**ReactJS**  
Js Array method to revise:

Map, fiter,Reduce,FindIndex,Concat,Slice,Splice

React Js –

-----------RectJS is a javascript library for building user interface

------------React is all about component

Component: A component in react is just a JavaScript function (which returns a template)

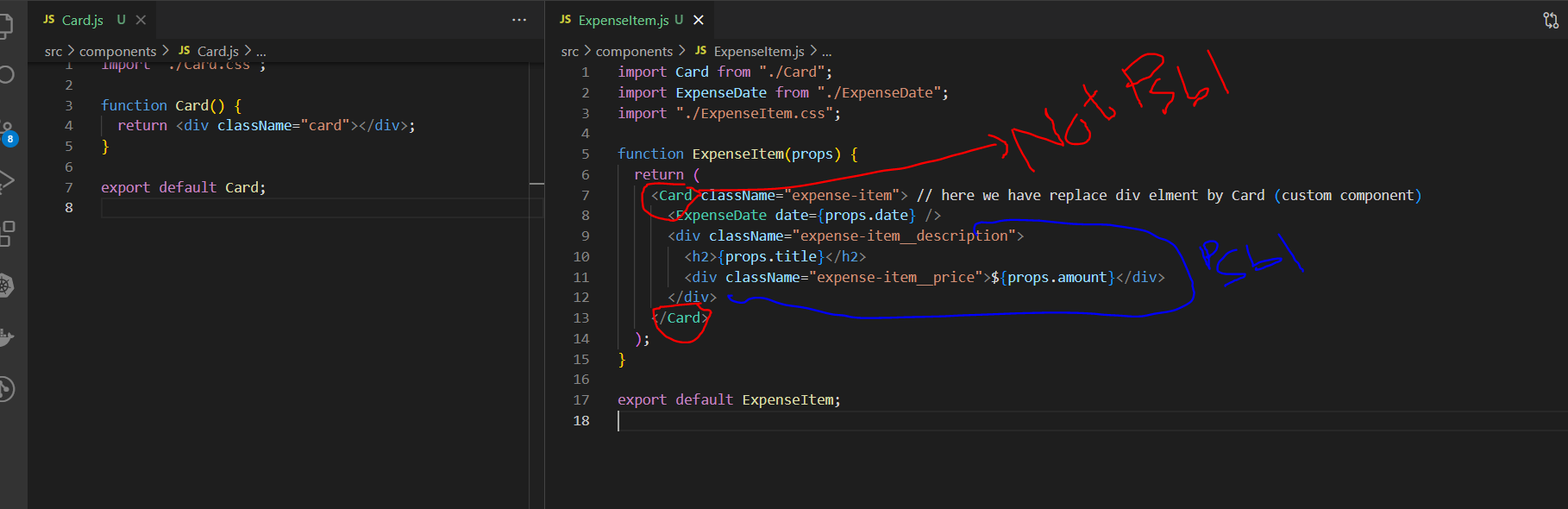
Why components

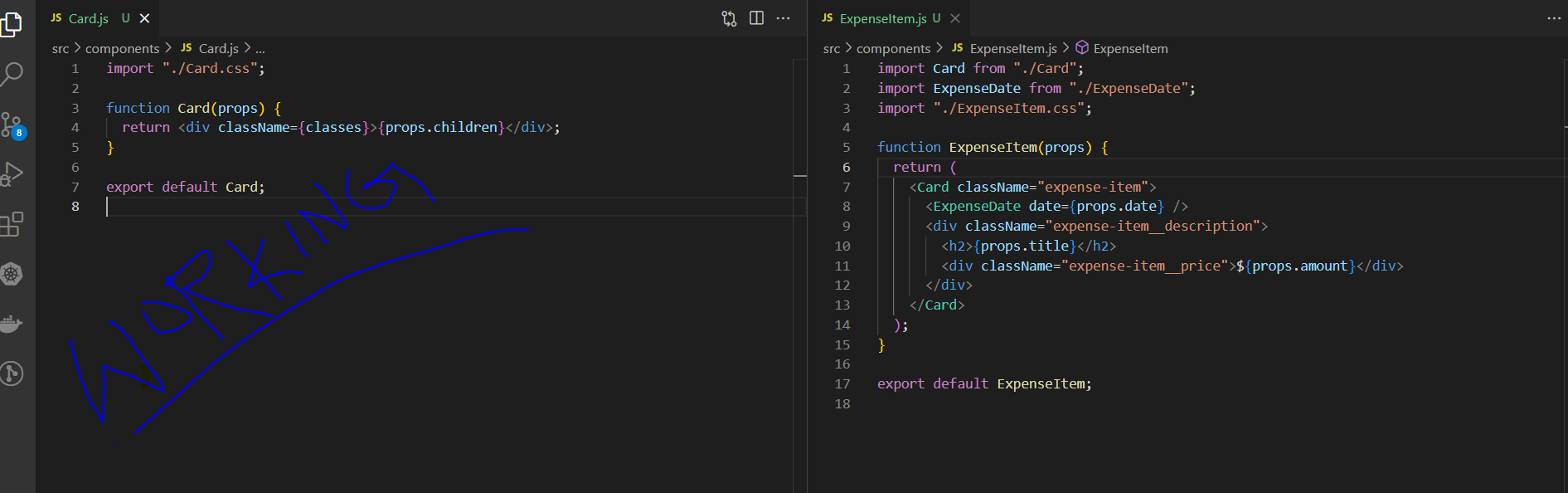
* Reusability
* Separation of concerns
* Split big chunk of code into small chunks

