

Vs code extension:

1. Thunder Client
2. ES7 – grahql
3. Bracket color
4. Auto rename
5. live server
6. prettier
7. Material Icon theme

What is React?

React is used to make standalone app and it made up by components.

What is components and why?

Components is made of by html, css and js for reducing repetitions of user interface.

Why:

1. Reusability = don’t repeat yourself
2. Separation of concerns = don’t do too many things in one at same place

How is a component built?

* Combine = html + CSS + JS => React
* It give’s declarative approach of building app

Setup :

1. npx create-react-app app\_name
2. cd app\_name
3. npm start

Name : File name should be in Camel case

**Props are used to pass data**, whereas state is for managing data.

Data from props is read-only, and cannot be modified by a component that is receiving it from outside.

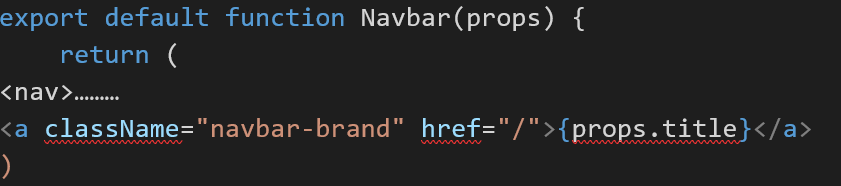
State data can be modified by its own component, but is private (cannot be accessed from outside).

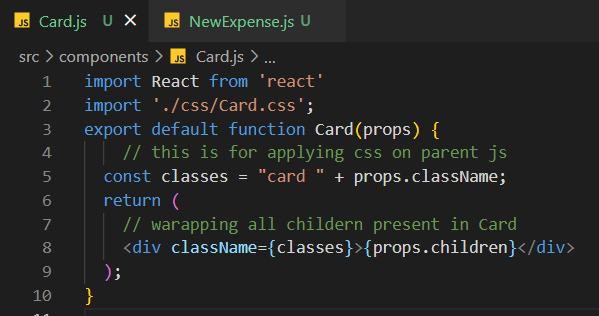
Props and prototypes :

Rfc short cut for creating react function

Imptr for prototype

Used to pass the data in component and prototypes is used to validate props types.



* How to combine css for more componets : pass props to componets and get props as className and combine and then return
* 

First Hooks :

1. useState() => it’s for maintain state of variable if you want to update your variable then use useState.
2. If we have multiple state then you can use different approach:
3. const[enterTitle,setEnterTitle] = useState(' ');
4. const[enterAmount,setEnterAmount] = useState(' ');
5. const[enterDate,setEnterDate] = useState(' ');
6. //  const [enterInput,setInput] = useState({
7. //     enterTitle:'',
8. //     enterAmount:'',
9. //     enterDate:''
10. //  })

const titleHandler = (event) => {

    setEnterTitle(event.target.value);

    // setInput({

    //     ...enterInput,

    //     enterTitle:event.target.value

    // });

    // setInput((prevState)=>{

    //     return {

    //     ...prevState,

    //     enterTitle:event.target.value

    //     }

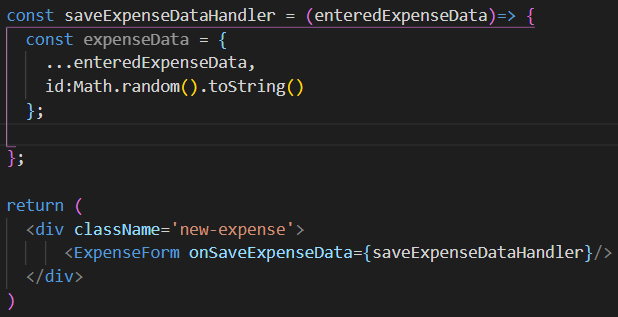
    // })

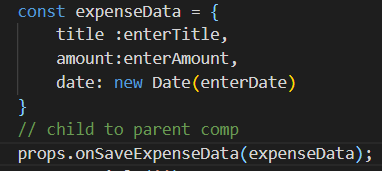
 }

Here we have three approaches first is use multiple state for each input and second use array and set all input in that and use spread operator for previous value and third way is to use prevState instead of spread operator because it run bydefault by react.

