**Index**

1. What is react?
2. Advantages of react.
3. How does react work?
4. Setup react project
5. Folder structure.
6. Create a component.
7. Style component.
8. Component export import.
9. Props properties
10. Default properties
11. Handle events in react.
12. If else
13. Ternary operator
14. Condition with logical operator
15. How to use if else inside JSX
16. Hook
17. State

* useState

1. how to read value from input
2. Map function in react.
3. Life cycle of a component
4. useEffect function.
5. Load component on button – conditional rendering and comp mounting and unmounting.
6. Getting data from API.
7. Project
8. Usecontext and contextAPI in react
9. useRef Hook
10. Routing in React
11. Form & Complex State Handling in React

**1. React**

React is a free open-source frontend JavaScript library for building interactive user interfaces and single-page applications based on components.

Library is nothing but a collection of some pre-written codes

HTML for creating content

CSS for style content.

JavaScript for generating dynamic content/ generating live content/ for events also.

**2. Advantages of using react**

There are the following advantages

Increase performance.

Decrease the line of code.

Faster development.

**3. How does react work- react concept**

* Component-based development library.
* Component is a piece/ part of UI- theory wise.
* Create components separately.
* Every component has its own/separate HTML (JSX), CSS, and JS.



* Component reusability.
* Components can be used again on the same page and again but with different data.
* Components are not dependent on each other

**4. Setup react project**

**Node:**

(Provides)It is a Java script environment/runtime.

Js runtime = running environment for JS.

**Note:** run JavaScript anywhere on PC without using any HTML.

Brower by default running environment for JavaScript.

Note: if we want to run JavaScript outside the browser, we need node JS.

Download and install node

>node –version, node -v

**NPM (Node Package Manager):**

There are a lot of libraries, and react is one of them.

NPM is a tool that you get when you install node js to help you install, uninstall, and manage libraries in your project.

**NPX (Node Package eXecute):**

Sometimes don’t you want to download the tool and directly use the library, then use npx. Npx executes the package from the internet without even downloading to the pc

**Create React project**

>cd react project

>npm create-react-app myapp

>cd myapp

>npm start

>ctrl+c

>npm run build

Note: vite, create-react-app are js build tool help to install js base project.

These are built tool to create maintain, run/compile project

**5. Folder structure**

**Package.json: -** just an overview of libraries

It is the heart of any JS project.

The configuration file of any js project.

Keep track of project information.

**Package-Lock \_JSON:**

Extended version of pakage. json

Information about dependencies/libraries in details

**Node Module:** contains dependencies

All dependencies mentioned in package.js are present in the node module folder.

Libraries are also called modules.

Modules are simply nothing but collections of function.

The library is nothing but a collection of modules

>delete node module folder

>npm install

**Public (Index.htm):**

Note: The public folder keeps assets.

Two components have a single js file.

**SRC Folder:**

Contains components.

Component is nothing but a function returning some html that’s all – coding wise.

**Guideline for component**

1 - Comp file name should be started with capital & follow camel case.

2 - To create component function name should be first capital letter follow camel case convention so that is can be identified as a component.

3 - The component name should be as component file name.

4 - To differentiate between component and element we start component name with capital letter.



**6. Creating Component**

A component is nothing but a function returning some html that’s all – coding-wise.

Write it as a function and load it as a tag like <APP/>

import './Header.css';

function Header() {

  return (

    <div className="header">

      <h1>this is header</h1>

    </div>

  );

}

export default Header;

Creating function by extension rafc

function return jsx (JavaScript XML) not html.

**JSX Guidelines-**

1. There should be one parent/wrapper element like div section main React Fragment<> </>
2. Attribute should be in camel case conventional

Like class= className

1. The tag should be closed
2. Self-closing tag like img and input should be closed.
3. To write java script inside jsx use {} brackets. This includes variable function call, and more
4. Comment in jsx.
5. function Product(props) {
6. return (
7. <div className="product">
8. {/\* <h3>{props.name}</h3>
9. <p>Rs {props. Price}</p> \*/}
10. </div>
11. );
12. }
13. export default Product;

**Babel-**

This is a library & compiler which is responsible for compiling this jsx into java script.

Like <h1></h1>

Babel compile/transpiler this jsx into js

document. createElement(‘h1’)

**difference between js/jsx**

JS is a programming language and JSX is a way of writing Java Script in terms of html/ writing js and html together.

**7. Styling Component**

1. **Separate CSS file**

**Header.css**

.header {

  border: 5px solid green;

}

No need to export

**Import CSS file**

import './Header.css'; //importing

function Header() {

  return (

    <div className="header">

      <h1>this is header</h1>

    </div>

  );

}

export default Header;

need to import

1. **inline CSS**

import "./Header.css";

function Header() {

  return (

    <div className="header">

      <h1 style= {{ backgroundColor: "yellow", color: "white" }}>// inline css

        this is header

      </h1>

    </div>

  );

}

export default Header;

pass as an object

**<h1 style=** **{{ ‘backgroundColor’: ‘yellow’, ‘color’: ‘white’ }}**

import React from 'react';

const **YourComponent** = () => {

  const styles = {

    color: 'blue',

*/\* Add more styles as needed \*/*

  };

  return <div *style*={styles}>Hello, World! </div>;

};

export default **YourComponent**;

1. **App.css**
2. we can apply class name to child component and style in App.css too
3. App.css only work for App Component and its Childs components.