Note:

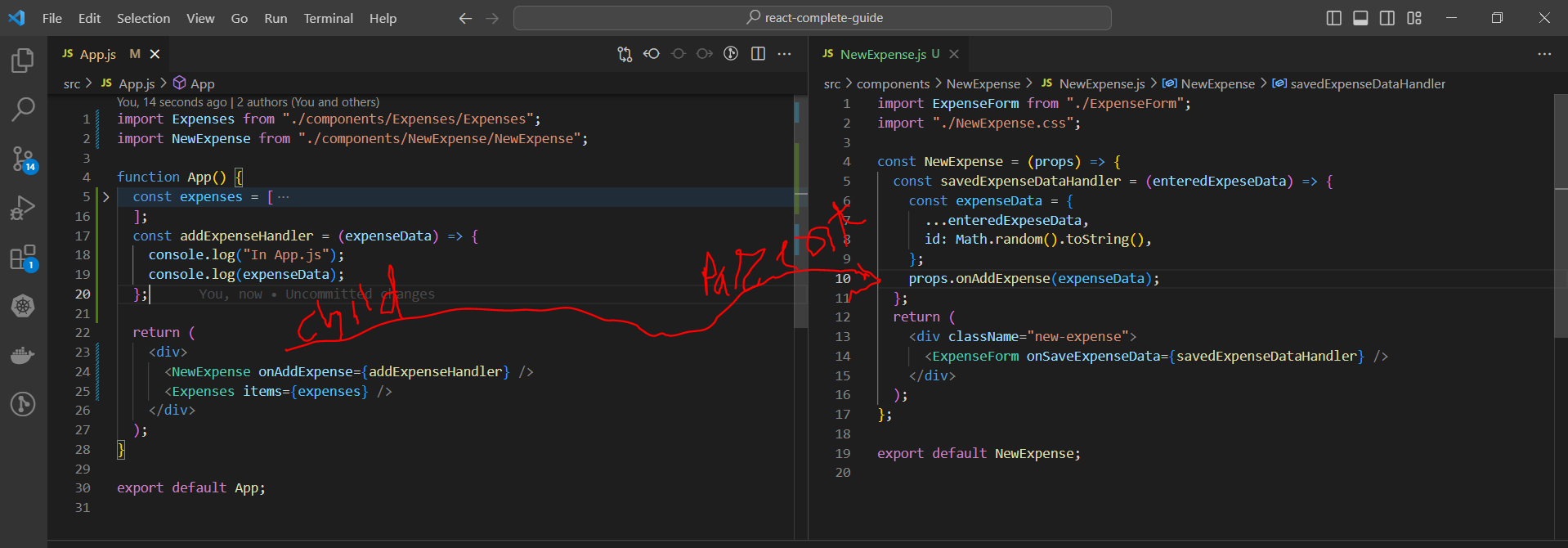
1. Html element defined in lowercase eg: div, tag
2. Component name always starts with uppercase letter coz it defined by developer

