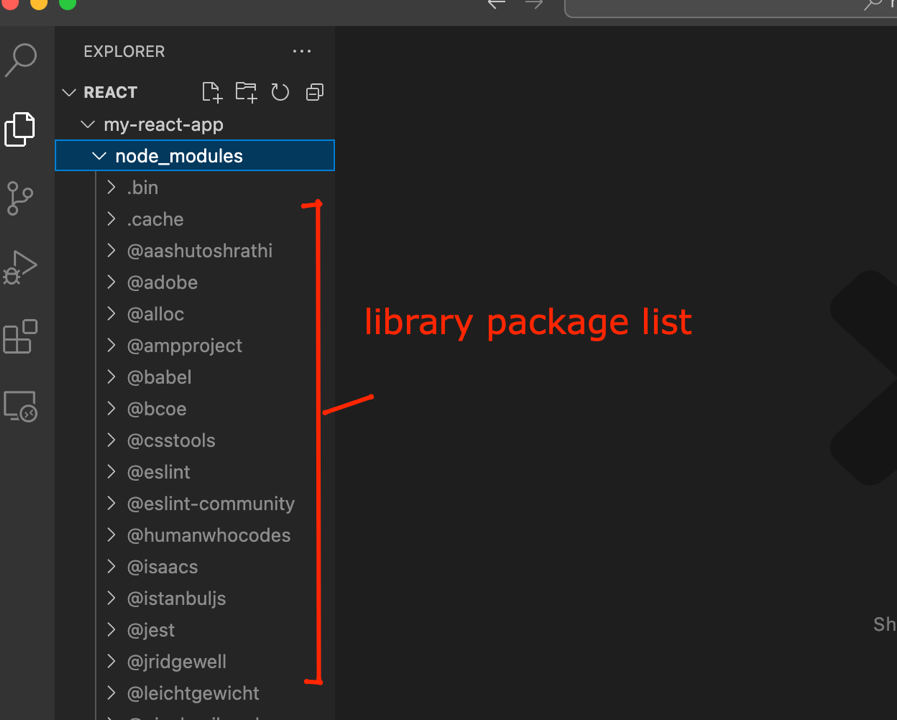
ကျနော်တို့ရေးမယ့် application မှာ သုံးမယ့် dependency library စာရင်းကို ဒီဖိုင်မှာ ကြည့်ရှု့နိုင်ပါတယ်။ ဒီ dependency library list ကို ကြည့်ပြီး လိုအပ်မယ့် library တွေထပ်ထည့်မယ်။ မလိုတော့တဲ့ library တွေကို ဖြုတ်ပလိုက်မယ်ပေါ့။



1. node\_modules folder

application project အတွက် လိုအပ်တဲ့ core library packages တွေအားလုံး ဒီ folder အထဲမှာ ရှိပါတယ်။





1. src folder

Application နဲ့ပါတ်သက်တဲ့ logic function မှန်သမျှ ဒီ folder ထဲမှာပဲ ရေးကြမှာ ဖြစ်ပါတယ်။ များသောအားဖြင့် application development အချိန်တွေကို ဒီ src folder ထဲမှာပဲ ကုန်ဆုံးရမှာ ဖြစ်ပါတယ်။

src / index.js

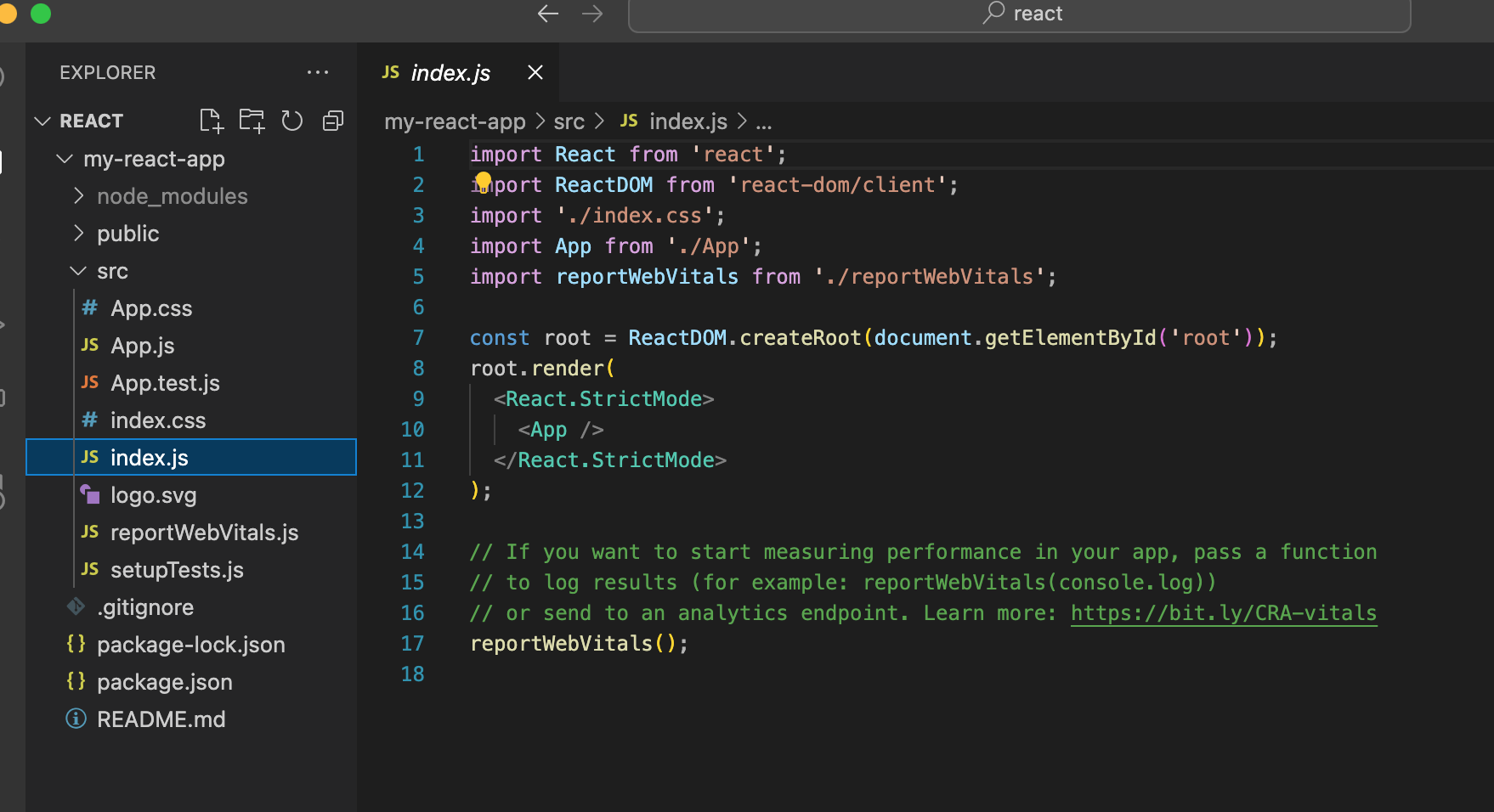
src folder ထဲက index.js file ကတော့ react application တစ်ခုလုံးရဲ့ စတင်တဲ့ နေရာဖြစ်ပါတယ်။