Note: index.css -Global CSS

We can write global CSS for the whole application.

**Style Priority**

Inline-highest

Separate CSS file between App.css

Separate CSS file>App.css

1. **CSS Module**

To avoid conflict between Local CSS (component CSS) and global CSS.

Local CSS overwrite global CSS.

1. CSS file name should be Button.module.css
2. Import styles from ‘./Button.module.css’; - any name
3. className={styles.class}
4. className = {styles[‘class’]}
5. className ={`${styles.class} container} local CSS and global CSS

import styles from "./Button.module.css";

const **Button** = () => {

  //console.**log**(styles);

  //return <button *className*={styles.mybtn}>Click me</button>;

return <button *className*={`${styles.mybtn} ${styles.btn}`}>Click me</button>;

};

export default **Button**;

Button Component

.mybtn {

  background: red;

  color: white;

}

.btn {

  border: none;

  padding: 10px 15px;

  border-radius: 5px;

  font-weight: 600;

  letter-spacing: 1px;

  transition: 0.2s ease;

}

.btn:hover {

  background-color: white;

  border: 1px solid red;

  color: red;

  cursor: pointer;

  font-weight: 600;

}

Button Component CSS

**8. Component export and import**

****

While exporting component just export component name without (). () this means we are calling the function.

While importing the component js and jsx file just give component name without extension .jsx or js because react understand it is js or jsx file by default.

In css file we have to mention extension like .css because react does not know css

**9. Props – Properties**

Props stands for properties in react.

Props in reacting are nothing but properties of a component.

The properties of an individual component are called props.

Defined as attributes in JSX

Ex

<div class=” abs” id= ‘one>

Class and id are nothing but properties of those elements

Attributes of the element are called the property of that element.

In react we can give custom properties to an element/component.

<Product name =” Nike” price = ”2000”></Product>

Name = “nazim”- string

Age = {35}, age = ‘’2000’’ - number or string

Ismarried = {true} – Boolean

Myfun = {handleClick}

Icon ={<McCall>} – component

Note: data flow one way from parent to child component



Note - Property pass to the component is the property of the props object.

Note- to verify the props object we can use console log in js and pre-tag in jsx.

Console.log(props)

<pre>{JSON.stringify(props)} </pre>

**10. Default Props & Props Type**



**11. Handle Events in React**

Function without parameter.

function App () {

  function myfun () {

    let num = prompt ("enter a number");

    alert (num \* num);

  }

  return (

    <div className="App">

      <button onClick={myfun}>click</button>

    </div>

  );

}

If we use {myfun ()} instead of {myfun}, then the function will be called automatically.

An event handler is like a button or image where I click, dbclick, mouseover, mouseout.

Event listener is a function that gets called wherever I perform the event.

Function with parameter

function App()

{

  function myfun(name) {

    alert ("hello " + name);

  }

  return (

<div className="App">

    <button onClick= {() =>{myfun("Nazim");}}>click</button>

</div>

  );

}

export default App;

Note – first calling this function () => {} this function doing nothing but calling another

Function myfun ().

Function with event object

function App() {

  function myfun(name) {

    alert("hello " + name);

  }

  return (

    <div className="App">

      <button onClick={(event) => {console.log(event); myfun("Nazim");}}>

        click

      </button>

    </div>

  );

}

export default App;

import React from "react";

export const **test** = () => {

  const **handleClick** = (e) => {

    e.**preventDefault**();

  };

  return (

    <>

      <button *onClick*={**handleClick**}>click me</button>

    </>

  );

};

Note” - event object provides info about whole event.

**12. Conditional rendering with If else**

Return jsx with the return keyword.

Cannot write directly if else condition inside Jsx return (no if else)

import "./App.css";

function App() {

  let login = true;

  if (login) {

    return (

      <div className="App">

        <h1>true</h1>

      </div>

    );

  } else {

    return (

      <div>

        <h1>false</h1>

      </div>

    );

  }

}

export default App;

Note –

If else login base condition

Let login =0 false

Let login =1 true (any number)

Let login = false (only declared)

Let login =’ nazim” true

Let login =”” false(empty)

Let login= true Boolean

Let login = false Boolean

Null = false

Undefined =false

(empty) “” =false

NaN- not a number = false

**13. Conditional rendering with Ternary Operator**

Quick way to choose between two options

We can apply the condition inside JSX with the help of the ternary of the operator. And return element/elements.

