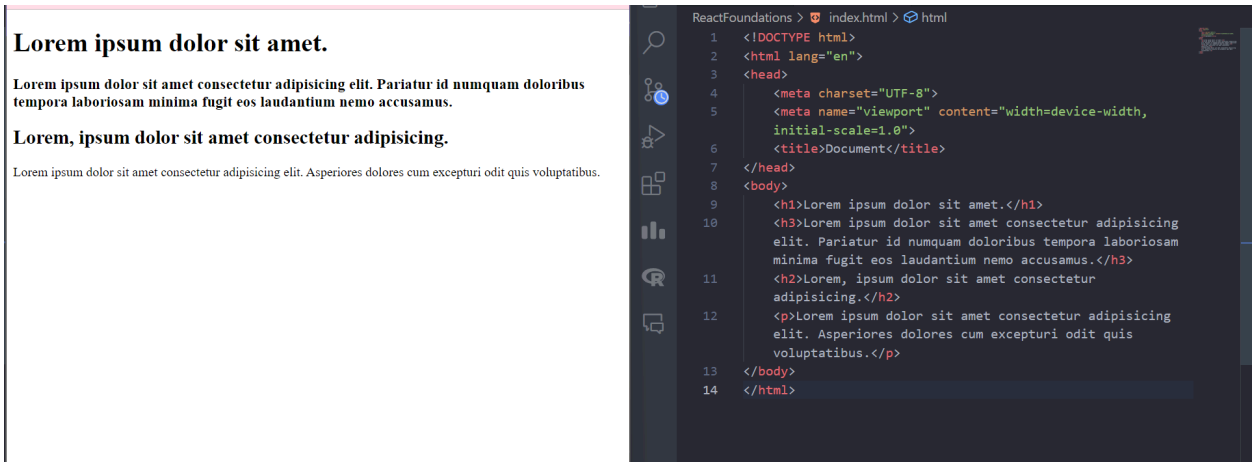
React Foundations

(Jsx, class vs className, static vs dynamic websites, State, components, re-rendering)

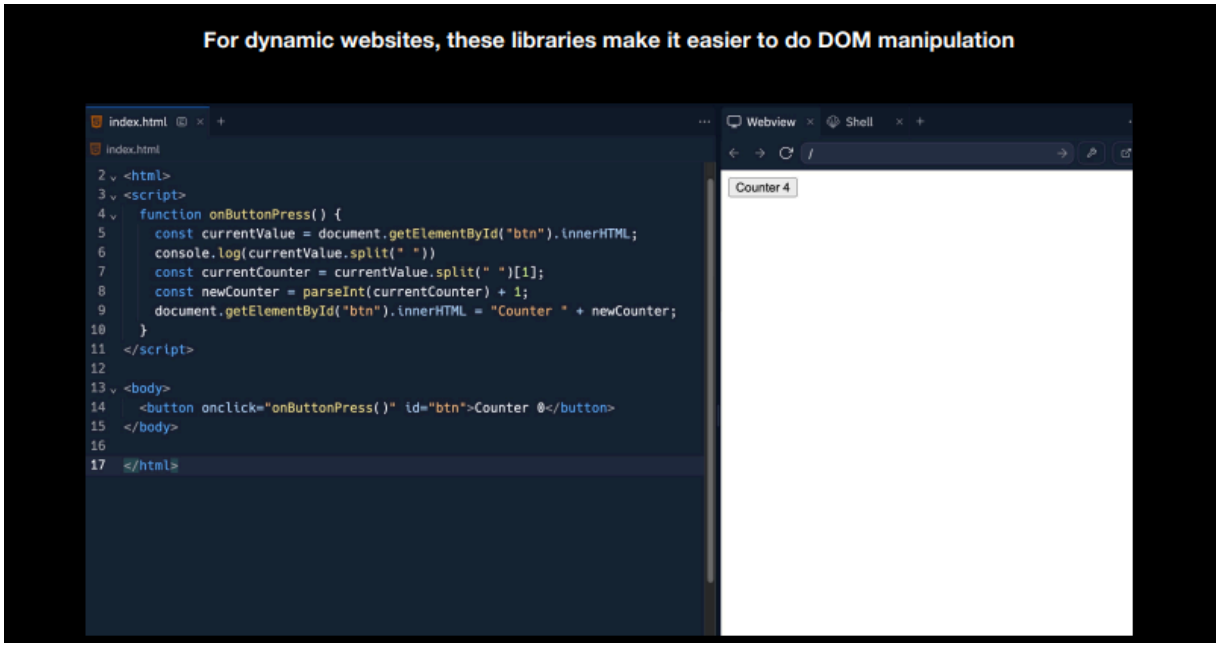
Why do we need React??