Virtual DOM မှာရှိနေတဲ့ logical structure JavaScript elements တွေကို render လုပ်ပေးတဲ့ file ဖြစ်ပါတယ်။ ကျောင်းသားတွေ မကြာခဏ မေးနေကြ မေးခွန်းကတော့ src/index.js file နဲ့ public/index.html နဲ့ ဘယ်လို ဆက်စပ်မှုရှိသလဲ ဆိုတာပါပဲ။ ဖြေရှင်းချက်ကတော့ node\_models folder ထဲက node\_modules/react-scripts/config/paths.js file ကို သွားဖွင့်ကြည့်မယ်ဆိုရင်

appHtml: resolveApp('public/index.html') cဆိုတဲ့ code ကို publicly accessible ဖြစ်အောင် သတ်မှတ်ထားတာ တွေ့ပါမယ်။

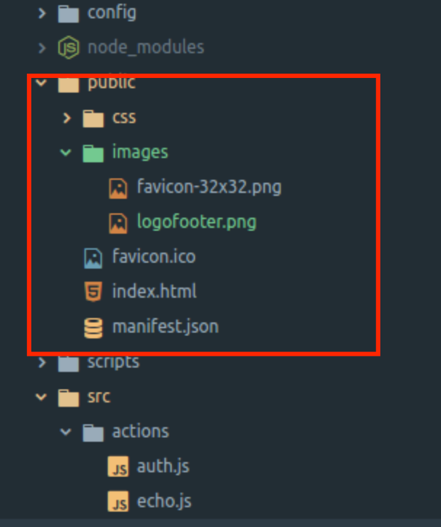
ဒီ code လေးက ကျနော်တို့ react application ကို စ run လိုက်တာနဲ့ entry point ဖြစ်တဲ့ index.js ဖိုင်က စပြီးအလုပ်လုပ်ပါတယ်။ index.js ဖိုင်က index.html ဖိုင်ကို ရှာပြီး output တွေကို index.html file မှာ render လုပ်ပေးပါတယ်။ node\_modules package ထဲမှာ config folder အထဲက path.js file က index.html file ကို application setup ထဲမှာ publicly accessible ဖြစ်အောင် သတ်မှတ်ရေးသားထားတဲ့အတွက် src/index.js file နဲ့ public/index.html file တွေက ဆက်စပ်ပြီး အလုပ်လုပ် နေရခြင်းဖြစ်ပါတယ်၊

src/index.js



1. public folder

public folder ထဲမှာတော့ static file တွေ၊ images တွေ ၊ CSS file တွေနဲ့ အခြားသော asset file တွေကို ထည့်ဖို့ ဖြစ်ပါတယ်။



public/index.html ( default structure)

React application တစ်ခုလုံးရဲ့ functional component တွေအာလုံးကို index.html file ထဲက root structure block ထဲမှာ index.js file ထဲက JavaScript ReactDOM code နဲ့ render လုပ်ပါတယ်။

How React render HTML

Index.js

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

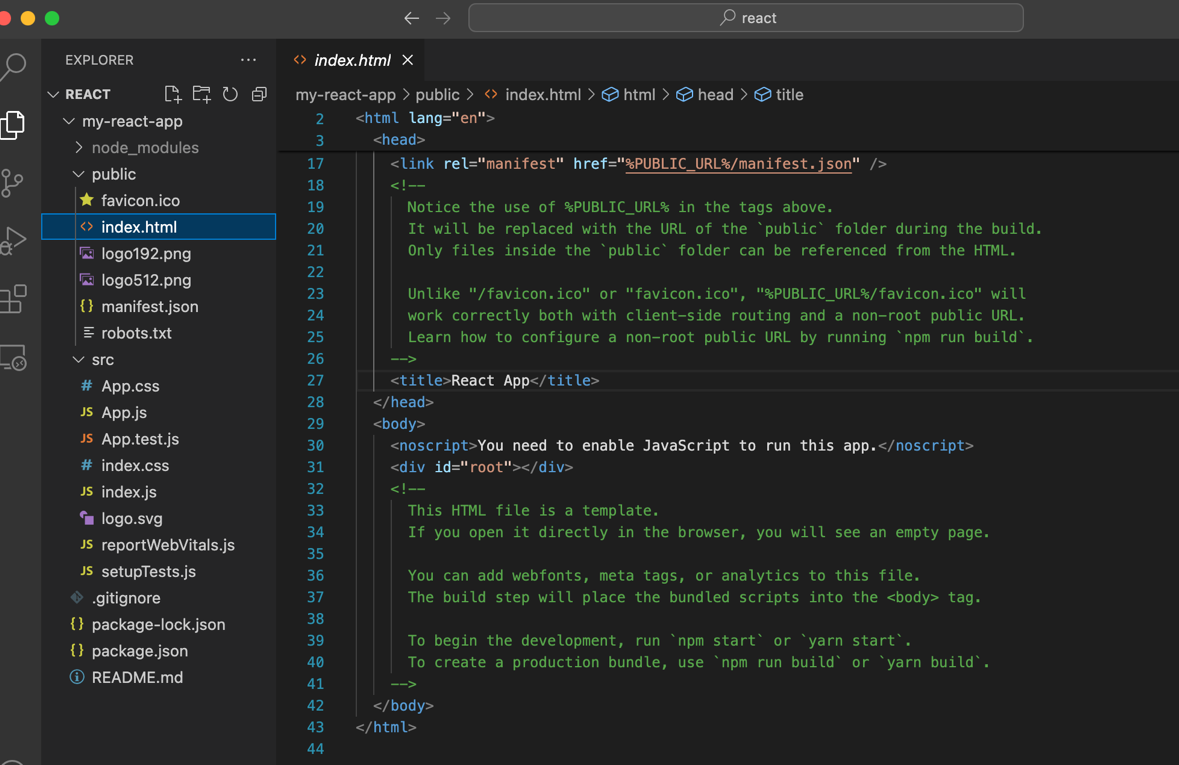
);