**Note: In react you must only have one root element for return statements**





**Child-to-Parent: Data binding**



**Dynamic add CSS class:**

      <div className={`form-control ${!isValid} ? 'invalid' : ''`}>

**Dynamic Add style (inline method)**

<label style={{ color: !isValid ? "red" : "black" }}>Course Goal</label>

        <input

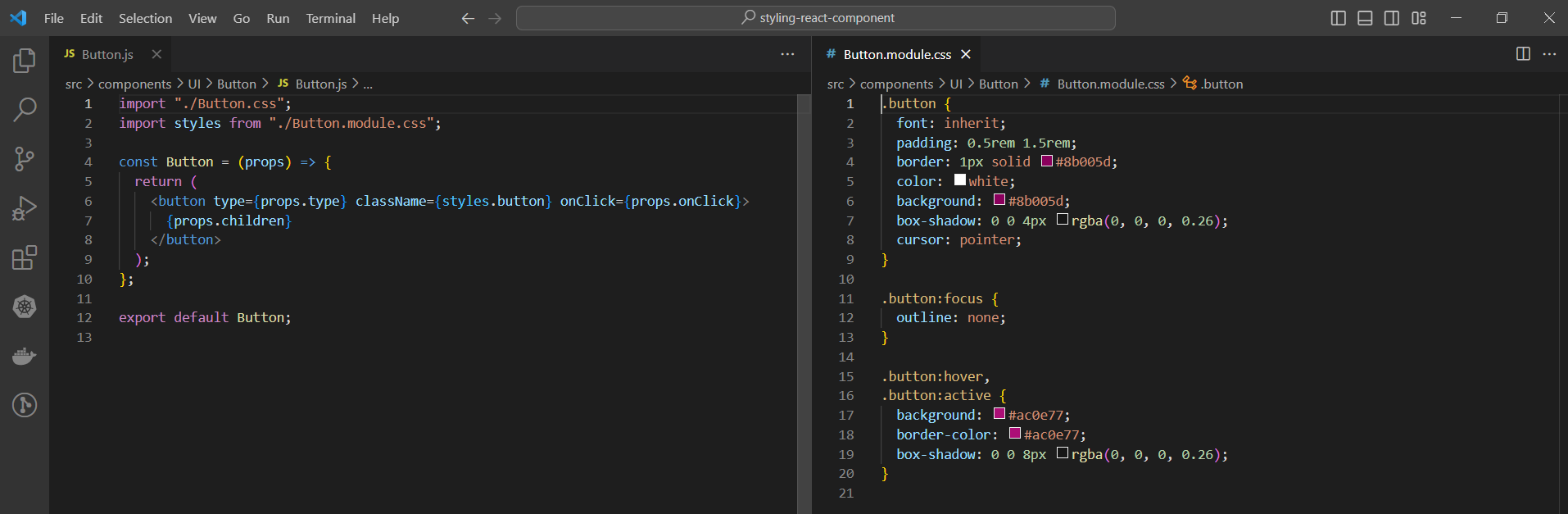
          style={{

            borderColor: !isValid ? "red" : "#ccc",

            background: !isValid ? "salmon" : "transparent",

          }}

**Using CSS Module**



**JSX Limitation**

 return(  
 <h2>Hi There</h2>

  <p>This does not work</p>  
)

**You can’t return more than one “root” JSX element (you also don’t store more than one**

**“root” JSX element in a variable)**

**Solution:**You can solve the above problem by wrapping around element using div

return(

<div>  
 <h2>Hi There</h2>

  <p>This does not work</p>

</div>  
)

**Note :** Using div creates “<div> soup “ problem .

**Solution** : You can use react fragment to overcome the div soup issue

return (

    <React.Fragment>

      <AddUser onAddUser={addUserHandler} />

      <UsersList users={usersList} />

    </React.Fragment>

  );

Or

import { useState, Fragment } from "react";

return (

    <Fragment>

      <AddUser onAddUser={addUserHandler} />

      <UsersList users={usersList} />

    </Fragment>

  );