1. How to pass props child to parent :

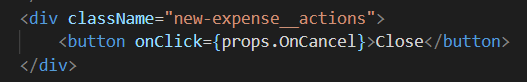
This is child component and you are passing this to parent so create one of the function as onSave… in parent so that parent will able to use from child





1. Parent to child





Style to components :

1. External way
2. Inline in tags
3. Dynamic inline style:

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

.form-control.invalid input{

  border-color: red;

  background: #ffd7d7;

}

.form-control.invalid label{

  color: red;

}

1. In external CSS lets we have many components then it may afftect to other tags so we have avoid this so use style componenets

* npm install --save styled-components
* import styled from 'styled-components';
* const Button = styled.button`
* font: inherit;
* padding: 0.5rem 1.5rem;
* border: 1px solid #8b005d;
* color: white;
* background: #8b005d;
* box-shadow: 0 0 4px rgba(0, 0, 0, 0.26);
* cursor: pointer;
* &:focus {
* outline: none;
* }
* &:hover,
* &:active {
* background: #ac0e77;
* border-color: #ac0e77;
* box-shadow: 0 0 8px rgba(0, 0, 0, 0.26);
* }
* `;
* export default Button;

Use dynamic style to stylecomponents using props :

const FormControl = styled.div`

  margin: 0.5rem 0;

& label {

  font-weight: bold;

  display: block;

  margin-bottom: 0.5rem;

  color: ${props => (props.invalid ? 'red':'black')};

}

`}

 <FormControl invalid={!isValid}>

       <label>Course Goal</label>

        <input

         type="text" onChange={goalInputChangeHandler} />

 </FormControl>

1. Use css modules so that your css and JS will be sepreated

Rename Button.css to Button.module.css

Import this css in Button.js like :

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

    <button type={props.type} className={styles.button} onClick={props.onClick}>

1. Dynamic way of css module

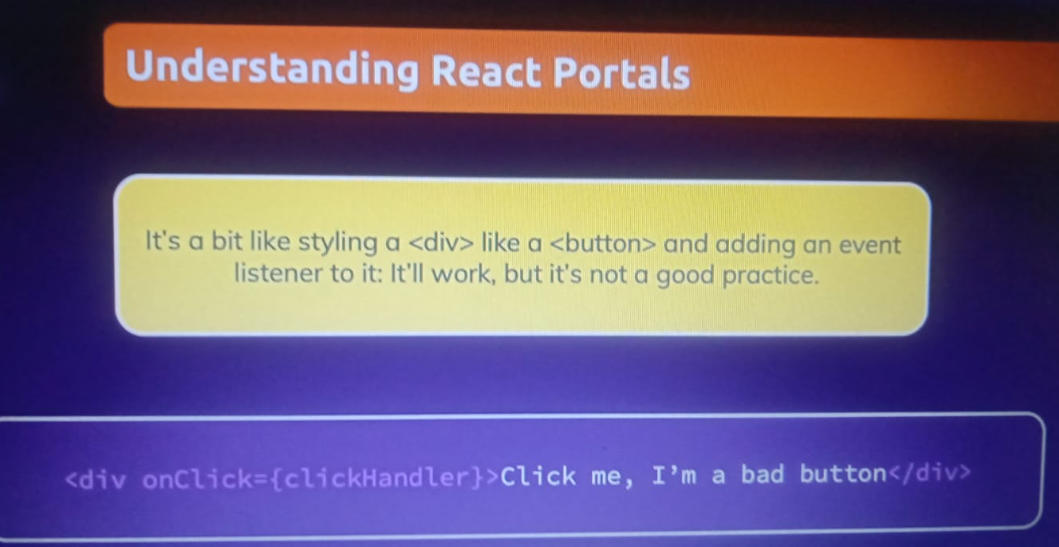
import styles from  './CourseInput.module.css';

       <div className={`${styles['form-control']} ${!isValid && styles.invalid}`}></div>

**React Fragment:**

Use JSX if we have more than two children so use fragment = > <></> or <Fragment></Freagment>

**React Portals:**



**You can use error, models popup in outside of root so use portals:**

import React from 'react'

import Button from '../Button/Button';

import Card from '../Card/Card';

import ReactDOM from 'react-dom';

import classes from './ErrorModal.module.css';

const Backdrop = (props) => {

    return <div className={classes.backdrop} onClick={props.onConfirm}/>;

