Spread operator:

Spread syntax (`...`) allows an iterable such as an array expression or string to be expanded in places where zero or more arguments (for **function calls**) or elements (for **array literals**) are expected, or an object expression to be expanded in places where zero or more key-value pairs (**for object literals**) are expected.

It is commonly used when you want to add a new item to a local data store, or display all stored items plus a new addition.

```
let parts = ['shoulders', 'knees'];

let lyrics = ['head', ...parts, 'and', 'toes'];

//  ["head", "shoulders", "knees", "and", "toes"]
```

```
let obj1 = { foo: 'bar', x: 42 };

let obj2 = { foo: 'baz', y: 13 };

let mergedObj = { ...obj1, ...obj2 };

// Object { foo: "baz", x: 42, y: 13 }
```

Notice the properties that did not match were combined, but the property that did match, `foo`, was overwritten by the last object that was passed,

```
let array = [...obj1]; // TypeError: obj is not iterable,can't make array.
```

Any argument in the argument list can use spread syntax, and the spread syntax can be used multiple times.

```
function myFunction(v, w, x, y, z,a) { console.log(v+w+x+y+z+a)}
let args = [0, 1];
myFunction(-1, ...args, 2, ...[3,-2]); // 3
```

```
let arr = [1,2,3];

let arr2 = arr;
arr2.push(4);
console.log(arr);  //1,2,3,4
console.log(arr2); //1,2,3,4
//arr and arr2 shares same address

let arr2 = [...arr]
arr2.push(4);
console.log(arr); // 1,2,3
console.log(arr2); //1,2,3,4

let obj = {
   name: "Udai",
   add : {
      country: "India",
      state : {
         code : "DL",
         pin : "111111"
      }
   }
}

let obj2 = obj;
obj2.name = "abcd" //updated in obj as well

let obj2 = {...obj}
obj2.name = "abcd"
console.log(obj); //not updated in obj
obj2.add.country = "abcd" //updated in obj as well
```

let obj2 = {...obj,add:{...obj.add}}
obj2.add.country = "abcd" //not updated in obj

let obj2 = {...obj,add:{...obj.add,state:{...obj.add.state}}}
obj2.add.state.code = 10

Shortcut to provide unique address to all objects:
let obj2 = JSON.parse(JSON.stringify(obj));
obj2.add.state.code = 10; //not updated in obj

Rest operator:

Rest syntax looks exactly like spread syntax. In a way, rest syntax is the opposite of spread syntax. Spread syntax "expands" an array into its elements, while rest syntax collects multiple elements and "condenses" them into a single element.A function definition can have only one `...`*restParam*.The rest parameter must be the last parameter in the function definition.

```
function myFun(a,  b, ...manyMoreArgs) {
  console.log("a", a) //a, one
  console.log("b", b) //b, two
  console.log("manyMoreArgs", manyMoreArgs) // manyMoreArgs,
//["three", "four", "five", "six"]
}

myFun("one", "two", "three", "four", "five", "six")
```

Destructuring:

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

In arrays:
Square brackets used in destructuring for arrays
let arr = ["Hi","I","am","Udai"];

//no need of separate variable declaration
let [a,b,c,d] = arr //a='Hi',b='I',c=am,d='Udai'
let [a,b] = arr //a='Hi',b='I''
let [a,b,,d] = arr //a='Hi',b='I',d='Udai'
let [a,b,c,d,extra='Hlo'] = arr //a='Hi',b='I',c=am,d='Udai',extra='Hlo'

A variable can be assigned its value via destructuring, separate from the variable's declaration.
let a, b;
[a, b] = [1, 2];

Default values
A variable can be assigned a default, in the case that the value unpacked from the array is undefined.

let a, b;
[a=5, b=7] = [1];
console.log(a); // 1
console.log(b); // 7

Destructuring in objects:
Curly brackets used in destructuring for arrays

let a, b;

({a, b} = {a: 1, b: 2}); //valid
{a, b} = {a: 1, b: 2} // Invalid as the {a, b} on the left-hand side is considered a block and not an object literal.
const {a, b} = {a: 1, b: 2} //valid

const {a: aa = 10, b: bb = 5} = {a: 3}; //Assigning to new variables names and providing default values -> aa=3 and bb=5

let obj = {
name : "Udai",
state : "Delhi",
country : "India"
}

Without destructuring
M1:let name = obj.name;
M2: let state = obj["state"];

