* React hooks Rules :
  + React hooks always start with use…

1. React Hooks are only used in function based components.
2. Only call hooks in top level
   1. Don’t call hook in loops .
   2. Don’t call hook in any block content.

* React fragment :

import React from ‘react’;

const compo = (props) =>{

return (

<React.Fragment>

</React.Fragment>

);

};

Export default Compo;

\*React fragment is use as wrapper component.

* useRef() hook:

const refContainer = useRef(initialValue);

useRef returns a mutable ref object whose .current property is initialized to the passed argument (initialValue).

Note : to use useRef on component :

import React, { useRef } from "react";

const MealItemForm = (props) => {

  const amountInputRef = useRef();

  return (

    <form className={styles.form} onSubmit={submitHandler}>

      <Input

        ref={amountInputRef}       //here we use ref

      />

    </form>

  );

};

export default MealItemForm;

Changes to Component:

import React from "react";

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

const Input = React.forwardRef((props, ref) => {

  return (

    <div className={styles.input}>

      <label htmlFor={props.input.id}>{props.label}</label>

      <input ref={ref} {...props.input} />

    </div>

  );

});

export default Input;

* React portals :

ReactDOM.createPortal(child, container)

Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

* Side Effect ( useEffect hook):

What is Effect (side Effect)?

1. Side effect : anything else
2. Store data in browser storage, send http request to backend server, set and manage timer.
3. These task must happen outside of normal component evaluation.

useEffect(() =>{ … }, [dependencies] );

A function that should dependancies of these

Be executed after effect.

Every component evalution the function only runs

If the specified dependencies if dependencies

Changes changed.

1. useEffect is execute every time when we render components.
2. If we provide dependecies to useEffect then after dependencies are changed then only useEffect execute.
3. useEffect( () =>{ }, [ ] )

If we provide empty dependencies then only once after mount useEffect fucntion executed.

1. useEffect ( () =>{

//code

return () =>{

//clean up code

}

}, [] }

So whenever useEffect run first clean up code get executed for clean up purpose after that code will get executed.

* useReducer() hook:

sometime we have more complex states – for example if we got multile state, multiple way of changing it or dependecies to other states.

useState() then become hard or error prone to use. It will produce bad , ineffiecient or buggy code in such scenarios.

useReducer() can be used as a replacement for useState( ) if we need more powerfull state management

const [state , despatchFunction ] = useReducer(reducerFunc, initialState, initFunc)

state a function that can be

snapshot use to dispatch a new initial state

action(trigger update initial function

of the state)

1. reducerFunc :

(prevState, action) => newState

A function is triggred automatically once action is dispatch via.

depatchFunction() it receives the last state snapshot and

should return new and updated state

(write this function outside component function)

import React, { useState, useReducer} from 'react';

const emailReducer = (state, action) =>{

  if(action.type === 'USER\_INPUT'){

    return {value : action.val, isValid : action.val.includes('@') }

  }

  if(action.type === 'INPUT\_BLUR'){

    return  {value : state.value, isValid : state.val.includes('@')};

  }

  return {value : '', isValid : false};

};

const Login = (props) => {

  const [emailState, dispatchEmail]=useReducer(emailReducer,{value : '', isValid : null })

    dispatchEmail({type : 'USER\_INPUT', val : event.target.value})

    dispatchEmail({type : 'INPUT\_BLUR'});

  const submitHandler = (event) => {

    event.preventDefault();

    props.onLogin(emailState.value, enteredPassword);

  };

};

export default Login;