For most static websites(which doesn't change its content (html doesn't change)t once written) we don't need much.



Even dynamic websites can be written in raw html css and javascript.

For dynamic websites, these libraries make it easier to do DOM manipulation.

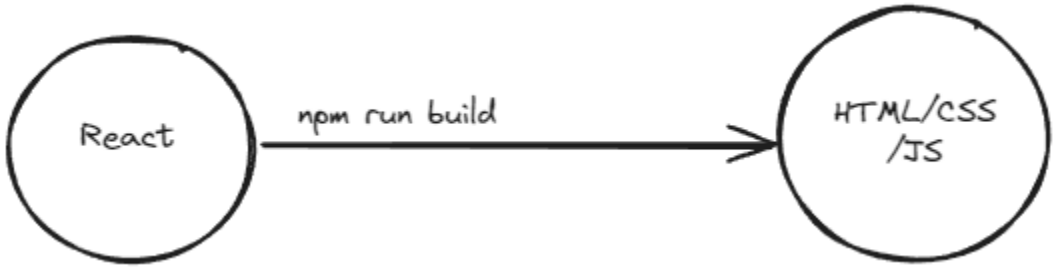


split : split the string given a delimiter.

This above example written in pure js. This is medium to write

As we write applications like linkedln which changes continuously it gets very hard to write.

React is just an easier way to write normal HTML/ CSS/ JS . It's a new syntax that under the hood gets converted into HTML/CSS/JS.



Why React??

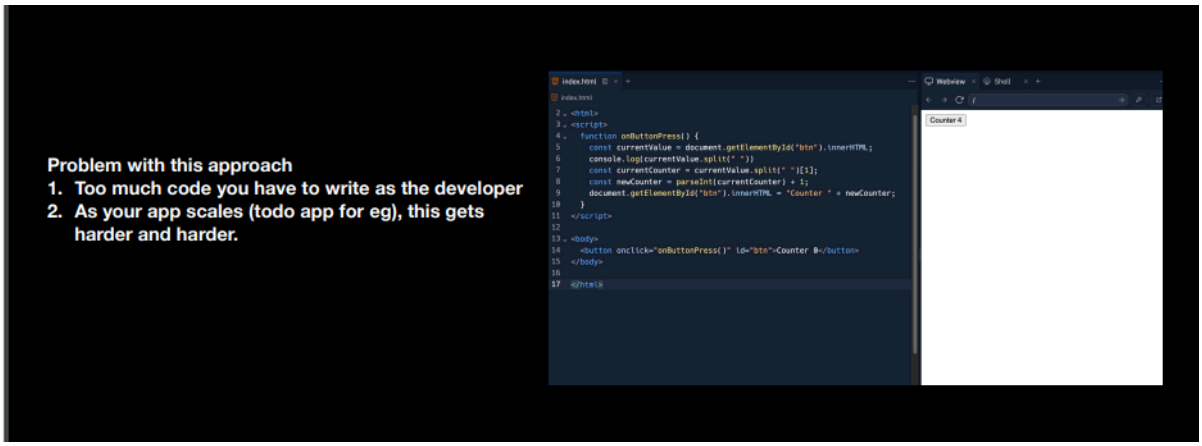
People realised it's harder to do DOM manipulation the conventional way.

There were libraries that came into the picture that made it slightly easy, but still for a very big app it's very hard (JQuery)

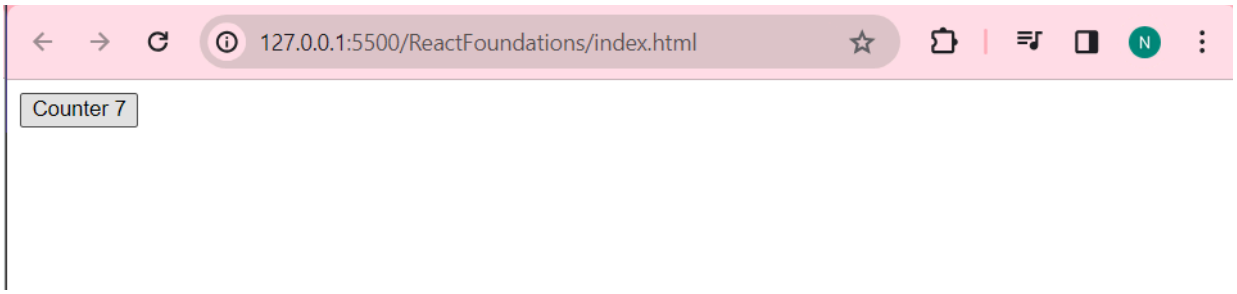
Eventually, VueJS/React created a new syntax to do frontends

Under the hood , the react compiler convert your code into HTML/ CSS/ JS.