With destructuring:
let {name,state,country} = obj
let {name,country,extra="default value"} = obj
//Constraint:name of new variable should match with name of object key

let {name:firstname,state,country,extra="default value"} = obj
//providing different name to a variable instead of object key name

Nested objects:

```
let obj = {
    name:"Udai",
    add: {
        country:"India",
        state:{
            code:"DL",
            pin:"111111"
        }
    }
}
```

let {name} = obj; //destructuring name

let {add:{country}} = obj // destructuring add then destructuring country,country='India'
let {add:{country:abcd}} = obj //abcd='India'

let {add:{state:{code:cd}}} = obj destructuring add then destructuring country then code,cd='DL'

Unpacking fields from objects passed as a function parameter

```
const user = {
  id: 42,
  displayName: 'jdoe',
  fullName: {
    firstName: 'John',
    lastName: 'Doe'
  }
};

function userId({id}) {
  return id;
}

function whois({displayName, fullName: {firstName: name}}) {
  return `${displayName} is ${name}`;
}

console.log(userId(user)); // 42
console.log(whois(user));  // "jdoe is John"
```

Setting a function parameter's default value

```
function drawChart({size = 'big', coords = {x: 0, y: 0}, radius = 25} = {}) {
  console.log(size, coords, radius);
  // do some chart drawing
}

drawChart({
  coords: {x: 18, y: 30},
  radius: 30
});
```

Note: In the function signature for drawChart above, the destructured left-hand side is assigned to an empty object literal on the right-hand side:

```
{size = 'big', coords = {x: 0, y: 0}, radius = 25} = {}
```

You could have also written the function without the right-hand side assignment. However, if you leave out the right-hand side assignment, the function will look for at least one argument to be supplied when invoked, whereas in its current form, you can call drawChart() without supplying any parameters. The current design is useful if you want to be able to call the function without supplying any parameters. The other can be useful when you want to ensure an object is passed to the function.

Combined Array and Object Destructuring

Array and Object destructuring can be combined. Say you want the third element in the array `props` below, and then you want the `name` property in the object, you can do the following:

```
const props = [
  { id: 1, name: 'Fizz'},
  { id: 2, name: 'Buzz'},
  { id: 3, name: 'FizzBuzz'}
];
const [,, { name }] = props;
console.log(name); // "FizzBuzz"
```

```
Destructure and spread operator combined
```

```
[a, b, ...rest] = [10, 20, 30, 40, 50];
console.log(rest);
// expected output: Array [30,40,50]
```

map & filter:

See from video

What is ‘react’ ?

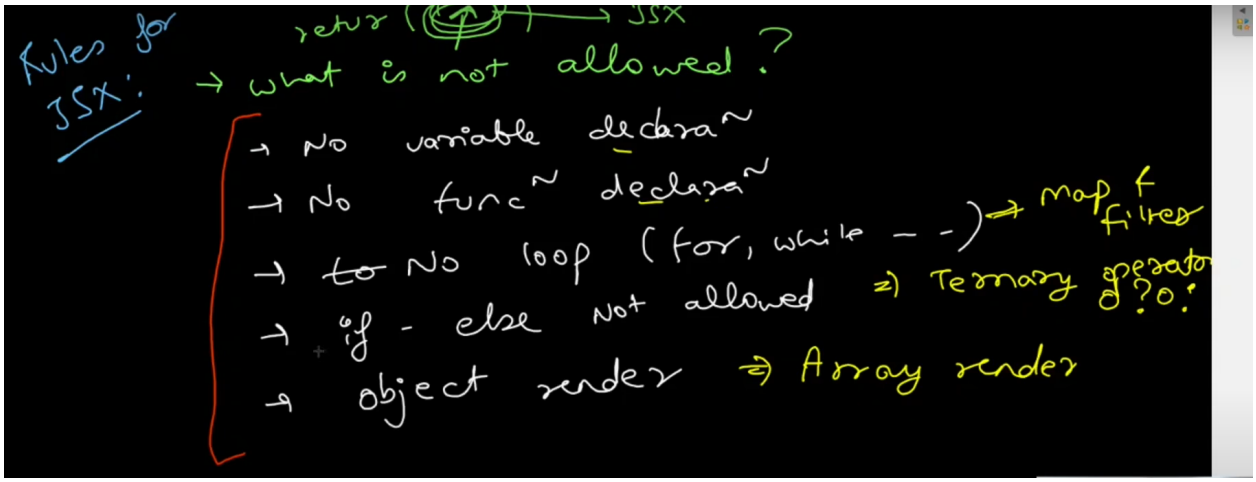
JS library used to develop UI or frontend.
React is fast as it paste code only where the changes are made not like dom where complete dom is made again even for a small change.