Syntax - {Condition (expression)? Expression if true: expression if false}

{! condition? <h1>Error do it again</h1>:<h1>Congratulation</h1>}

import "./App.css";

function App () {

  //let login;//false (declared but value not assigned)

// let login=”” empty//false

  //let login = "nazim";//true

  //let login = 0; //false

  let login = 1; //true

  return (<div className="App">{login ? <h1>true</h1>: <h1>false</h1>} </div>)

}

export default App;

Example 2

const **test** = () => {

  let emptyMessage = foodsItems.length == 0? <h3>I am Still hungry</h3>: null;

  return <div>{emptyMessage}</div>;

};

export default **test**;

**Note**

null – do nothing, kuch mat kro

**14. Conditional rendering with Logical operator**

Useful for rendering content when a condition is true

Syntax

{condition && <h1>true statement</h1>} - no else block is required

import "./App.css";

function **App**() {

  let myName = 1;

  return <>{myName && <h1>true</h1>}</>;

}

export default **App**;

condition = expression

**14. Conditional Rendering with if else inside jsx**

Alternate of ternary operator and apply if else and else if.

We can apply the condition inside the jsx.

Syntax

{ ()() }

{

( () =>{if-else} ) ()

}

import "./App.css";

function App() {

  let login = 0;

  return (

    <div className="App">

      {(() => {

        if (login) {

          return (

            <div>

              <h1>true</h1>

            </div>

          );

        } else {

          return (

            <div>

              <h1>False</h1>

            </div>

          );

        }

      })()}

    </div>

  );

}

export default App;

**15. Hook**

* The function that lets you use react features.
* Some functions that let you use react features.
* Hooks are nothing but some inbuilt function that lets you use the react features or which lets you connect to the react feature.
* Hooks can be used in function base components

**16. State**

Every component can have a state.

The state is not nothing but the data of a component.

Let name = “Nazim”

it is a fixed data/normal variable of the component because it does not change the component /UI so it is not stated.

If a state changes, the component should change. This is state

import "./App.css";

function App () {

  let name = "nazim";

  function myfun () {

    name ="khan";

    alert(name);

  }

  return (

  < div className="App">

      <h1>{name}</h1>

      <button onClick={myfun}>Click me</button>

    </div>)

}

export default App;

In js variable update but not update in component/UI

**useState**

useState is a hook which is used to create state variable.

Importance of state variable:

Whenever the state variable changes component re-render.

useState hook is used for Dom manipulation.

useState hook is used for state management.

Syntax

import {useState} from ‘react’; - useState fun provide by react library

const [count, setCount] =useState (0);

count = state variable - can hold any type of data like array object, string, number Boolean, etc. like a normal variable.

setCount = state fun - hold new value and the anonymous function which returns new value

import "./App.css";

import {useState} from "react";

function App() {

 const[name, setName] = useState("nazim");

function myfun(){

    setName("Anit")

  }

  return (

  <div className="App">

      <h1>{name}</h1>

      <button onClick={myfun}>Click me</button>

    </div>)

}

export default App;

example of useState hold new value

import "./App.css";

import {useState} from "react";

function **App** () {

  let [myname, **setName**] = **useState**("Nazim");

  function **changeName** () {

**setName**((pre) => {

      console.**log**(pre);

      return `${pre} khan`;

    });

  }

  return (

    <>

      <h1>my name is {myname}</h1>

      <button *onClick*={**changeName**}>Change Name</button>

    </>

  );

}

export default **App**;

working with setter function with anonymous function that holds old value array and returning new value

<button onClick={() => setCount((count) => count + 1)}>

count is {count}

</button>

Example of useState hold anonymous function with previous value

import "./App.css";

import { useState } from "react";

function App() {

 const[name, setName] = useState("nazim");

  return (

  <div className="App">

      <h1>{name}</h1>

      <button onClick={() =>{setName("anil")}}>Click me</button>

    </div>)

}

export default App;

use directly inside an anonymous function

React understands this is a state variable if it changes, react re-renders/refreshes and reloads the same component. App/Page is not reloaded only that component re-render.

Whenever the value of the state variable changes, the component re-render but the normal variable does not do so and avoids use of the normal variable until it is a fixed variable.

**17. How to read the value input box in react**

Read value from input

import "./App.css";

function App() {

  function myfun(){

    const value = document.querySelector('input').value;

    alert(value)

  }

  return (

  < div className="App">

      <h4>Enter your name</h4>

     <input type='text'/>

      <button onClick={myfun}>click</button>

  </div>)

}

export default App;

this is the normal way in js

Read value from input box in react

import "./App.css";

import {useState} from 'react';

function App() {

  let [name, setName] = useState('')

  return (

  < div className="App">

      <h4>Enter your name</h4>

     {/\* <input type='text' onChange={(event)=>{ console.log(event.target.value) }}/>   \*/}

     <input type='text' onChange={(event)=>{ setName(event.target.value) }} value={name}/>

  </div>)