Index.html

<div id="root">

</div>

ဒီနေရာမှာ ReactDOM အကြောင်းနည်းနည်းပြောခြင်ပါတယ်။ Application project ဘယ်လောက်ပဲ ကြီးကြီး ၊ ဘယ်လောက်ပဲ သေးသေး





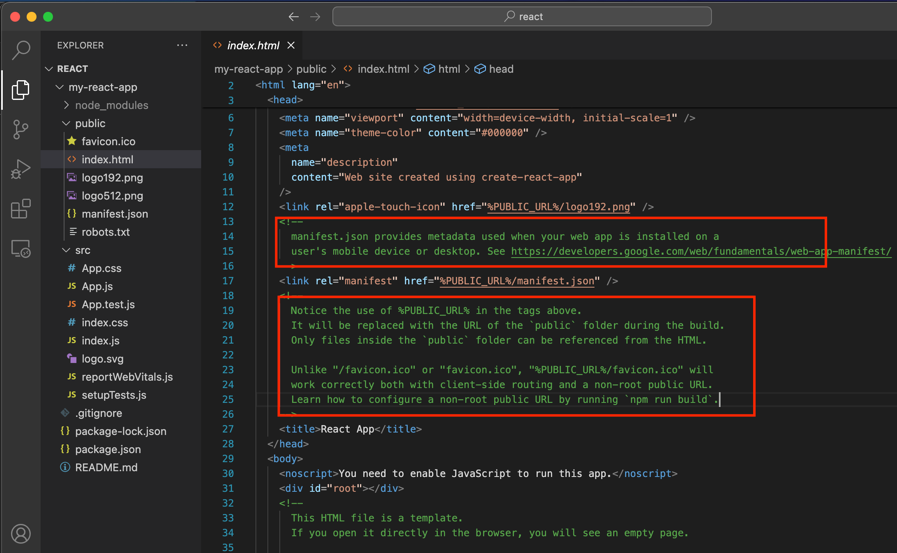
Lesson 3

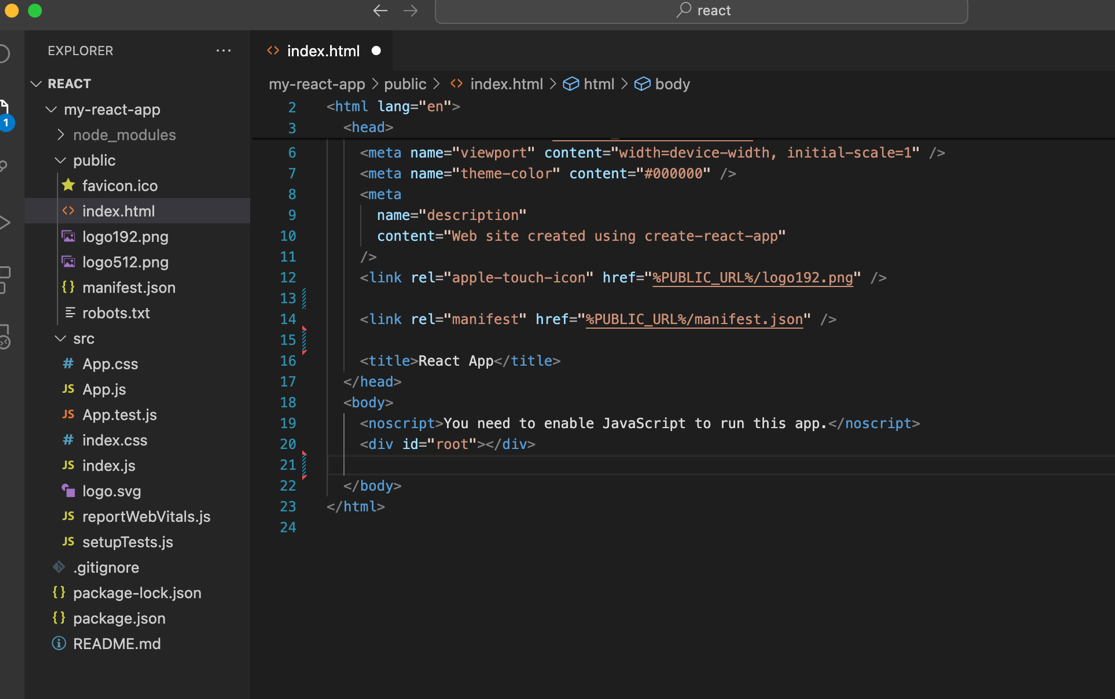
 Link:

File Structure & Code Optimization

React Application တစ်ခုရဲ့ file structure နဲ့ function relationships တွေကို ကောင်းမွန်စွာနားလည်နိုင်အောင် file structure နဲ့ code တွေကို cleaning လုပ်ပါမယ်။ ရိုးရှင်း ပြီး ဘာမှ မရှိတဲ့ project အနေအထားတစ်ခုကနေ complex project အနေအထားတစ်ခုသို့ရောက်အောင် တစ်ဆင့်ပြီးတစ်ဆင့် လေ့လာကြမှာ ဖြစ်ပါတယ်။

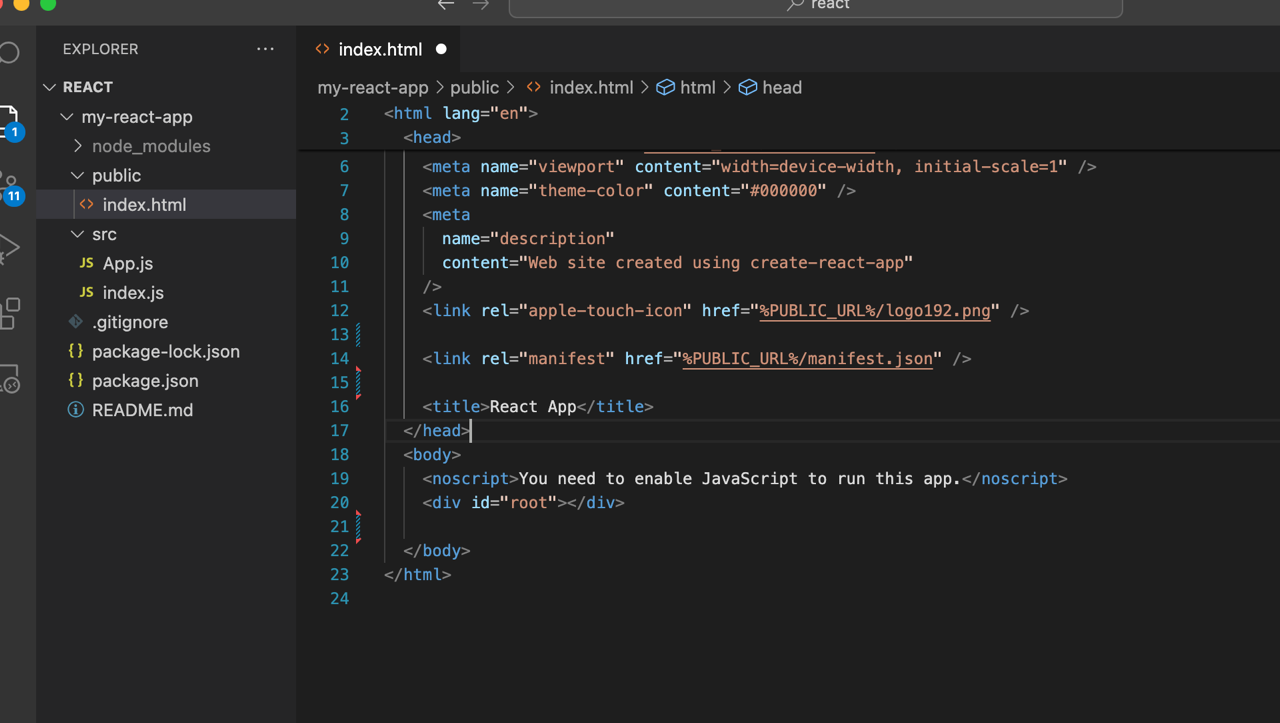
ဉီးစွာ Index.html file ထဲက မလိုအပ်တဲ့ comment အပိုင်းတွေကို ဖျက်လိုက်ပါမယ်။