‘this’ for react:

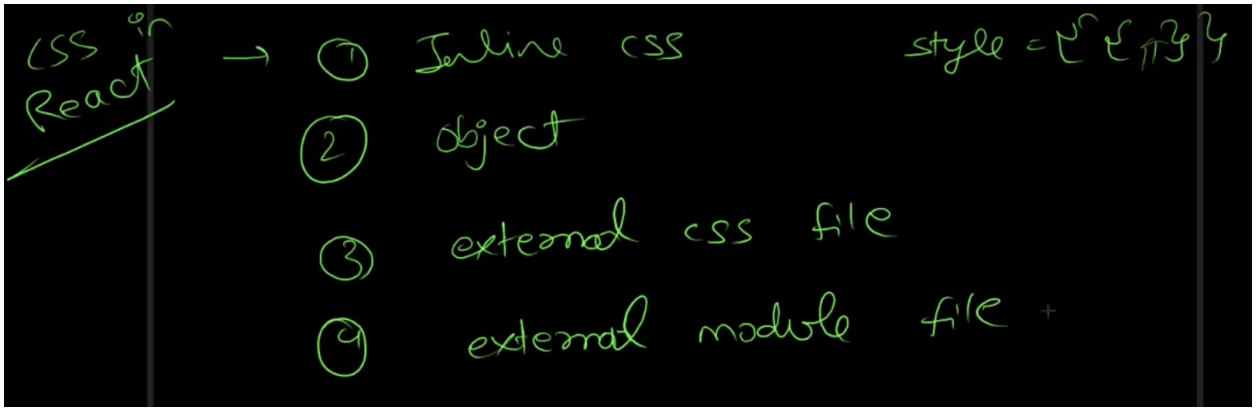
Object through which function is called.
.bind(this): It binds the listener of the function to the current class, then when you use this pointer inside of the onBeforeFirstShow function, the this pointer refer to encapsulated class and you can access to its members.

Arrow function has no own ‘this’.He sees this of parent function.While normal function has his own this.If not global this i.e. windows will be its object.

What is JSX?



In class component,render fn displays UI.Whenever there is



Change in state,render function is called again.
We can change state only by using this.setState fn,not directly.
We use arrow functions so that ‘this’ is object of that class only,otherwise we would have to bind
Changes in arrays and objects in react are always immutable

CLient side rendering vs server side rendering

Arguments in components are called props.Destructuring is used in taking arguments of components.

Class components:

We can do only one export default in a file.

State:
Render:displayed by UI.
Whenever state is changed,re render is called and changes are reflected on UI.

Immutable changes are made in react (i.e. new address is provided) so that changes are detected for rerendering.So spread operator is always used for state change.

Arrow function is used to prevent the function from running when first time rendering

Routing:

Everything that gets rendered will need to go inside the **<BrowserRouter>** element i.e. it enables routing.

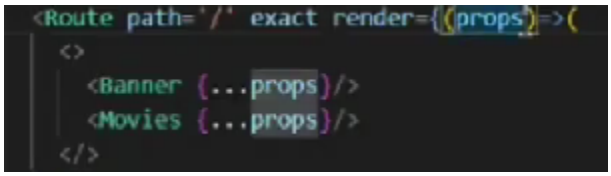
<Route> tags. are the links between the components.Two problems arises with only using <Route> tag.