};

const ModalOverlay = (props) => {

    return (

<Card className={classes.modal}>

            <header className={classes.header}>

                <h2>{props.title}</h2>

            </header>

            <div className={classes.content}>

                <p>{props.message}</p>

            </div>

            <footer className={classes.actions}>

                <Button onClick={props.onConfirm}>OKAY</Button>

            </footer>

        </Card>

    );

};

const ErrorModel = (props) => {

  return (

    <>

    {

      ReactDOM.createPortal(

        <Backdrop onConfirm={props.onConfirm}/>,

        document.getElementById('backdrop-root')

      )

      }

      {

      ReactDOM.createPortal(

        <ModalOverlay onConfirm={props.onConfirm} title={props.title} message={props.message}/>,

        document.getElementById('overlay-root')

      )

      }

    </>

  );

};

export default ErrorModel

**in index.html**

<div id="backdrop-root"></div>

    <div id="overlay-root"></div>

    <div id="root"></div>

**Use useRef :**

**If we don’t to use useState hook you can use useRef like with useState you will have to write more code but if you want to only read data and not to change anything then you can go with useref**

  const ename = useRef();

  const eage = useRef();

<input

         type="text" ref={ename}/>

         <label>Add Age</label>

         <input

         type="number"  ref={eage}/>

         </div>

const addUserHandler = event => {

      event.preventDefault();

      const rname = ename.current.value

      const rage = eage.current.value;

if(+rage <1){

        setError({

          title:"Invalid age Input",

          message:"Please enter age in number only !!"

        });

        return;

      }

      props.onAddUser(rname,rage);

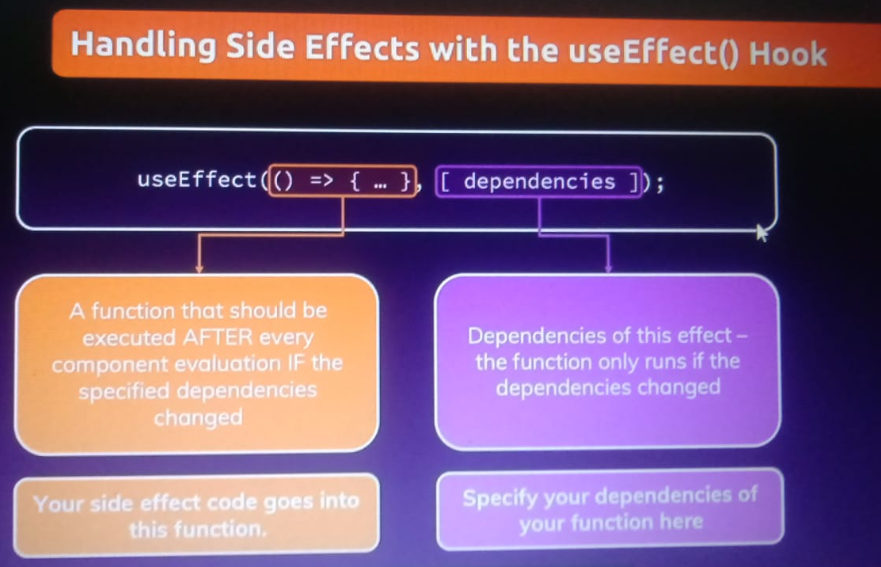
      ename.current.value = '';

      eage.current.value = '';

}

**Here it is uncontrolled components because it’s not controlled by react cause you set value like normally not by state so it’s not good but benefit like has less code**

**UseEffect Hook :**



**It is used when we want to do task only onces like let say if user is logged then there data store in localstorage and if we refresh then it will persist we [] it is run onces only**

useEffect(()=>{

    const loggedUser = localStorage.getItem('isLoggedIn');

    if(loggedUser ==='1'){

      setIsLoggedIn(true);

    }

  },[]);

useEffect(()=>{

    setFormIsValid(

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

    );

  },[enteredEmail,enteredPassword]);

**Here we have two dependency as it needed when state change of email and password and this setformisvalid is re run**

**UseReduer :**

//The useReducer Hook is similar to the useState Hook.

//It allows for custom state logic.

//If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

//useReducer(<reducer>, <initialState>)

//The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.

//The useReducer Hook returns the current stateand a dispatchmethod.

// working  : we have state used manage state of variable and dispatcher is used for dispatched your actions

// and make use by state

UseContext :