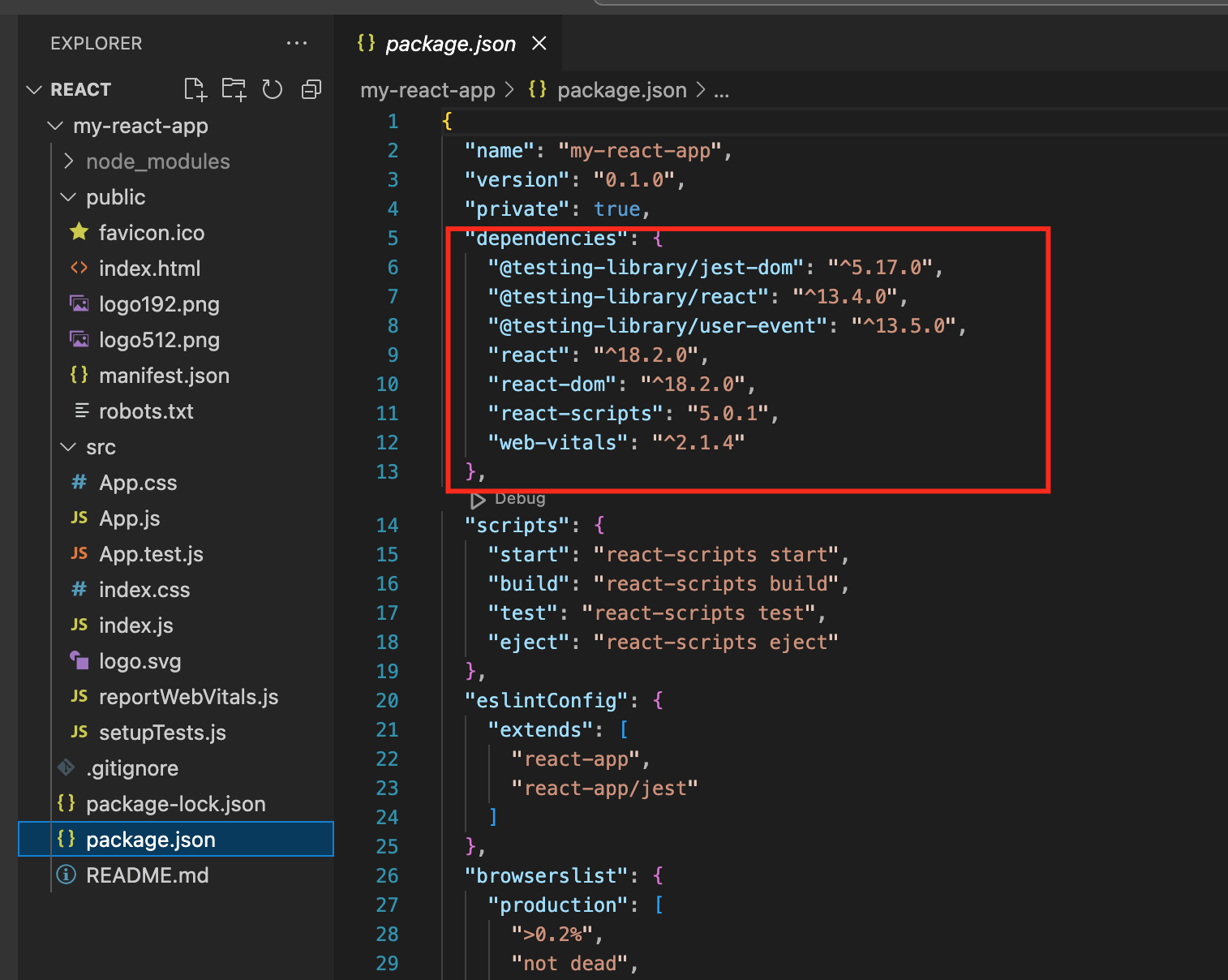




မလိုအပ်တဲ့ ဖိုင်တွေကို လည်း delete လုပ်လိုက်ပါ။ အောက်ပါအတိုင်း public folder ထဲမှာ index.html file ပဲ ရှိနေမယ်။ src folder ထဲမှာ app.js file နဲ့ index.js file ပဲ ကျန်ပါမယ်။

Before deleting files After deleting files





Index.js file ကို လည်း အောက်ပါအတိုင်း code cleaning လုပ်ပါမယ်။ React ဘယ်လိုအလုပ်လုပ်တယ်ဆိုတာကို ပိုမိုနားလည်အောင် file တွေ ၊ code တွေကို အခုလို ရှင်းလင်းရတာဖြစ်ပါတယ်။

Before deleting codes

import React from 'react';

import ReactDOM from 'react-dom/client';

import './index.css';

import App from './App';

import reportWebVitals from './reportWebVitals';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

// If you want to start measuring performance in your app, pass a function

// to log results (for example: reportWebVitals(console.log))

// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals

reportWebVitals();

After deleting codes

import React from 'react';

import ReactDOM from 'react-dom/client';

import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(

<React.StrictMode>

<App />

</React.StrictMode>

);

App.js

Before deleting codes

import logo from './logo.svg';

import './App.css';

function App() {

return (

<div className="App">

<header className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<p>

Edit <code>src/App.js</code> and save to reload.

</p>

<a

className="App-link"

href="https://reactjs.org"

target="\_blank"

rel="noopener noreferrer"

>

Learn React

</a>

</header>

</div>

);

}

export default App;

After deleting codes

function App() {

return (

<h1> React App </h1>

);

}

export default App;

Default Code တွေကို update လုပ်ပြီးရင် အောက်ပါအတိုင်း ပြန် run ကြည့်ပါ။

cd my-react-app

my-react-app> npm start

Compiled successfully!

You can now view my-react-app in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.100.41:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

webpack compiled successfully

Output:



Congratulation ပါ။ File structure နဲ့ code optimization အောင်မြင်သွားပါပြီ။ ဆက်ပြီးတော့ React Component အကြောင်းတွေကို လေ့လာကြပါမယ်။

 Lesson 4

 Link:

React Components

Example 1

src/app.js

const Item=()=>{

return (

<ul>

<li>Category :Coffee </li>

<li>Name :Americano </li>

<li>Price :3000 </li>

</ul>

);

}

const App=()=>{

return (

<>

<h1>React App ( Component Example 1 ) </h1>

<hr/>

<Item/>

<Item/>

<Item/>

</>

);

}

export default App;

Example 2

- src

- - Header.jsx

- - Item.jsx

- - Footer.jsx

- - App.js

Output:

import Header from './Header'

import Item from './Item'

import Footer from './Footer'

const App=()=>{

return (

<>

<Header/>

<hr/>

<Item/>

<hr/>

<Footer/>

</>

);

}

export default App;

App.js

Header.jsx

const header=()=>{

return (

<h1>React App ( Component Example 2 ) </h1>

);

}

export default header;

const Item=()=>{

return (

<>

<ul>

<li> Category : Coffee </li>

<li> Name : Americano </li>

<li> Price : 3500 Kyats </li>

<li> <button>Order Now</button> </li>

</ul>

</>

);

}

export default Item;

Item.jsx

const Item=()=>{

return (

<>

<ul>

<li> Category : Coffee </li>

<li> Name : Americano </li>

<li> Price : 3500 Kyats </li>

<li> <button>Order Now</button> </li>

</ul>

</>

);

}

export default Item;

Footer.jsx

 Lesson 4

 Link:

Props

Example 1

Output:

const Item=(props)=>{

return(

<>

<ul>

<li> Category : {props.category} </li>

<li> Name : {props.name} </li>

<li> Price : {props.price} </li>

</ul>

</>

);

}

const App=()=>{

return (

<>

<Item category='coffee' name='Americano' price={3500} />

<Item category='coffee' name='Expresso' price={4000} />

<Item category='juice' name='Apple' price={2000} />

</>

);

}

export default App;

App.js

Project files

- src

- - App.js

- - Student.jsx

Example 2

Output:

🖳 Codes:

import Student from './Student'

const App=()=>{

return (

<>

<Student name={'Mg Mg'} age={18} isStudent={true} />

<Student name={'Soe Soe'} age={38} isStudent={false} />

<Student name={'Moe Moe'} age={21} isStudent={true} />

</>

);