Lets see an example



```html
<!DOCTYPE html>
<html>
<script>
  function onButtonPress() {
    const currentValue = document.getElementById("btn").innerHTML;
    console.log(currentValue.split(" "))
    const currentCounter = currentValue.split(" ")[1];
    const newCounter = parseInt(currentCounter) + 1;
    document.getElementById("btn").innerHTML = "Counter " + newCounter;
  }
</script>
<!-- parsing to integer  -->
<body>
  <button onclick="onButtonPress()" id="btn">Counter 0</button>
</body>

</html>
```



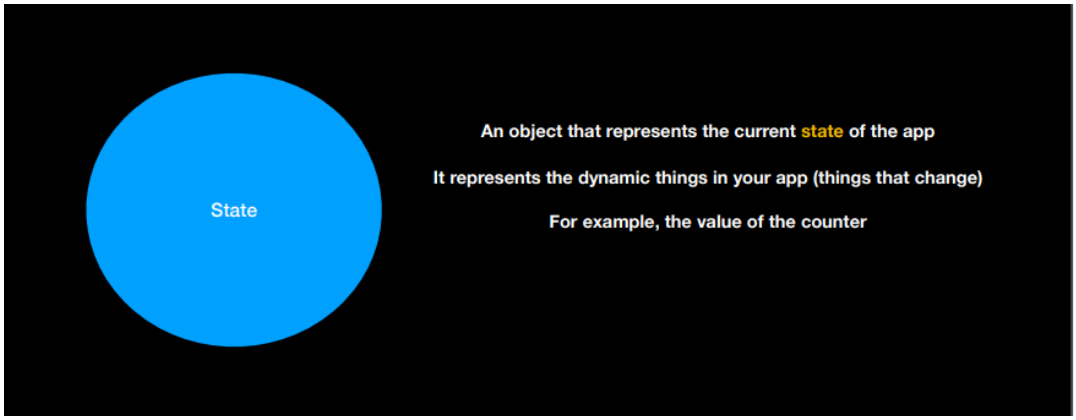
Some react jargon

When we create a react app, we need to worry about three thing

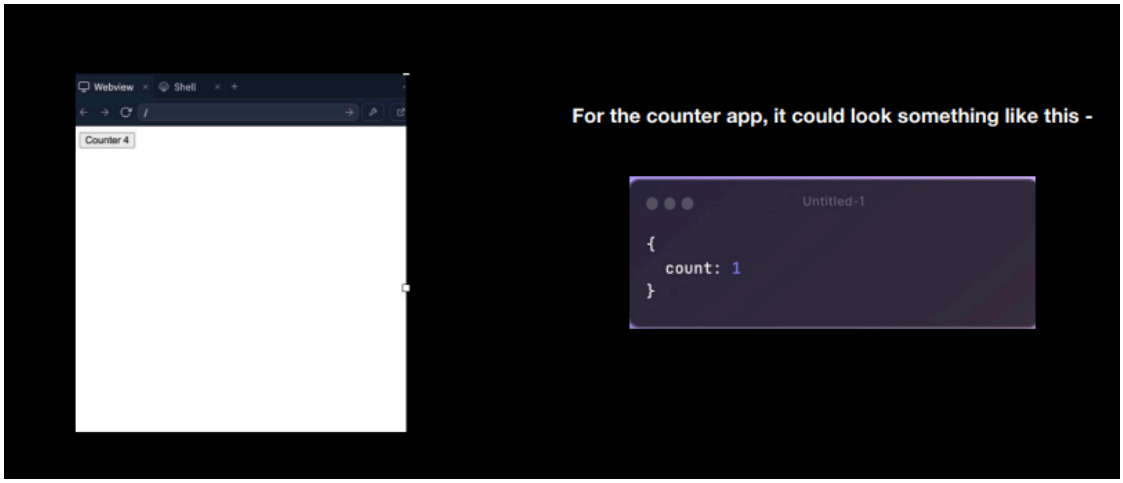
- Components
- State
- Re-rendering

Whenever we write react code we need to write State and Components code rest React handles.