}

export default App;

example 1

Reading input value and storing in an object

import "./App.css";

import { useState } from "react";

function App() {

  let [name, setName] = useState("");

  let [age, setAge] = useState();

  let person = { userName: name, userAge: age };

  console.log(person);

  return (

    <div className="App">

      <h4>Enter your name</h4>

      <h2>{name}</h2>

      <h2>{age}</h2>

      <input type="text" onChange={(event) => {setName(event.target.value); }}placeholder="Enter Name"/>

      <input type="text"  onChange={(event) => {setAge(event.target.value);  }}placeholder="Enter age" />

    </div>

  );

}

export default App;

example2

**18. Map function in React**

import "./App.css";

function App() {

  let coountries = [

    { name: "india", capital: "New Delhi" },

    { name: "South Africa", capital: "Cape Town"},

    { name: "New Zealand", capital: "Wellington"},

  ];

  return (

    <div className="App">

      <div className="data">

        <h3>{coountries[0].name}</h3>

        <h3>{coountries[0].capital}</h3>

      </div>

      <hr />

      <div className="data">

        <h3>{coountries[1].name}</h3>

        <h3>{coountries[2].capital}</h3>

      </div>

      <hr />

      <div className="data">

        <h3>{coountries[2].name}</h3>

        <h3>{coountries[2].capital}</h3>

      </div>

    </div>

  );

}

normal way (hard code)

import "./App.css";

function App () {

  let countries = [

    { name: "India", capital: "New Delhi" },

    { name: "South Africa", capital: "Cape Town" },

    { name: "New Zealand", capital: "Wellington" },

  ];

  return (

    <div className="App">

      {

countries.map((element, index) => {

        return (

          <div key={index}>

            <h1>{element.name}</h1>

            <h2>{element.capital}</h2>

            <hr />

          </div>

        );

      })

}

    </div>

  );

}

export default App;

using map function data on the same component

import "./App.css";

import Header from "./components/Header";

import Movie from "./components/Movie";

import movies from "./movies.json";

function App () {

  return (

    <div className="App">

      <Header />

      <div className="main">

        {

movies.map((element, index) => {

          return (

            <Movie

              key={index}

              title={element.Title}

              year={element.Year}

              img={element.Poster}

            />

          );

        })

}

      </div>

    </div>

  );

}

export default App;

map function data in an external file in JSON format

[

    {

      "imdbID": "tt0096895",

      "Title": "Batman",

      "Year": "1998",

      "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTYwNjAyODIyMF5BMl5BanBnXkFtZTYwNDMwMDk2.\_V1\_.jpg"

    },

    {

      "imdbID": "tt0468569",

      "Title": "The Dark Knight",

      "Year": "2008",

      "Poster": "https://ia.media-imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODAyMTk2Mw@@.\_V1\_UX182\_CR0,0,182,268\_AL\_.jpg"

    },

    {

      "imdbID": "tt2975590",

      "Title": "Batman v Superman: Dawn of Justice",

Data in array form

**19. The life cycle of a component**

Every component created in React has something that is called a life cycle.

Life cycle 3 stages

Mounting -loading

Updating/ Re-renders – reloading only when state change

Unmounting -removal of component

When a component is loaded in the webpage first time that is called a mounting component.

When a component is removed from the webpage that is called unmounting component

There are 3 things that component does

1. mounting (when first-time load).
2. whenever the state change component refreshes itself that is called reloading/updating/re-rendering component.
3. Component unload/ unmounted

**20. useEffect function**

**Hooks**

hooks are functions that let you connect to react features

.

useEffect connects to the life cycle method.

Because it is called when a component is mounted, remounted, or unmounted

we don’t call useEffect function, we only use it.

It is internally made to be called when a component is mounted /reloaded/unmounted

We write code inside the useEffect function

When we perform a task whenever the component mounts/load , reload and the component unmounts

useEffect is a side effect function

useEffect is a function that is by default called when the component mounted, reloaded/rendered and unmounted.

**useEffect Syntax**

****



Note: - why app running twice?

React is running in strict mode.

The first time it runs only for checking the app

The Second time it runs as final all

import "./App.css";

import Product from './components/Product'

import {useEffect} from 'react';

function App() {

  useEffect(()=>{

    console.log("component rendered");

  })

  return (

    <div className="App">

          <Product name = "samsung" price="20000"/>

    </div>

  );

}

export default App;

useEffect is called when the component mount for first time

import "./App.css";

import Product from './components/Product'

import {useEffect, useState} from 'react';

function App() {

  const[name, setName]=useState("nazim")

  useEffect(()=>{

    console.log("component rendered");

  })

  return (

    <div className="App">

          <Product name = "samsung" price="20000"/>

          <h2>Welcome {name}</h2>

          <button onClick={()=>{setName("anit")}}> click me </button>

    </div>

  );

}

export default App;

useEffect gets called again when the component re-render due to state variable change

We know whenever the state variable changes

that particular component re-render.