(1)Multiple suitable components can be shown:This is solved by Switch tag. Switch ensures only first suitable component is rendered.Route should be placed inside the <Switch> tags.

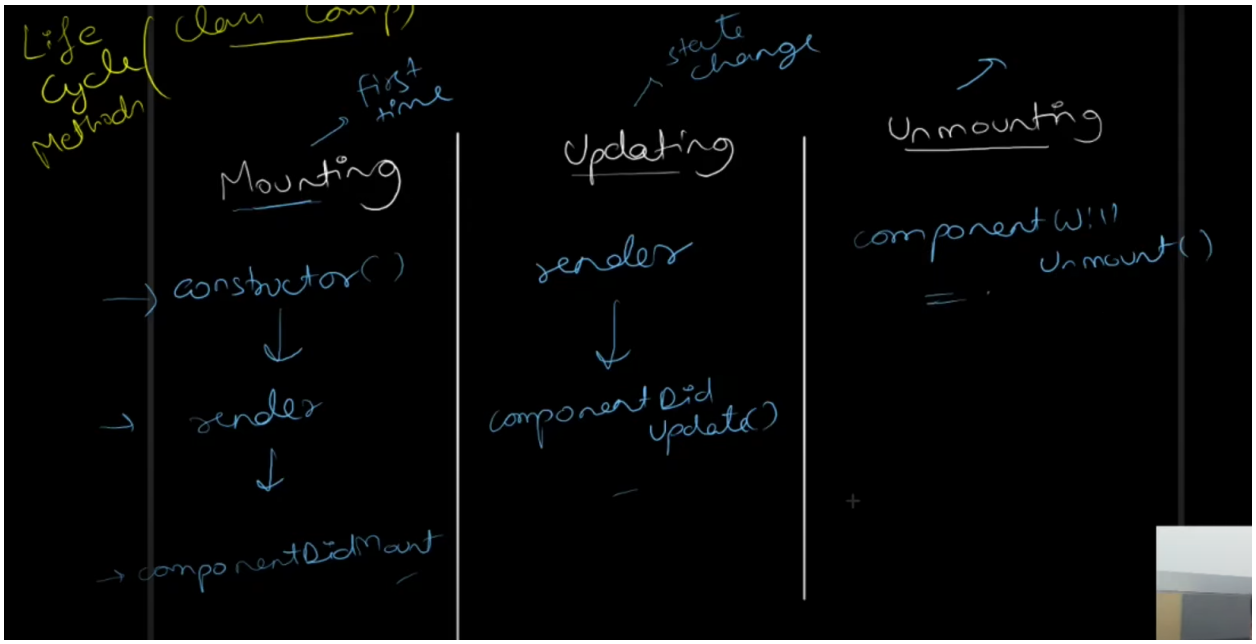
(2)
Suppose there are two paths (1)./ (2)./abc
If you type localhost/abc even using switch,first component will be shown because route tries to find first match.To overcome this problem,use **exact** keyword.

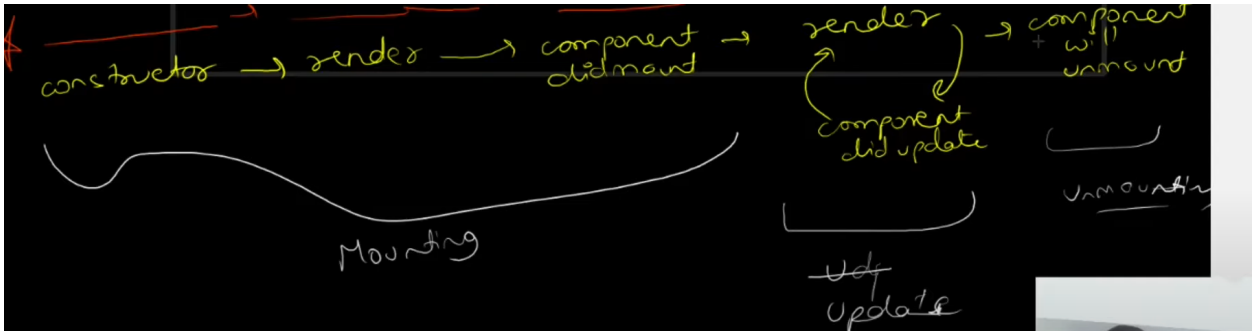
→ To route from one component to another use link tag.