}

export default App

App.js.

const Student=(props)=>{

return(

<>

<ul>

<li> Name : {props.name} </li>

<li> Age : {props.age} </li>

<li> Student : {props.isStudent ? "Yes" : "No"} </li>

</ul>

</>

);

}

export default Student

Student.jsx

 Lesson 4

 Link:

Conditional Rendering

Project files

- src

- - App.js

- - Greetings.jsx

Example 1

If isLogin is true

Output:

If isLogin is false

Output:

Source Codes:

import Greetings from './Greetings'

const App=()=>{

const login=false;

const username="Northern City"

return (

<>

<Greetings isLogin={login} name={username} />

</>

);

}

export default App

App.js

const Greeting=(props)=>{

if(props.isLogin)

return <h1> Welcome {props.name} </h1>

return <h1> Please Login to continue </h1>

}

export default Greeting

Greetings.jsx

const Greeting=(props)=>{

const welcome\_prompt=<h1> Welcome {props.name} </h1>

const login\_prompt=<h1> Please Login to continue </h1>

return (props.isLogin ? welcome\_prompt : login\_prompt);

}

export default Greeting

Update 1

import PropTypes from 'prop-types'

const Greeting=(props)=>{

const welcome\_prompt=<h1> Welcome {props.username} </h1>

const login\_prompt=<h1> Please Login to continue </h1>

return (props.isLogin ? welcome\_prompt : login\_prompt);

}

Greeting.proptype={

isLogin : PropTypes.bool,

username: PropTypes.string,

}

Greeting.defaultProps={

isLogin:false,

username:"Guest",

}

export default Greeting

Update 2

Updated Output:

🖎 Notes:

Application တွေကြီးထွားလာသည်နှင့်အမျှ typechecking နဲ့ bug တွေကို ကျနော်တို့ ရှင်းလင်းကြပါတယ်။ အချို့သော application တွေမှာ type checking ပြုလုပ်ရန် Flow သို့မဟုတ် TypeScript ကဲ့သို့သော JavaScript extension များကို အသုံးပြုကြတယ် ။ ဒါပေမယ့် အဲဒါတွေကို အသုံးမပြုရင်တောင် React မှာ build-in typechecking ရှိပါတယ်။ အစိတ်အပိုင်းတစ်ခုအတွက် props များပေါ်မှာ typechecking လုပ်ဆောင်ဖို့ propTypes ကို အသုံးပြုကြခြင်းဖြစ်ပါတယ်။

React Component default values တွေကို ကြိုတင် သတ်မှတ်ချင်တဲ့အတွက် defaultProps method ကို အသုံးပြုထားတာ ဖြစ်ပါတယ်။

 Lesson 5

 Link:

List

Project files

- src

- - App.js

- - List.jsx

Example 1.

Sample Output

import List from './List'

const App=()=>{

return (

<>

<List />

</>

);

}

export default App

App.js

const List=()=>{

const fruits=["Apple","Orange","Banana","Coconut","Mango"];

const list\_items=fruits.map((fruit)=><li>{fruit}</li>)

return (<ul> {list\_items} </ul>);

};

export default List

List.jsx

const List=()=>{

const fruits=["Apple","Orange","Banana","Coconut","Mango"];

fruits.sort() // or fruit.reverse()

const list\_items=fruits.map((fruit)=><li>{fruit}</li>)

return (<ul> {list\_items} </ul>);

};

export default List

List.jsx (updated)

Sorted output:

Example 2.

Project Files

- src

- - App.js

- - List.jsx

Sample Output:

import List from './List'

const App=()=>{

return (

<>

<List />

</>

);

}

export default App

App.js

const List=()=>{

const fruits=[ {"id":1,"name":"Apple","price":1000},

{"id":2,"name":"Orange","price":1200},

{"id":3,"name":"Banana","price":2000},

{"id":4,"name":"Coconut","price":3000},

{"id":5,"name":"Mango","price":2500},

{"id":6,"name":"Pineapple","price":3000},

{"id":7,"name":"Watermelon","price":4000}

];

//display all friuts

const list\_items=fruits.map((fruit)=><li **key={fruit.id}**>{fruit.name} ({fruit.price})</li>)

return (<ul> {list\_items} </ul>);

};

export default List

List.jsx

const List=()=>{

const fruits=[ {"id":1,"name":"Apple","price":1000},

{"id":2,"name":"Orange","price":1200},

{"id":3,"name":"Banana","price":2000},

{"id":4,"name":"Coconut","price":3000},

{"id":5,"name":"Mango","price":2500},

{"id":6,"name":"Pineapple","price":3000},

{"id":7,"name":"Watermelon","price":4000}

];

//Alphabetical Sort

fruits.sort((a,b)=>a.name.localeCompare(b.name))

//display all friuts

const list\_items=fruits.map((fruit)=><li key={fruit.id}>{fruit.name} ({fruit.price})</li>)

return (<ul> {list\_items} </ul>);

};

export default List

List.jsx

( Sort by Name )

🖳 Output:

//Alphabetical Sort

//fruits.sort((a,b)=>a.name.localeCompare(b.name))

//Alphabetical Reverse Sort

//fruits.sort((a,b)=>b.name.localeCompare(a.name))

//Numeric Sort by id

//fruits.sort((a,b)=>a.id - b.id)

//Numeric Sort by id reverse

//fruits.sort((a,b)=>b.id - a.id)

//Expensive Fruits (price is greater than 2000)

//const expensiveFruits=fruits.filter((f)=>f.price>2000)

//Cheap Fruits (price is less than or equal to 2000)

//const cheapFruits=fruits.filter((f)=>f.price<=2000)

List.jsx

( Try all sorting )

Project Files

- src

- - App.js

- - List.jsx

Example 3.