**Example of useEffect**

import "./App.css";

import Product from './components/Product'

import {useEffect, useState} from 'react';

function App() {

  const[name, setName]=useState("nazim")

  useEffect(()=>{

    fetch('https://jsonplaceholder.typicode.com/todos/1')

      .then(response => response.json())

      .then(json => console.log(json))

  })

  return (

    <div className="App">

          <Product name = "samsung" price="20000"/>

          <h2>Welcome {name}</h2>

          <button onClick={()=>{setName("anil")}}> click me </button>

    </div>

  );

}

export default App;

now data inside useEffect is coming on component load due to useEffect fun and also on reload because state variable change cause component re-render.

import "./App.css";

import Product from './components/Product'

import {useEffect, useState} from 'react';

function App() {

  const[name, setName]=useState("nazim")

  useEffect(()=>{

    fetch('https://jsonplaceholder.typicode.com/todos/1')

      .then(response => response.json())

      .then(json => console.log(json))

  },[])

  return (

    <div className="App">

          <Product name = "samsung" price="20000"/>

          <h2>Welcome {name}</h2>

          <button onClick={()=>{setName("anil")}}> click me </button>

    </div>

  );

}

export default App;

due to blank dependency array [] useEffect runs only when component load and not run when component re-render due to state variable change.

import "./App.css";

import Product from './components/Product'

import {useEffect, useState} from 'react';

function App() {

  const[name, setName]=useState("nazim")

  const[age, setAge] = useState(35);

  useEffect(()=>{

    fetch('https://jsonplaceholder.typicode.com/todos/1')

      .then(response => response.json())

      .then(json => console.log(json))

  },[age])

  return (

    <div className="App">

          <Product name = "samsung" price="20000"/>

          <h2>Welcome {name}</h2>

           <h2>{age}</h2>

          <button onClick={()=>{setName("anil")}}> click me </button>

          <button onClick={()=>{setAge(36)}}> click me </button>

    </div>

  );

}

export default App;

useEffect fun run for the first time when the component load and the second time when the state variable inside the dependency array [] will change/ when the component re-renders because of age state variable.

Note –

1. By default, useEffect fun is called for every render and re-render.
2. You can stop it from getting called on re-render by applying a blank dependency array [].
3. You can also pass a variable (props, state) in a dependency array for which particular re-render you want to call useEffec fun.

**21. Load component on button**

To see the mounting and unmounting

import "./App.css";

import Product from "./components/Product.js";

function App() {

  return (

    <div className="App">

      <h2>Welcome</h2>

      <Product name="samsung" price="2000" />

    </div>

  );

}

export default App;

by default, component load this way

Conditional rendering for mounting and unmounting the component

import "./App.css";

import Product from "./components/Product.js";

import { useState } from "react";

function App() {

  const [productVisible, setProductVisible] = useState(false);

  return (

    <div className="App">

      {<h2>Welcome</h2>}

      {productVisible == true ? <Product name="samsung" price="20000" /> : null}

      <button onClick={() => setProductVisible(true)}>Show product</button>

      <button onClick={() => setProductVisible(false)}>Hide product</button>

    </div>

  );

}

export default App;

On the component mounting useEffect fun gets a call that is inside Product comp

import "./Product.css";

import { useEffect } from "react";

function Product(props) {

  useEffect(() => {

    fetch("https://jsonplaceholder.typicode.com/todos/1")

      .then((response) => response.json())

      .then((json) => console.log(json));

  });

  return (

    <div className="product">

      <h3>{props.name}</h3>

      <p>{props.price}</p>

    </div>

  );

}

export default Product;

Component is mounting and unmounting by clicking on button in app.js. (Product Comp)

Note: - useEffect get call when the component mount but not get call when unmount the component by clicking the button until we add

return function() { console.log(‘dummy data’)} inside useEffect fun

import "./Product.css";

import { useEffect } from "react";

function Product(props) {

  useEffect(() => {

    fetch("https://jsonplaceholder.typicode.com/todos/1")

      .then((response) => response.json())

      .then((json) => console.log(json));

      return function () {

      fetch("https://jsonplaceholder.typicode.com/todos/1")

        .then((response) => response.json())

        .then((json) => console.log(json));

    };

  });

  return (

    <div className="product">

      <h3>{props.name}</h3>

      <p>{props.price}</p>

    </div>

  );

}

export default Product;

Now use effect getting call on component mount and as well as component unmount

**22. Getting data from API**

import "./App.css";

import { useEffect, useState } from "react";

function App() {

  const [users, setUser] = useState([]);

  useEffect(() => {

    fetch(`https://jsonplaceholder.typicode.com/users`)

      .then((res) =>{ if(!res.ok){

        throw Error(res.statusTex)};

        return res.json()})

      .then((data) => {

        console.log(data);

        setUser(data);

      }).catch(error=>alert(error));

  }, []);

  return (

    <div className="App">

      <h1>hello world</h1>

      {/\* <pre>{JSON.stringify(users)}</pre> \*/}

      {users.map((elm, index) => {

        return (

          <div key={index}>

            <h3>Name - {elm.name}</h3>

            <h3>email - {elm.email}</h3>

          </div>

        );