→Under one route ,we can show multiple components using render and pre-defined props.



Life cycle methods (basic)





We do all side effects waala kaam (time taking tasks like api calls,async await etc)in componentDidMount so that all required is rendered at we can paste everything together on DOM to save time.

Synchronous JavaScript: As the name suggests synchronous means to be in a sequence, i.e. every statement of the code gets executed one by one. So, basically a statement has to wait for the earlier statement to get executed.

When we append the keyword “async” to the function,
The function contains some Asynchronous Execution
The returned value will be the Resolved Value for the Promise.

In case the function is returning some value, the value will be available as the resolved value from the promise.
If no value is returned, the resolved value will be “undefined”

Adding “await” before a promise makes the execution thread to wait for asynchronous task/promise to resolve before proceeding further. When we are adding the “await” keyword, we are introducing synchronous behavior to the application. Even the promises will be executed synchronously.

Axios.get is used to.0

Hooks allow us to "hook" into React features such as state and lifecycle methods.Because of this, class components are generally no longer needed.

Useeffect render ke baad chalta hai

Three variations:
No dependency array:First time componentDidMount and after every change in any state,works like componentDidUpdate
Empty dependency array:Runs first time only ,i.e. Works only like componentDidMount
Variables in dependency array:Works like componentDidUpdate only when changes are there in those variables which are in dependency array.
d
Since change is detected by address change,we always use spread operator.

Context API:

Step 1:Making a khaali dibba in context.js from which any component can extract value
const MyContext = React.createContext(defaultValue);
Creates a Context object. When React renders a component that subscribes to this Context object it will read the current context value from the closest matching Provider above it in the tree.

Step 2:Jis component ki value hame contex.js ke object mei store karani hai use MyContext.Provider se pass karna padega aur object ki value initialise karni padegi
<MyContext.Provider value={/* some value */}>

Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.The Provider component accepts a value prop to be passed to consuming components that are descendants of this Provider. One Provider can be connected to many consumers. Provi ders can be nested to override values deeper within the tree.

Step 3:How a component will extract value from MyContext.js?
Context val=useContext(context)

→ Whenever there is a change in any component,its child parents will also rerender.To avoid this export child component like this:

```
export default React.memo(Child)
```

What is uuidv4 in react?

Uuid v4 is **a React library or package that creates a universally unique identifier** (UUID). It is a 128-bit unique identifier generator

The **then()** method in JavaScript has been defined in the Promise API and is used to deal with asynchronous tasks such as an API call. Previously, callback functions were used instead of this function which made the code difficult to maintain.

Syntax:

```
demo().then(
    (onResolved) => {
        // Some task on success
    },
    (onRejected) => {
        // Some task on failure
    }
)
```