Sample Output:

const List=(props)=>{

const category=props.category

const items=props.items;

const list\_items=items.map((item)=> item.category===category ? <li key={item.id}>{item.id} - {item.name} </li>: null )

return (<ul>

<h1>{category}</h1>

{list\_items}

</ul>);

};

export default List

List.jsx

import List from './List'

const App=()=>{

const items=[ {"id":1,"name":"Apple","price":1000,"category":"fruit"},

{"id":2,"name":"Carrot","price":1200,"category":"vegetable"},

{"id":3,"name":"Orange","price":1200,"category":"fruit"},

{"id":4,"name":"Banana","price":2000,"category":"fruit"},

{"id":5,"name":"Coconut","price":3000,"category":"fruit"},

{"id":6,"name":"Potato","price":1500,"category":"vegetable"},

{"id":7,"name":"Mango","price":2500,"category":"fruit"},

{"id":8,"name":"Pineapple","price":3000,"category":"fruit"},

{"id":9,"name":"Watermelon","price":4000,"category":"fruit"},

{"id":10,"name":"Tomato","price":1000,"category":"vegetable"}

];

return (

<>

<List items={items} category="fruit" />

<List items={items} category="vegetable" />

</>

);

}

export default App

App.js

✍ Notes:

Ternary operator အကြောင်းနည်းနည်းပြောပြချင်တယ်၊ သာမှန် conditional statement ပုံစံ ကို single line condition ပုံစံအဖြစ် ပြောင်းလဲ ထားခြင်းဖြစ်ပါတယ်။

Traditional Conditional Statement

Syntax:

If(con){

T;

}

else{

F;

}

con. ? T : F;

ကျနော်တို့ရေးထားတဲ့ အောက်ပါ source code ကို analysis လုပ်ကြည့်ရအောင်

🖳 Code:

const list\_items= items.map(

(. item) =>

item.category===category ?

Ternary operator

<li key={item.id}>{item.id} - {item.name} </li>

:

null

)

အခုနောက်ပိုင်း developer တွေ ternary operator ပုံစံကို logical operator နဲ့ ပြောင်းလဲ ရေးသားတဲ့ ပုံစံကို အောက်ပါအတိုင်းမြင်ရမယ်။

con ? True Block : False Block con && True Block

လေ့ကျင့်ခန်းထဲက ternary operator ပုံစံကို အောက်ပါအတိုင်း logical and ပုံစံ ပြောင်းရေးပြီး output ကို ကြည့်ကြည့်ပါ။ ရလဒ် အတူတူဖြစ်နေတာကို တွေ့ရမှာ ဖြစ်ပါတယ်။

🖳 Code:

const list\_items= items.map(

(. item) =>

item.category===category &&

<li key={item.id}>{item.id} - {item.name} </li>

)

Component Data type checking နဲ့ default data initialization အကြောင်းဆက်ပြောပြချင်တယ်။ Data Type Checking အတွက် အောက်ပါအတိုင်း ရေးသားနိုင်ပါတယ်။

**🖳 Code:**

List.propTypes={

category: PropTypes.string,

items:PropTypes.arrayOf(

PropTypes.shape({ id:PropTypes.number,

name:PropTypes.string,

price: PropTypes.number,

}

)

)

}

Type Checking Test

const items=[ {"id":1,"name":"Apple","price":"10", "category":"fruit"},

{"id":2,"name":"Carrot","price":1200,"category":"vegetable"},

{"id":3,"name":"Orange","price":1200,"category":"fruit"},

{"id":4,"name":"Banana","price":2000,"category":"fruit"},

{"id":5,"name":"Coconut","price":3000,"category":"fruit"},

{"id":6,"name":"Potato","price":1500,"category":"vegetable"},

{"id":7,"name":"Mango","price":2500,"category":"fruit"},

{"id":8,"name":"Pineapple","price":3000,"category":"fruit"},

{"id":9,"name":"Watermelon","price":4000,"category":"fruit"},

{"id":10,"name":"Tomato","price":1000,"category":"vegetable"}

];

Error Report:

Data Type checking ပြီးသွားရင် တော့ Data type initialization ကို ရေးကြပါမယ်။

🖳 Code:

List.defaultProps={

category:"Category",

items:[],

}

Items list ကို empty လုပ်ပြီး အောက်ပါ output ကို ရအောင် try ကြည့်ပါ။

Spread Operator

Coding ရေးတဲ့အခါ Array data တွေကို မကြာခဏ ပေါင်းစည်းခြင်း၊ ဖြတ်တောက်ခြင်း ၊ ပြုပြင်ခြင်းတွေကိုလုပ်ရပါတယ်။ ရိုးရိုး Array method တွေဖြစ်တဲ့ push ၊ pop ၊ splice ၊ slice တွေအပြင် spread operator ( … ) တွေကို လည်း အကျွမ်းတစ်ဝင်ဖြစ်နေဖို့ လိုပါတယ်။ အောက်ပါ Example တွေကို လိုက်လုပ်ပြီး လေ့လာကြည့်ကြရအောင်နော်။

Example 1

Array Source:

const student\_list = [

{"id":1,"name":"mg mg","address":"hlaing"},

{"id":2,"name":"su su","address":"innsein"},

{"id":3,"name":"moe moe","address":"sanchaung"},

{"id":4,"name":"thu zar","address":"la tha"},

];

Display All Students:

Console.log([…student\_list])

Display All Data:

student\_list.forEach((s)=>{

console.log('---------------')

console.log(`Id : ${s.id} `)

console.log(`Name : ${s.name} `)

console.log(`Address : ${s.address} `)

console.log('---------------')

})

Add New Object into student list

Syntax:

[...array, new\_object ];

Code:

const new\_student={

"id":3,

"name":"aung aung",

"address":"san chaung"

}

const new\_arr=[...student\_list,new\_student];

console.log("new arr = ",new\_arr)

Output:

Before adding new student

After adding new student

Update Array Object by Id

Syntax:

{…array,…update\_object}

Code :

const update\_student={

"id":1,

"name":"mg mg",

"address":"san chaung update"