      })}

    </div>

  );

}

export default App;

fetching data from API using promise then

import "./App.css";

import { useEffect, useState  } from "react";

function App() {

  const [users, setUser] = useState([]);

  useEffect(() => {

    async function fetchData () {

      try{

        const res = await fetch(`https://jsonplaceholder.typicode.com/users`);

        if(!res.ok){

          throw Error(res.statusText)

        }

        const data = await res.json();

        console.log(data);

        console.log(typeof data)//object

        setUser(data);

      }

      catch(error){

       alert(error)

      }

    }

   fetchData();

  }, []);

  return (

    <div className="App">

      <h1>hello world</h1>

      <pre>{JSON.stringify(users)}</pre>

      {

        users.map((elm, key)=>{

          return (

            <div key={key}>

                <h1>{elm.name}</h1>

                <h2>{elm.email}</h2>

            </div>

          )

        })

      }

    </div>

  );

}

export default App;

fetching data from API using async function

**23.Project**

**Key points**

* useState
* uesEffect.
* Fetch fun with error handling
* Taking out an array of data from a data object that is received from API.
* Responsive Inbox for news search and getting inbox value with event object of onChange event.
* Use of if else inside the inbox for blank value.
* Error handling if data is not found from API.
* Remove blank news card array from array data with filter fun.
* Loop component with map fun.
* Pass props to component.
* Receive props on the news component
* Handling props that image src is null with ternary operator
* To shorten the news description, use substring and concat function with chaining operator

**24. useContext and Context API in React**

Manage state globally

UseContext hook is kind of replacement of redux and solution to props drilling.

How o makes the data available globally, there are two ways.

1. Context provided by react
2. Third party tool, library called as redux.

In react context API helps you create the container(context)-like an area

Context means container or area.

contextAPI helps you create a container in which you can share the data and make the data available globally.

We create container around our main component which is app component.

Whatever I created data in any component, I tell the contest this is global data I want

**APP.js**

import { useState, createContext } from "react";

import Product from "./components/Product";

import "./App.css";

export const UserContext = **createContext** (); *// create a context*

function **App** () {

  const [username, **setUsername**] = **useState**("nazim");

  const person = {name: "nazim", age: 35, city: "Dehradun”};

  return (

    <>

      {*/\* // wrap everything in context and give value \*/*}

      <**UserContext.Provider** *value*={person}>

        <**Product** />

      </**UserContext.Provider**>

    </>

  );

}

export default **App**;

creating context and providing/sharing data to context(container)

Note – value has only one (data) parameter that can be passed for multiple data use object or array. We can also pass function

**Product.js**

import Category from “. /Category";

import { UserContext } from “../App";//import Context

import { useContext } from "react";//import usecontext to get data form context

const **Product** = () => {

  const styles = {border: "3px solid red”, padding: "20px”, fontSize: "30px",};

  const u = **useContext**(UserContext);//get the value from context created

  console.**log**(u);

  return (

    <>

      <div *style*={styles}>

        Product {u.name}

        <**Category** />

      </div>

    </>

  );

};

export default **Product**;

category component is the child component of the Product component

**Category.js**

import {UserContext} from "../App"; *//import context*

import {useContext} from "react"; *//useContect to get data from context*

const **Category** = () => {

  let u = **useContext** (UserContext); *//get the value from context created*

  console.**log**(u);

  const styles = {

    border: "2px solid green",

  };

  return <div *style*={styles}>Category {u.age}</div>;

};

export default **Category**;

example for object values

* a=2
* b=24
* let obj = {a, b} //if I have values in variables
* obj
* {a: 2, b: 24}

**Class base Component**

import React from "react";

class **Demo** extends **React**. **Component** {

**render** () {

    return (

      <div>

        <h1>This is class component</h1>

      </div>

    );

  }

}

export default **Demo**;

importing is same as functional component

**26. Routing**

Whenever we open or access something based on a path/route, that is called routing. Whatever we are opening/accessing, it can be a file, it can be a webpage, it can be a function and it can also be a component.

To achieve the routing, we have to installed one library that is react router dom.