Or

return (

    <>

      <AddUser onAddUser={addUserHandler} />

      <UsersList users={usersList} />

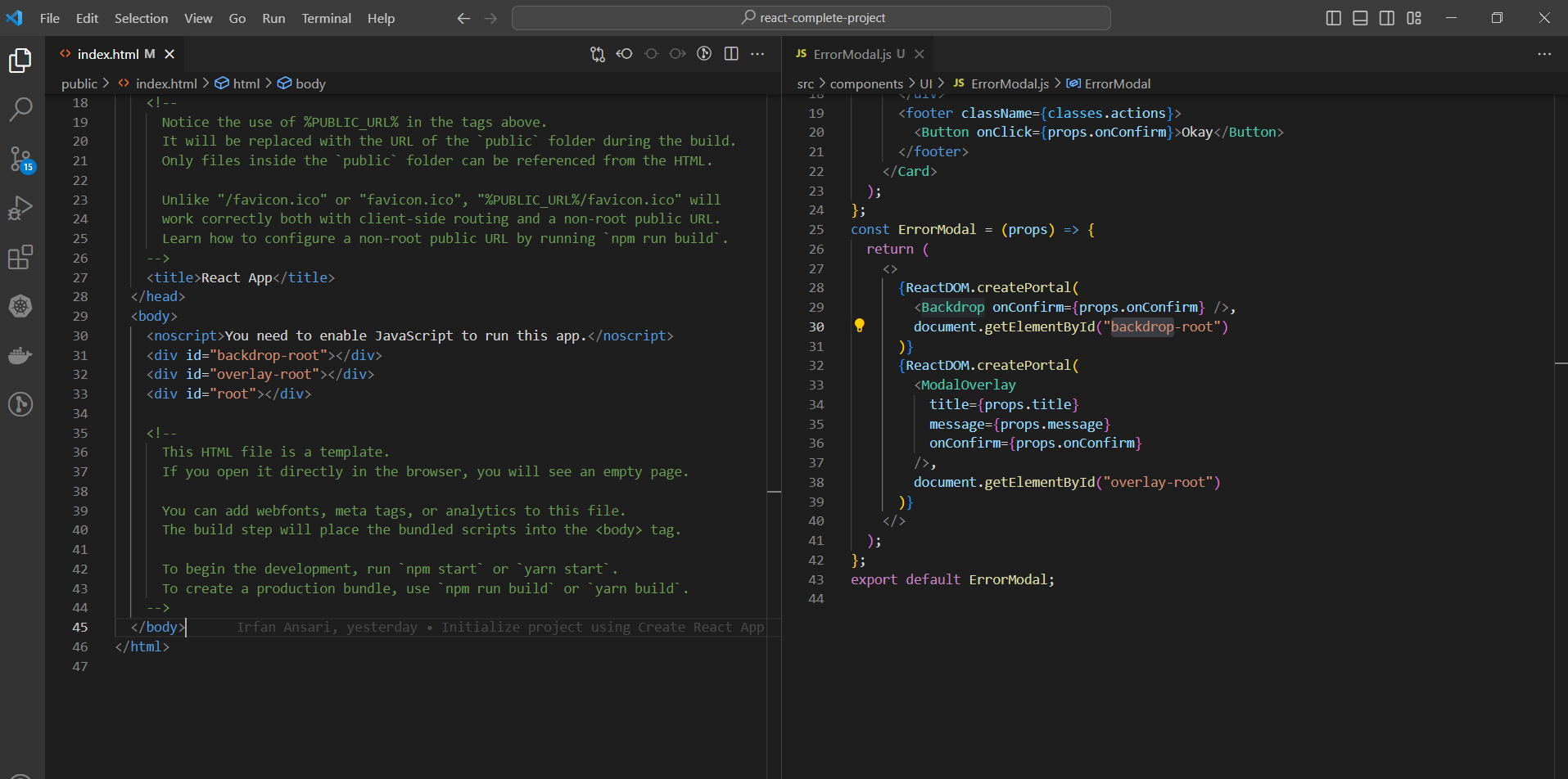
    </>

  );

**React Portal:**

A method to render a component in a particular div

Eg:



**Working with Ref’s**

Using the useRef you can get the current value of an element.

**Eg**: Commented code are before using the ref

import { useRef, useState } from "react";

import Wrapper from "../Helpers/Wrapper";

import Button from "../UI/Button";

import Card from "../UI/Card";

import ErrorModal from "../UI/ErrorModal";

import classes from "./AddUser.module.css";

const AddUser = (props) => {

  const nameInputRef = useRef();

  const ageInputRef = useRef();

  // const [enteredUsername, setEnteredUsername] = useState("");

  // const [enteredAge, setEnteredAge] = useState("");

  const [error, setError] = useState();

  // const userNameChangeHandler = (event) => {

  //   setEnteredUsername(event.target.value);

  // };

  // const ageChangeHandler = (event) => {

  //   setEnteredAge(event.target.value);

  // };

  const addUserHandler = (event) => {

    event.preventDefault();

    const enteredName = nameInputRef.current.value;

    const enteredUserAge = ageInputRef.current.value;

    if (enteredName.trim().length === 0 || enteredUserAge.trim().length === 0) {

      setError({

        title: "Invalid Input",

        message: "Please enter a valid name and age (non-empty values)",

      });

      return;