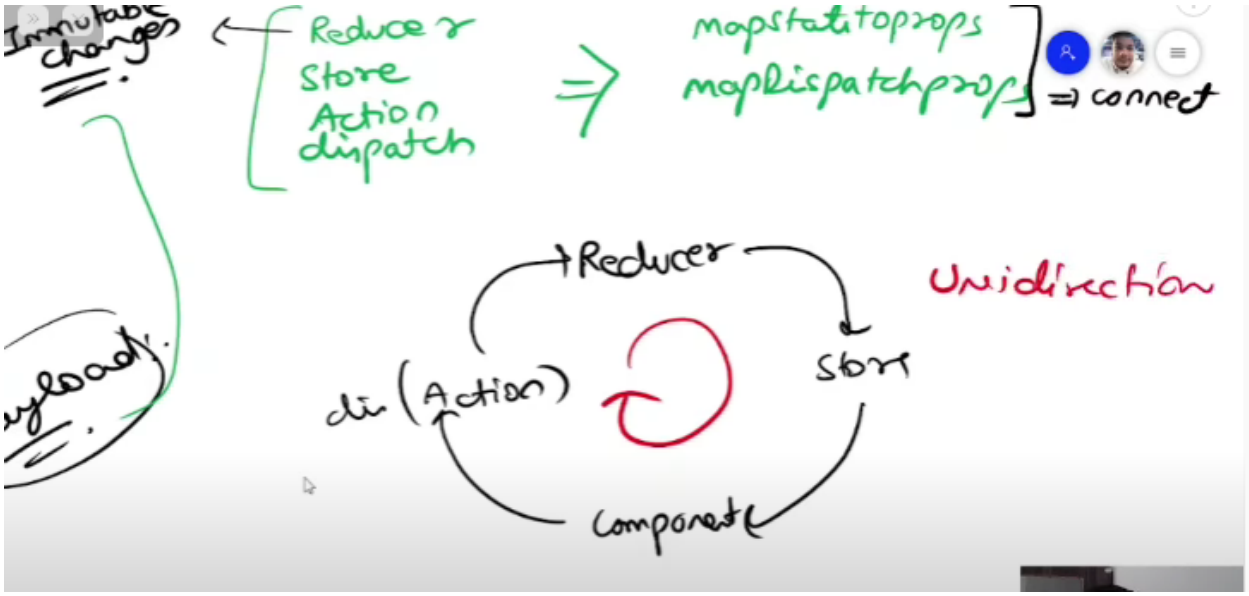
You can *listen* to a document with the onSnapshot() method. An initial call using the callback you provide creates a document snapshot immediately with the current

contents of the single document. Then, each time the contents change, another call updates the document snapshot.

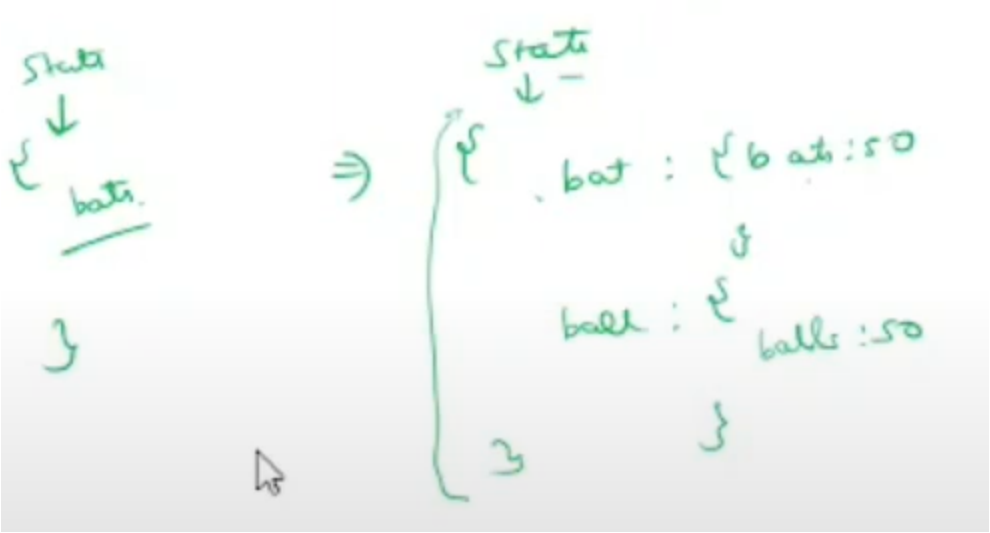
Redux

In app.js ,we wrap components that may require values from store using provider.

reducer:It defines initial state as well changes state.We do immutable changes in reducer
Store:creates global store for states
mapStateToProps is a function that gets ‘state’ as an argument from store and returns an object to component function.The function of that component gets the object as props.
The connect() function connects a React component to a Redux store.
mapDispatchToProps is used for dispatching actions(objects) to the reducer.
Action:It is a normal object having type and payload(payload is optional to write) as keys.The reducer changes state according to type of action received.



rootReducer combines two or more reducer by making an object of objects



Action.js →so that if we change action name ,we have to change at one place only

Reducer shouldn’t have async function

All files of redux are under one folder ‘redux’.

Hooks useSelector and useDispatch are alternative shortcuts to mapStateToProps and mapDispatchToProps respectively.