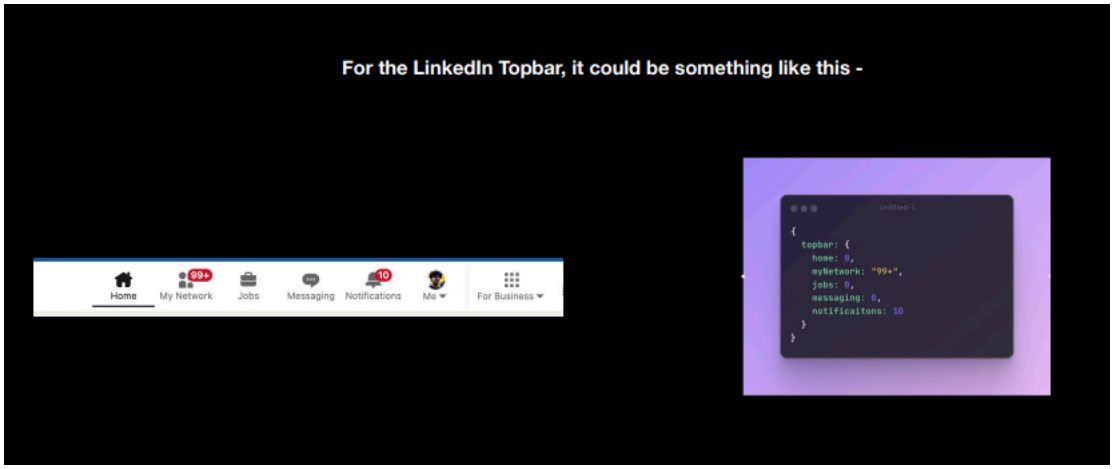
State:



State of dynamic Counter APP

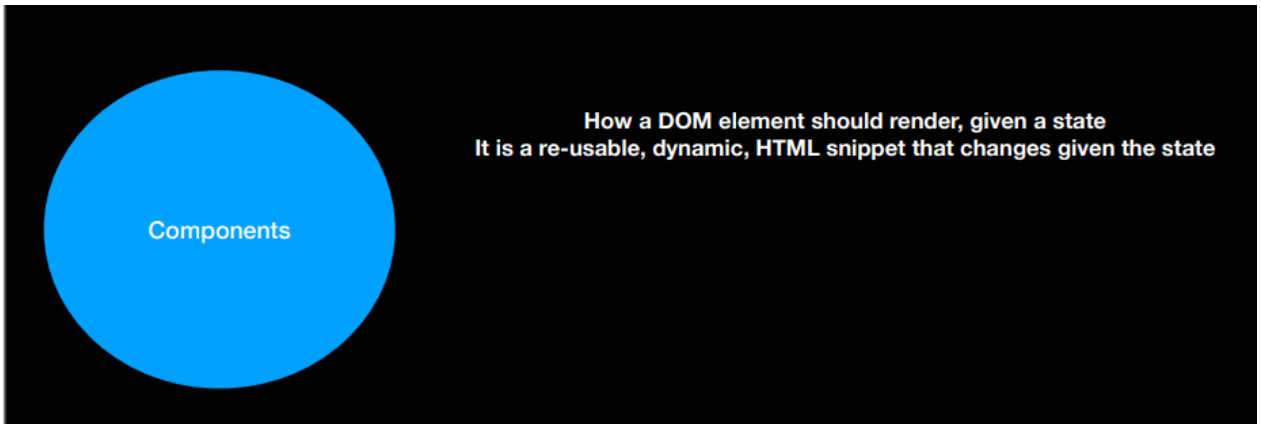


For LinkedIn Topbar

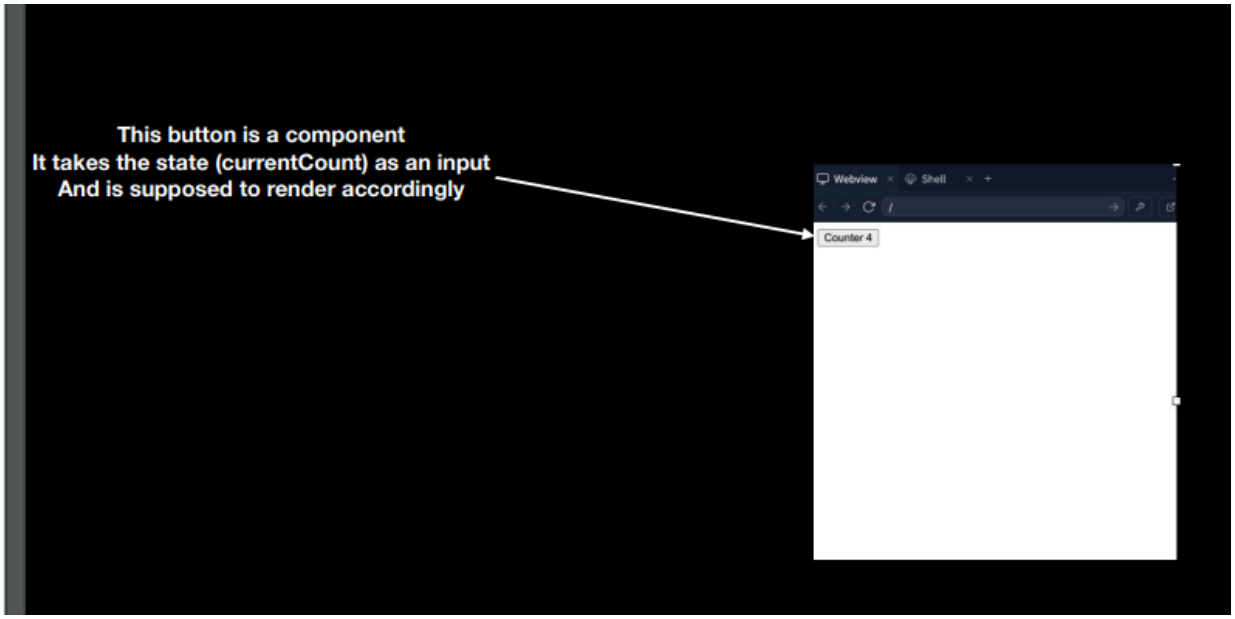


This topbar has bunch of components

Component

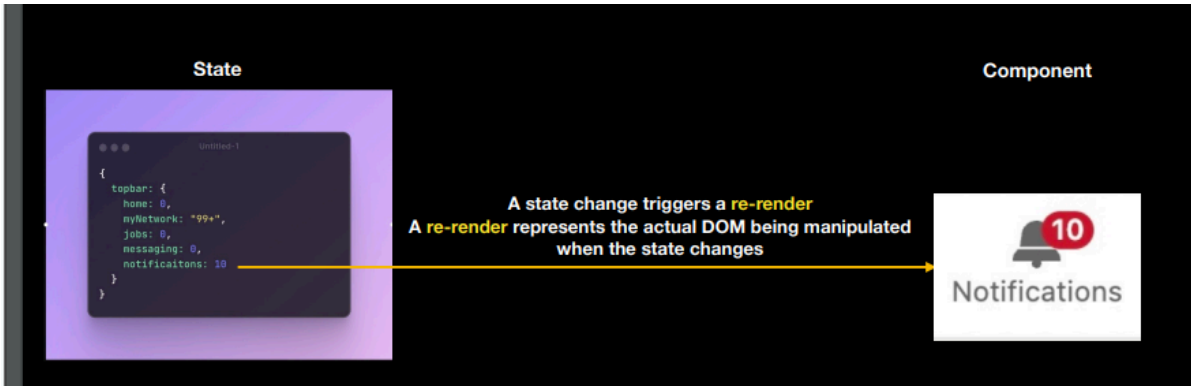


Lets see our example of Dynamic counter App

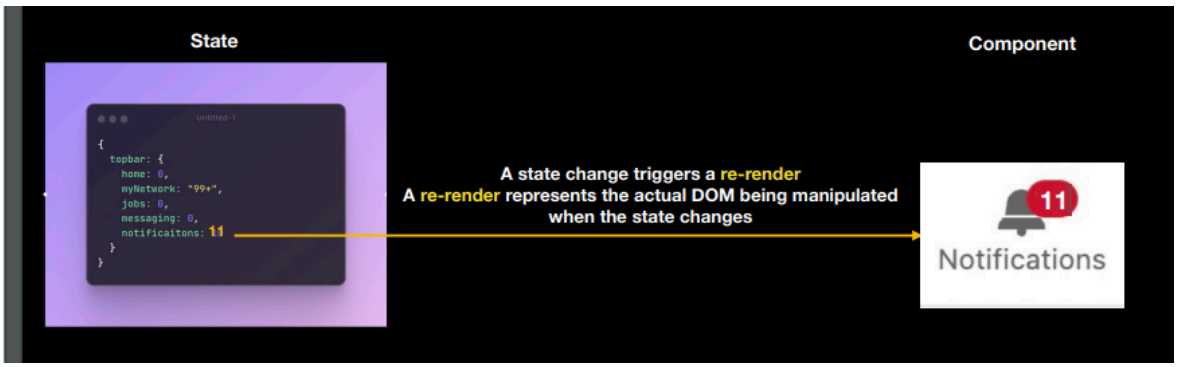


Re-Rendering

Re-rendering is the actual DOM being manipulated , when the state changes



notifications count increases to 11

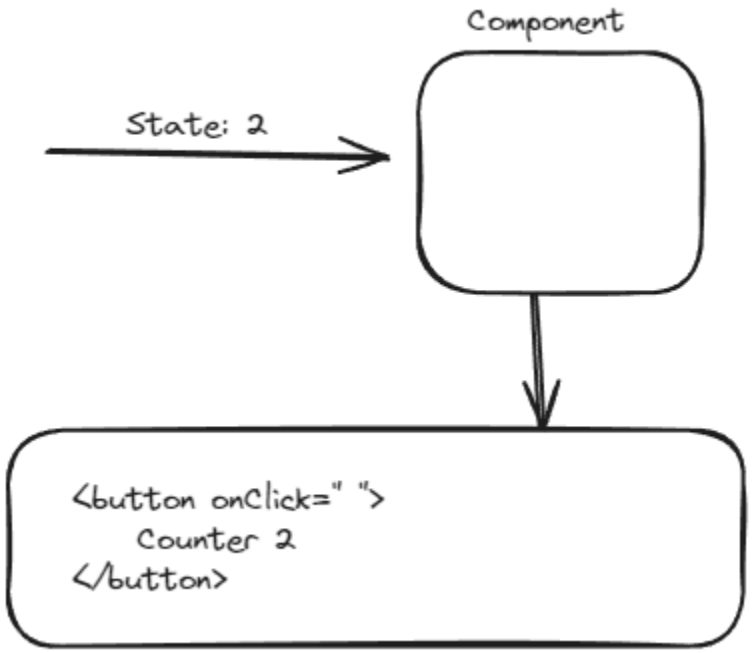


We have to usually define all our components once, and then all you have to do is update the state of your app. React takes care of re-rendering your app.

Let's create a counter app using state/components (still using Javascript and DOM manipulation**)**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div id="buttonParent">
    </div>
    <script>
        // state
```

```
        let state = {
            count: 0
        }
        function onButtonPress(){
            state.count++;
            buttonComponentReRender()
        }
        // function define -> function define -> function define ->
functioncall (makes sure control reaches inside the
buttonComponentReRender)
        // rerendering a DOM and re putting a button
        function buttonComponentReRender(){
            document.getElementById("buttonParent").innerHTML = "";
// Clearing the DOM , if we dont clear the DOM more and more button will
keep on getting appended.
            const component = buttonComponent(state.count);
            // calls button  component