    }

    if (+enteredUserAge < 1) {

      // added plus('+') icon to make number (default is string)

      setError({

        title: "Invalid age",

        message: "Please enter a valid age (> 0)",

      });

      return;

    }

    // props.onAddUser(enteredUsername, enteredAge);

    // setEnteredAge("");

    // setEnteredUsername("");

    props.onAddUser(enteredName, enteredUserAge);

    nameInputRef.current.value = "";

    ageInputRef.current.value = "";

  };

  const errorHandler = () => {

    setError(null);

  };

  return (

    <Wrapper>

      {error && (

        <ErrorModal

          title={error.title}

          message={error.message}

          onConfirm={errorHandler}

        />

      )}

      <Card className={classes.input}>

        <form onSubmit={addUserHandler}>

          <label htmlFor="username">Username</label>

          <input

            id="username"

            type="text"

            // value={enteredUsername}

            // onChange={userNameChangeHandler}

            ref={nameInputRef}

          />

          <label htmlFor="age">Age(Years)</label>

          <input

            id="age"

            type="number"

            // value={enteredAge}

            // onChange={ageChangeHandler}

            ref={ageInputRef}

          />

          <Button type="submit">Add User</Button>

        </form>

      </Card>

    </Wrapper>

  );

};

export default AddUser;

**controlled vs uncontrolled components**

controlled – When we manage state in our component then that is called controlled component.

Un-controlled – when we don’t mange state that is called un-controlled compoentnet.

**useEffect**

useEffect is use to handle side effect .

**What to add & Not to add as Dependencies**

In the previous lecture, we explored useEffect() dependencies.

You learned, that you should add "everything" you use in the effect function as a dependency - i.e. all state variables and functions you use in there.

That is correct, but there are a **few exceptions** you should be aware of:

* You **DON'T need to add state updating functions** (as we did in the last lecture with setFormIsValid): React guarantees that those functions never change, hence you don't need to add them as dependencies (you could though)
* You also **DON'T need to add "built-in" APIs or functions** like fetch(), localStorage etc (functions and features built-into the browser and hence available globally): These browser APIs / global functions are not related to the React component render cycle and they also never change
* You also **DON'T need to add variables or functions** you might've **defined OUTSIDE of your components** (e.g. if you create a new helper function in a separate file): Such functions or variables also are not created inside of a component function and hence changing them won't affect your components (components won't be re-evaluated if such variables or functions change and vice-versa)

So long story short: You must add all "things" you use in your effect function **if those "things" could change because your component (or some parent component) re-rendered.** That's why variables or state defined in component functions, props or functions defined in component functions have to be added as dependencies!

Here's a made-up dummy example to further clarify the above-mentioned scenarios:

1. import { useEffect, useState } from 'react';
3. let myTimer;
5. const MyComponent = (props) => {
6. const [timerIsActive, setTimerIsActive] = useState(false);
8. const { timerDuration } = props; // using destructuring to pull out specific props values
10. useEffect(() => {
11. if (!timerIsActive) {
12. setTimerIsActive(true);
13. myTimer = setTimeout(() => {
14. setTimerIsActive(false);
15. }, timerDuration);
16. }
17. }, [timerIsActive, timerDuration]);
18. };

In this example:

* timerIsActive is **added as a dependency** because it's component state that may change when the component changes (e.g. because the state was updated)
* timerDuration is **added as a dependency** because it's a prop value of that component - so it may change if a parent component changes that value (causing this MyComponent component to re-render as well)
* setTimerIsActive is **NOT added as a dependency** because it's that **exception**: State updating functions could be added but don't have to be added since React guarantees that the functions themselves never change
* myTimer is **NOT added as a dependency** because it's **not a component-internal variable** (i.e. not some state or a prop value) - it's defined outside of the component and changing it (no matter where) **wouldn't cause the component to be re-evaluated**
* setTimeout is **NOT added as a dependency** because it's **a built-in API** (built-into the browser) - it's independent from React and your components, it doesn't change

**useEffect cleanup**

“React performs the cleanup when the component unmounts. However,… effects run for every render and not just once. This is why React also cleans up effects from the previous render before running the effects next time.”

The cleanup is commonly used to cancel all subscriptions made and cancel fetch requests.

  useEffect(() => {

    const identifier = setTimeout(() => {

      console.log("checking for validity");

      setFormIsValid(

        enteredEmail.includes("@") && enteredPassword.trim().length > 6

      );

    }, 500);

    return () => {

      console.log("CLEANUP");

      clearTimeout(identifier);

    };

  }, [enteredEmail, enteredPassword]);