* Npm install react-router-dom

React router dom package/library gives 3 main components

1. <BrowserRouter> </BrowserRouter> - read path from browser
2. <Routers></ Routers>
3. <Router- has two parameter path and element/>

import "./App.css";

import About from “. /components/About";

import Product from “. /components/Product";

import Service from “. /components/Service";

import {BrowserRouter, Routes, Route} from "react-router-dom";

function **App** () {

  return (

    <>

      {*/\* read browser path and decide which comp show based on path \*/*}

      <**BrowserRouter**>

        <**Routes**>

          <**Route** *path*="/" *element*={<**About** />} />

          <**Route** *path*="/product" *element*={<**Product** />} />

          <**Route** *path*="/service" *element*= {<**Service** />} />

        </**Routes**>

      </**BrowserRouter**>

    </>

  );

}

export default **App**;

basic routing by manually

**Links**

Link should be under BrowserRouter and don’t use an anchor tag use link component.

import styles from "./App.module.css";

import About from "./components/About";

import Product from "./components/Product";

import Service from "./components/Service";

import {BrowserRouter, Routes, Route, Link} from "react-router-dom";

function **App**() {

  return (

    <>

      {*/\* read browser path \*/*}

      <**BrowserRouter**>

        <nav>

          <ul *className*={styles.navigation}>

            <li>

              <**Link** *className*={styles.link} *to*="/">

                Home

              </**Link**>

            </li>

            <li>

              <**Link** *className*={styles.link} *to*="/product">

                Product

              </**Link**>

            </li>

            <li>

              <**Link** *className*={styles.link} *to*="/service">

                Service

              </**Link**>

            </li>

          </ul>

        </nav>

        <**Routes**>

          <**Route** *path*="/" *element*={<**About** />} />

          <**Route** *path*="/product" *element*={<**Product** />} />

          <**Route** *path*="/service" *element*={<**Service** />} />

        </**Routes**>

      </**BrowserRouter**>

    </>

  );

}

export default **App**;

routing by navigation links

**27. useRef hook**

useRef is used to create a variable whose value persist after rerender and does not cause re-render

State variable value persist after rerender but normal variable that value getting change does not persist after rerender.

import "./App.css";

import Refdemo from "./Refdemo";

import { useState, useEffect } from "react";

function **App**() {

  const [name, **setName**] = **useState**("Nazim");

  let nVar = 0;//value reset when comp rerender

**useEffect** (() => {

    console.**log**(name);

    console.**log**(nVar);

  });

  return (

    <>

      <h1>useRef React</h1>

      <**Refdemo** />

      <h1>my name {name}</h1>

      <button *onClick*={() =>{**setName**("khan"); nVar=1;}// change nVar value

    }>click me</button>

    </>

  );

}

export default **App**;

nVar variable value not persist after re-render this is the problem

persist means continue to exist- bane rahatee hai

**Note:**

State variable – value persist but causes rerender

Normal variable – does not rerender but value does not persist

useRef – value persist and it will not cause rerender. Combination of state variable and normal variable

**Objective**

we need a variable whose value persist thought the renders

we need a variable whose value persist and it will not cause rerednder

for this we use useRef hook

import "./App.css";

import Refdemo from "./Refdemo";

import {useState, useEffect, useRef} from "react";

function **App** () {

  const [name, **setName**] = **useState**("Nazim");

  let nVar = 0;

  let rVar = **useRef**(10);

**useEffect** (() => {

    console.**log**(name);

    console.**log**(nVar);

    console.**log** (rVar.current);

  });

  return (

    <>

      <h1>useRef React</h1>

      <**Refdemo** />

      <h1>my name {name}</h1>

<h1>my name {nVar}</h1>

<h1>my name {rVar.current}</h1>

      <button

*onClick*= {() => {

**setName**("khan");

          nVar = 1;

          rVar.current = 11;

        }}

      >

        click me

      </button>

    </>

  );

}

export default **App**;

Note: - let rVar= useRef (10)

Variable rVar does not hold the value 10 directly.

It holds the value like this

rVar={current:10}

Now rVar variable value persist after rerender that is not persist with normal variable

**Note difference between state variable and useRef variable**

When state variable value changes, component re-render and when useRef variable value change, component not rerender. Only render is difference

**useRef for dom manipulation**

useRef is also used for dom manipulation.

useRef is the only variable which can hold html element in react, normal variable cant holds the value like in js.

99% we don’t need to dom manipulation

import "./App.css";

import {useRef} from "react";

function **App** () {

  let pTag = **useRef** (); *//blank useRef*

  console.**log**(pTag);

  return (

    <>

      <p *ref*={pTag}>hello world</p>

      <button

*onClick*= {() => {

pTag.current.innerHTML = "red";

          pTag.current.style.color = "red";

        }}

      >

        change content

      </button>

    </>

  );

}

export default **App**;

**Two uses of useRef**

* It is used to store data which can persist thought re-render.
* It is used to store HTML element which you want to manipulate.