document.getElementById("buttonParent").appendChild(component);
        }
        // component
        function buttonComponent(count){
            const button = document.createElement("button");
            button.innerHTML = `Counter ${count}`;
            button.setAttribute("onclick","onButtonPress()");
            return button;
        }
        buttonComponentReRender();
    </script>
</body>
</html>
```



React underthe hood work like this.

In react we define

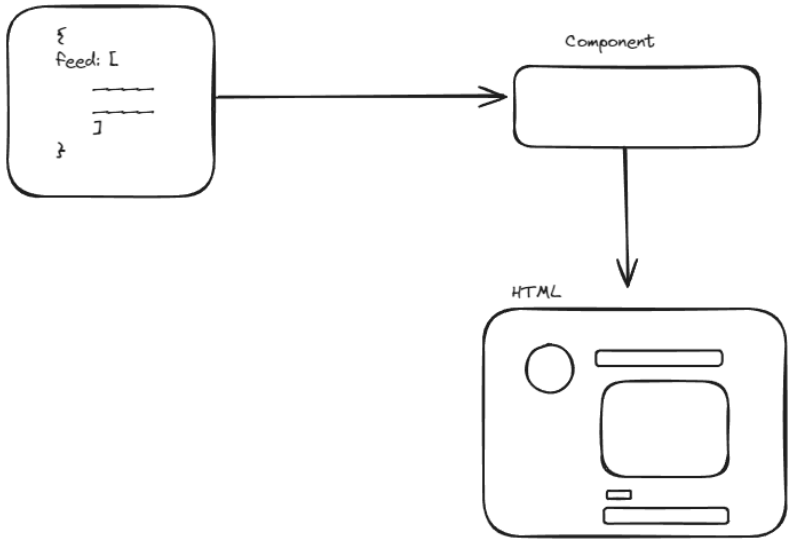
Initial state

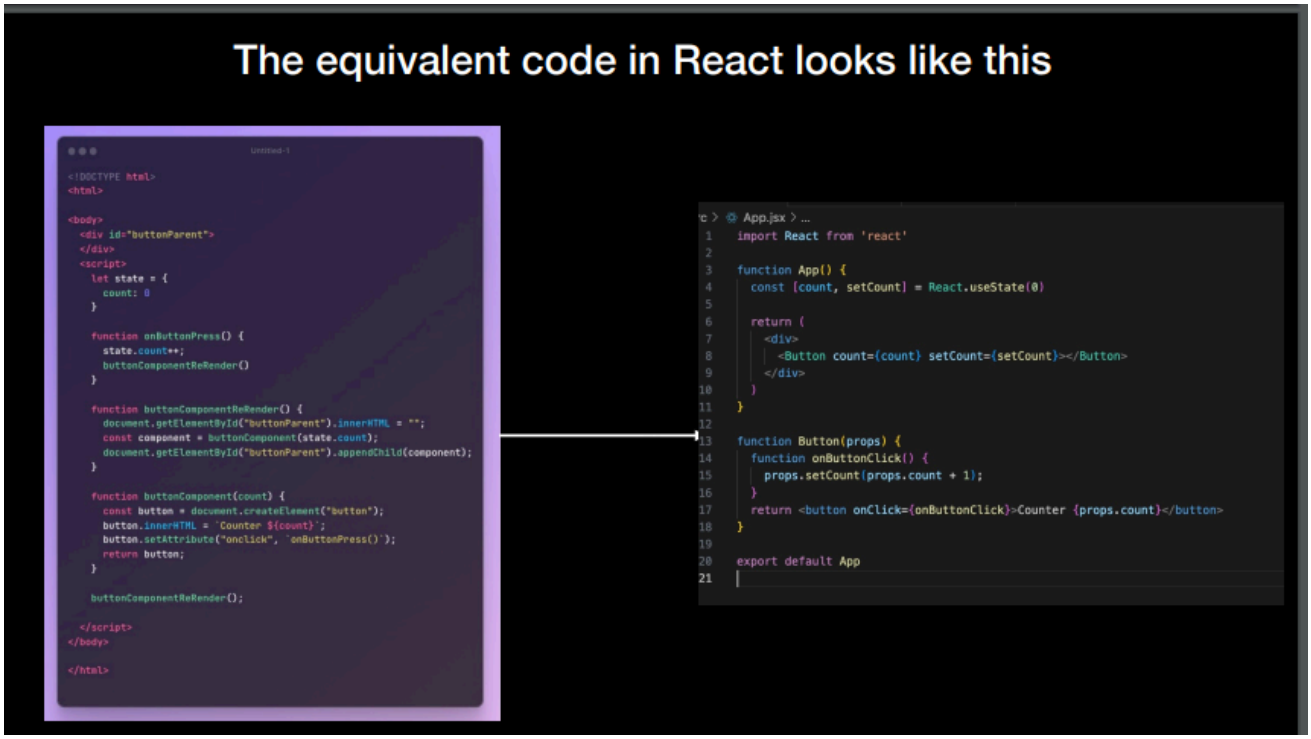
Component

A function which take props or state as input and return back actual HTML of something on our website it is component.
Here in the above example we are removing everything and adding , in real world React will be calculating the diff and then updating the state.

Dynamic: LinkedIn, Facebook, Insta

Global State
Of instagram
{
name:Thapa
feed:[
{
photoCount:_____
likeCunt:_____
Comments:{
1:" Ham tere bin ab reh nahi skte tere bina kya vajudh mera"
}
}
]
}





We need to define state in certain way otherwise React wont change state.

const arr = [1,2]

arr[0] //1

const [a,b] = arr

a //1

b //2

Same as

const a = arr[0]

const b = arr[1]

Array destructuring

Object destructuring:

const { name } = {

name : “thapa”,

age : 22

}

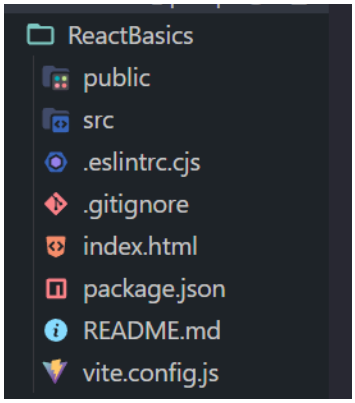
const [count, setCount] = React.useState(0)

0 is the initial state value

Lets write some React code

npm create vite@latest

This will setup basic boilerplate React application.



index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
    <title>Vite + React</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
    <!-- main.jsx is a Js file . Inside which where we can write both JS
and xml -->
  </body>
</html>
```

What is jsx ?

JSX stands for JavaScript XML. It is syntax extension for Javascript, most commonly used with React, a popular JavaScript library for building user interfaces. JSX allows you to write HTML-like code directly within JavaScript. This makes it easier to create and manage the user interface in React applications.