}

const new\_arr=student\_list.map((s)=>{

if(s.id===1){

return {...s,...update\_student}

}

else{

return s;

}

})

Output:

Delete and object from array

Syntax:

array.filter((obj)=> obj.id!==delete\_id);

Code:

let new\_arr=student\_list.filter((s)=> s.id!==1);

Output:

လေ့ကျင့်ရန် −

Hints:

Array Source:

const student\_list = [

{"id":1,"name":"mg mg","address":"hlaing"},

{"id":2,"name":"su su","address":"innsein"},

{"id":3,"name":"moe moe","address":"sanchaung"},

{"id":4,"name":"thu zar","address":"la tha"},

];

New Object:

const newStudent={

"id":5,

"name":"tun tun",

"address":"innsein new"

}

Object to be updated:

const updateStudent={

"id":2,

"name":"su mon",

"address":"innsein update"

}

Display Data:

const StudentList=({students})=>{

return (

<>

<ul>

{students.map(s => (

<li key={s.id}>

{s.id} - {s.name} ( {s.address} )

</li>

))}

</ul>

</>

);

}



Lesson 6

 Link:

Click Event

Project Files

- src

- - App.js

- - Button.jsx

Example 1

Output

import Button from './Button'

const App=()=>{

return (

<>

<Button name={'Click Me'}/>

</>

);

}

export default App

App.js

function Button(props){

let count=0;

const handleClick= ()=> {

count++;

console.log(`you click button 1 - ${count} times `)

}

return (<button onClick={handleClick}> {props.name} </button>)

}

export default Button

Button.jsx

⮳ Single click

🖳 Code:

🗎 Button.jsx

function Button(props){

let count=0;

const handleClick= (e)=> {

count++;

count%2===0 ? e.target.textContent="Single Click!" : e.target.textContent=`${props.name}`

}

return (<button onClick={(e)=>handleClick(e)}>{props.name} </button>)

}

export default Button

🗎 App.js

import Button from './Button'

const App=()=>{

return (

<>

<Button name={'Click Me'}/>

</>

);

}

export default App

⮳ Double click

🖳 Code :

function Button(props){

let count=0;

const handleClick= (e)=> {

count++;

count%2===0 ? e.target.textContent="Double Click!" : e.target.textContent=`${props.name}`

}

return (<button onDoubleClick={(e)=>handleClick(e)}>{props.name} </button>)

}

export default Button

Button.jsx

Lesson 7

 Link:

useState

Example 1

Increase and decrease counter

Sample Output:

-src

- - App.js

- - Button.jsx

- - CountDisplay.jsx

Project Files

🖳 Code:

import CountDisplay from './CountDisplay'

import { useState } from 'react';

const App=()=>{

const [counter,setCounter]=useState(0)

const clickHandle1=(prev)=>{

prev=prev-1

setCounter(prev)

}

const clickHandle2=(next)=>{

next=next+1

setCounter(next)

}

return (

<>

<button onClick={()=>clickHandle1(counter)}> Decrease Count </button>

&nbsp;

<button onClick={()=>clickHandle2(counter)}> Increase Count </button>

<CountDisplay count={counter}/>

</>

);

}

export default App

App.js

🗎 CountDisplay.jsx

import { useState } from 'react';

const CountDisplay=({ count }) => {

const [prevCount, setPrevCount] = useState(count);

const [trend, setTrend] = useState(null);

if (prevCount !== count) {

setPrevCount(count);

setTrend(count > prevCount ? 'increasing' : 'decreasing');

}

return (

<>

<h1>{count}</h1>

{trend && <p>The count is {trend}</p>}

</>

);

}

export default CountDisplay

Example 2

Project Files

- src

- - App.js

- - Item.jsx

- - Item.css

Output

App.js

import Item from "./Item"

import { useState } from "react"

import "./Item.css"

const App=() => {

const [fav,setFav]=useState(1)

const items= [

{id:1,name:"Americano",price:3000},

{id:2,name:"Expresso",price:4000},

{id:3,name:"Latte",price:3200}

];

const handleFav=(id)=>{

setFav(id)

}

return (

<>

<h1>Northern City Cafe </h1>

<hr/>

<div className="row">

{

items.map((item) =>

<Item key={item.id} onFavorite={handleFav} favorite={fav===item.id} id={item.id} name={item.name} price={item.price} />

)

}

</div>

</>

)

}

export default App

Item.jsx

import "./Item.css"

import { useState } from "react"

const Item=(props)=>{

const [likes,setLike]=useState(0)

const handleLike=()=>{

const newLikes=likes+1

setLike(newLikes)

}

const handleDislike=()=>{

const newLikes=likes-1

setLike(newLikes)

}

const handleFav=()=>{

props.onFavorite(props.id)

}

return (

<>

<div className="card">

<h1>{props.name} </h1>

<p> price: {props.price} <br/><br/>

Likes: {likes} <br/>

Favorite : {props.favorite ? "YES" : "NO"} </p>

<p className="cta">

<span className="like-btn" onClick={handleLike} >like </span>

<span className="dislike-btn" onClick={handleDislike}>dislike </span>

<span className="fav-btn" onClick={handleFav}>favorite </span>

</p>

</div>

</>

)

}

export default Item

Item.css

.row{

display: flex;

flex-direction: row;

justify-content: start;

align-items: center;

gap:20px;

}

.card{

width:200px;

display: flex;

flex-direction: column;

justify-content: start;

align-items: center;

padding:20px;

border-radius: 20px;

border:1px solid gray;

}

.cta{

display: flex;

flex-direction: row;

justify-content: center;

align-items: center;

gap:10px;

}

.like-btn,.dislike-btn,.fav-btn{

padding:5px;

border:1px solid black;

border-radius: 10px;

}

.like-btn:hover,.dislike-btn:hover,.fav-btn:hover{

padding:5px;

border:1px solid rgb(249, 66, 16);

border-radius: 10px;

color:rgb(251, 251, 251);

cursor: pointer;

background-color: black;

}

Forms