**React Installation**

1. Next.js is a framework of react and react is promoting to shift there for production purposes.

2. Now we can not setup react like old times. Now have to use VITE, a powerful tool to install react.

3. Installation steps:



Steps: npm create vite > React > Just JS

Shortcut: rsc

**React facts**

1. React is made to make Developer life easy. Not performance Optimization.
2. React is a javascript object
3. React creates a separate dom which is known as virtual DOM
4. JSX simplifies react. Like:

const element = React.createElement("div", null,

[React.createElement("p",null, "hello world"),

React.createElement("p",null, "hello world"),

React.createElement("p",null, "hello world")]);

Simplified:

<div> <p> hello world </p> <p> hello world </p> <p> hello world </p> </div>

1. Babel transpiles XML into Javascript
2. State is an important thing in React that helps to simplify process through state variable.
3. State is an array of var and function. So, in [var, function], function helps to update the value of var
4. 2 types of rendering – Client side rendering and server side rendering

**Virtual DOM**

DOM is fast but the process of showing the change of DOM is slow.



To make it fast, we can do:

1. Optimize Batch update: As an example, for a 10000 time loop, if we do batch update (div.innerHTML=value) in every loop, then it gets much slower. But after the loop, if we do the batch update, then it takes no time. So, without, virtualDOM, we can optimize batch update to make the site faster.

2. Less DOM operation

Here comes Virtual DOM. It acts like a rough DOM sheet to use reconciliation algorithm to do the exact change and make the operation much faster.



Example:



Here, when we insert gggggg, the normal DOM changes every element. But Virtual DOM only changes the element after gggggg or the necessary parts by comparing.

\*\*Virtual DOM compares between previous UI and new UI and render only the changed part, not the whole page.

**Start with react**

1. Run with vite: npm run dev / npx vite
2. Build react project: npm run build / npx vite build

\*vite combines the whole scattered project and centralize them for deployment

**File Description and Components**

We are creating single page application using react. That means, only one index.html. And single page aplication loads only once.

vite.config.js – Auto generated file. Nothing to change in general

**package.json** – All used package and the version with every description

* dependencies packages will go for final production not devdependencies
* when downloading the node modules, the files are downloaded according to the dependencies and devdependencies and also auto-generates package-lock.json

package-lock.json – nothing to change

index.html – the whole application stays inside the div of root. And it loads only once. So, performance enhances.

.gitignore – the things that won’t be pushed into the github

**Src** – The whole application stays. The main place to work.

* Main.jsx: is the starting file to run the react app
* Index.css: auto generated
* App.jsx: this is connected with the main.jsx. There is a function
* App.css: auto generated
* Assets: auto generated

Public – by default vite logo

Node\_module - All dependency package files installed, nothing to work with

dist – nothing to work on

Organize react files

1. Remove index.css, App.css with the link in main.jsx
2. App.jsx:

function App() {

  return (

    <>

    <h1>React App</h1>

    </>

  )

}

export default App

1. In assets, delete react.svg and create css and images folder
2. In scr, create pages and component folder

VS code extentions:

1. Auto close tag
2. Auto Import - ES6, TS, JSX, TSX
3. Auto Rename Tag
4. ESLint – for auto correction and suggestion
5. npm Intellisense – help to import from node modles
6. Path Intellisense – auto path suggestion
7. Postman – simulate rest api
8. Prettier
9. Reactjs code snippets – provide shortcut ex: rsc, rcc
10. Snipped – screenshot – ctrl+shift+p
11. Stylelint – fix css mistakes
12. Tailwind CSS IntelliSense – suggests css class
13. Thunder client – same like postman
14. VSCode React Refactor – helps to put something in variable or function
15. vscode-icons

Newly created files:

Inside src/

/assets/css, /assets/images

/component

/pages

So, basic file structure in src –



Difference between html and **JSX**:

1. we can directly use js inside, {}
2. Always return a single parent element. Example: everything inside <div><div/>, <span><span/> etc.
3. There is a self close option for tags. Ex: <img/>, which is not in html
4. Here, we have to use ‘className’ instead of ‘class’ in html
5. All attribute has to be camel case. In html, onclick. But in JSX, onClick
6. We used inline css in html(style=”color:red;”). But we have to make it object here(style={{color:’red’}}).

JSX essential:

1. Inline if else –

      {marks>80?<h1>Brilliant Result</h1>:<h1>Average result</h1>}

1. Immediately invoke (this execute immediately) –

        {(()=>{

          //Write any js code. But return it to show, as this is a function.

        })()}

1. Loop –

          {

            arr.map((item,i)=>{

              return<li key={i.toString()}>{item}</li>

              //key is to catch the iteration number

            })

          }

Here, we are using .map instead of for or while loop because .map returns something. Where for and while just executes codes.

1. **Conditional rendering** (after login, we will see logout. Vice-versa) –

We can do this by using a. if else, b. switch statement, c. ternery operator, d. logical &&, e. immediately invoke function

1. if else 🡪 problem: expensive, as code is too much

  if (status==true){

    return (

      <div>

        <h1>Login Status</h1>

        <button>Logout</button>

      </div>

    );

  }

  else{

    return (

      <div>

        <h1>Logout Status</h1>

        <button>Login</button>

      </div>

    );

  }

Because of (problem: expensive, as code is too much) the problem, we have to reduce code. So –

const LoginStatus=(status)=>{

    if (status){

        return <button>Logout Btn</button>

    }

    else{

        return <button>Login Btn</button>

    }

}

const CDRen\_ifelse = () => {

    return (

        <div>

            <h1>Login status</h1>

            {LoginStatus(false)}

        </div>

    );

};

export default CDRen\_ifelse;

1. switch 🡪

const header = ()=>{

    return <h1>Login status(switch)</h1>

};

const CDRen\_switch = () => {

    const status = true;

    switch(status){

        case true:

            return (

                <div>

                    {header()}

                    <button>Logout</button>

                </div>

            )

        case false:

            return (

            <div>

                {header()}

                <button>Login</button>

            </div>)

        default:

            return null

    }

};

export default CDRen\_switch;

1. ternary 🡪

import React from 'react';

const CDRen\_ternary = () => {

    let status = true

    return (

        <div>

            <h1>Login status(Ternary)</h1>

            {

                status?

                <button>Logout</button>

                :

                <button>Login</button>

            }

        </div>

    );

};

export default CDRen\_ternary;

1. logical &&(only one condition and clean) 🡪

import React from 'react';

const CDRen\_ternary = () => {

    let status = true

    return (

        <div>

            <h1>Login status(Ternary)</h1>

            {

                status?

                <button>Logout</button>

                :

                <button>Login</button>

            }

        </div>

    );

};

export default CDRen\_ternary;

1. Invoke function(the main moto is conditional rendering. Executes by its own after initiating “{(()=>{})()}” ) 🡪