**28. Form and Complex State Handling in React**

* How to update state when it contains array or object

import "./App.css";

import {useState} from "react";

function **App** () {

    let [animals, **setAnimals**] = **useState** (["Lion", "Cheetah", "Hyena"]);

  function **addAnimal** () {

    let copyanimals = [...animals];

    copyanimals.**push**("deer");

    console.**log**(copyanimals); // ['Lion', 'Cheetah', 'Hyena', 'deer', 'deer']

**setAnimals**(copyanimals);

  }

  return (

    <>

      {animals.**map**((animal, key) => {

        return <h3 *key*={key}>{animal}</h3>;

      })}

      <button *onClick*={**addAnimal**}>add new animal</button>

    </>

  );

}

export default **App**;

The setter function holds a new value or an anonymous function which take old value as a parameter and returns the new value.

import "./App.css";

import {useState} from "react";

function **App** () {

  let [myname, **setName**] = **useState**("Nazim");

  function **changeName** () {

**setName**((pre) => {

      console.**log**(pre);

      return `${pre} khan`;

    });

  }

  return (

    <>

      <h1>my name is {myname}</h1>

      <button *onClick*={**changeName**}>Change Name</button>

    </>

  );

}

export default **App**;

working with setter function with anonymous function that holds old value

import "./App.css";

import {useState} from "react";

function **App** () {

  let [animals, **setAnimals**] = **useState** (["Lion", "Cheetah", "Hyena"]);

  function **addAnimal** () {

**setAnimals**((pre) => {

      console.**log**(pre);

      return [...pre, "Deer"];

    });

  }

  return (

    <>

      {animals.**map**((animal, key) => {

        return <h3 *key*={key}>{animal}</h3>;

      })}

      <button *onClick*={**addAnimal**}>add new animal</button>

    </>

  );

}

export default **App**;

working with setter function with anonymous function that holds old value array and returning new value

Note: - two uses of setter function

1. Directly setting a new value.
2. Getting old value as a parameter and do some operation with it and return new value

**Form Handling – Part-1**

import {useState} from 'react'

import './App.css'

function **App** () {

  const [name, **setName**] = **useState** ('');

  const [email, **setEmail**] = **useState** ('')

  function **handleName**(event) {

**setName** (() =>event.target.value)

  }

  function **handleEmail**(event) {

**setEmail** (() =>event. target.value)

  }

  let obj= {name, email};

  console.**log**(obj);

  return (

    <>

    <h4>{name}</h4>

    <h4>{email}</h4>

      <input *type* ="text" *onChange*={**handleName**} *value*={name}*placeholder*="enter name"/>

      <input *type* ="email" *onChange*={**handleEmail**} *value*={email}*placeholder*="enter email"/>

    </>

  )

}

Problem n no of fields n no of useState and no no of functions

**Form Handling – Part-2**

import "./App.css";

import ComplexState from “. /ComplexState";

import { FormHandling } from “. /FormHandling";

import {useState} from "react";

function **App** () {

   let [details, **setDetails**] = **useState** ({

    name: "",

    email: "",

    phone: "",

  });

  function **handleInput**(event) {

**setDetails**((preObj) => {

      return { ...preObj, [event.target.name]: event.target.value};

    });

  }

  function **handleSubmit**() {

    console.**log**(details);

  }

  return (

    <>

      <input

*type*="text"

*name*="name"

*onChange*={**handleInput**}

*placeholder*="enter name"

*value*={details.name}

      />

      <input

*type*="email"

*name*="email"

*onChange*={**handleInput**}

*placeholder*="enter email"

*value*={details.email}

      />

      <input

*type*="phone"

*name*="phone"

*onChange*={**handleInput**}

*placeholder*="enter phone"

*value*={details.phone}

      />

      <button *onClick*={**handleSubmit**}>Submit</button>

    </>

  );

}

export default **App**;

from handling with one state variable and common event listener

let n = ‘hello’;

let obj = {n: ‘nazim’}

let obj= {[n]: ‘nazim’}

obj {hello: ‘nazim’}

**Random**

React is all about components, props, and hooks that are all

If that data retenders the component, then it is a state variable and no a fixed variable.

<button onClick= {function () {console.log(‘hello’} }>click me</button>

<button onClick={ ()=>{console.log(‘hello’} }>click me</button>

Npx create-react-app secondapp -> extension js

Npx creae-vite secondapp –template react -> extension jsx

Npm install

**useState and useEffect**

useState is used to create a state variable

useEffect is a function that gets called when the component loads/mounts, reloads/rerenders, and unmounts

whenever the state variable changes the component re-renders/reloads

whenever the component reloads usEffect function gets called

Map function is used to print fetched bulk data (in form of array or object) on webpage in react.

ForEach is used to print fetch bulk data (in form of array or object) on webpage in JS.

**React Fragment <> </>**

import React from “react”;

export Function App () {

Return (

<React.Fragment>

</React.Fragment>

)

}

Note - return multiple elements without a wrapping parent.

No extra div shortcut <> </>

**Inside function return() we can write**

1. JSX
2. Java Script as {} expression and map fun/loop
3. Component

**Toggle component using state**

import React, {useState} from 'react';

const **ToggleComponent** = () => {

  const [isVisible, **setIsVisible**] = **useState**(true);

  const **toggleVisibility** = () => {

**setIsVisible**(!isVisible);

  };

  return (

    <div>

      <button *onClick*={**toggleVisibility**}>Toggle Component</button>

      {isVisible && <**YourComponentToShow** />}

    </div>

  );

};

const **YourComponentToShow** = () => {

  return <p>Your component content here.</p>;

};

export default **ToggleComponent**;