Here main.jsx is a js file, inside which we can write both JS and xml

main.jsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.jsx'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
    <App />
)
```

Take the root element and render this Application

App.jsx

```
import './App.css'

function App() {
  return (
      <div>
        HI THERE
      </div>
  )
}
export default App
```

To run this application we do

npm install

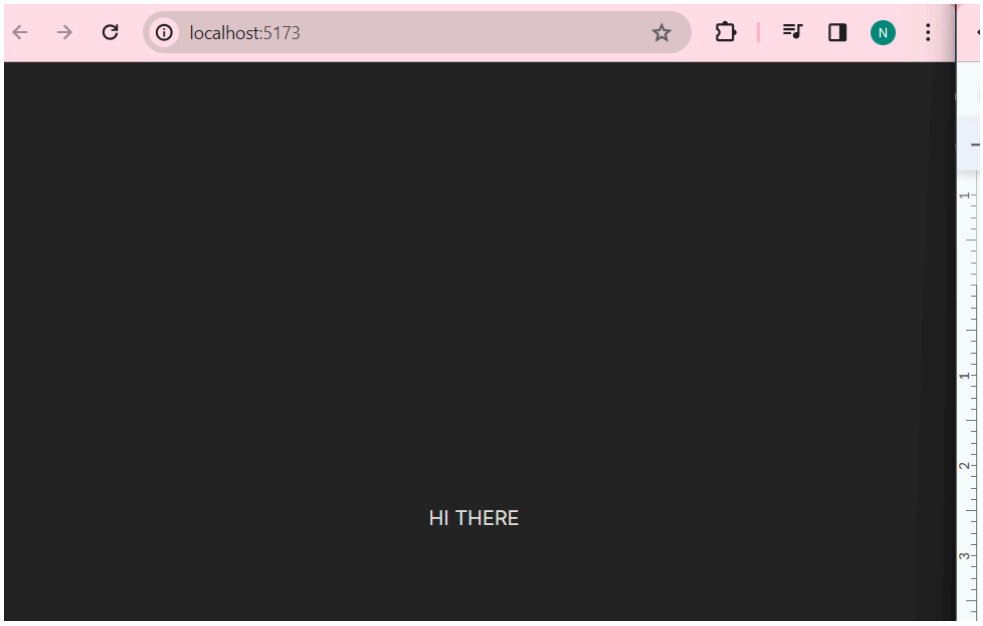
npm run dev

App.css has some css style setting

Do

npm run build to get html css js code from React code .

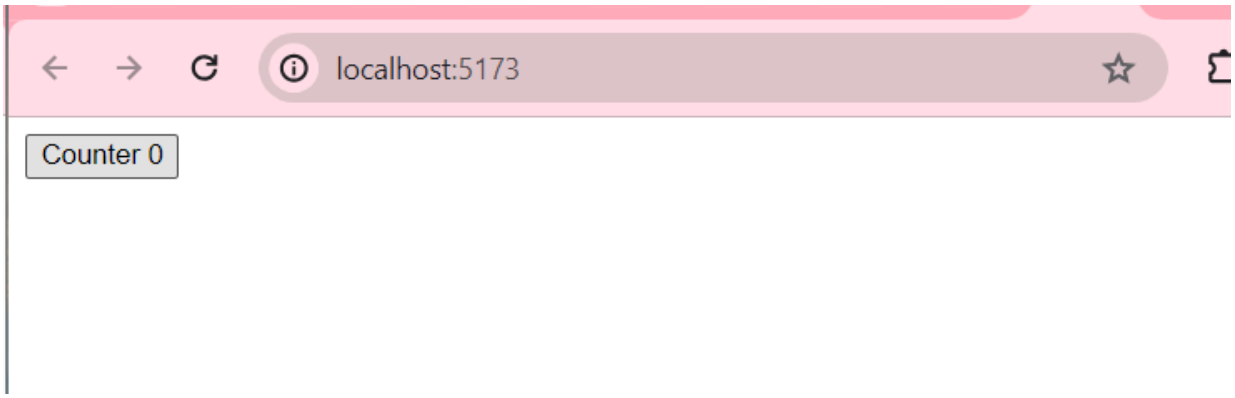
Now we can even delete our React code and we can host this **dist** in AWS and we will have fully functional code.



Now lets Create Our Counter App (Dynamic Website**)**

```
// state, components
// global state
let state = {
  count:0
}

function App() {
  return (
      <div>
        <button>Counter {state.count}</button>
      </div>
  )
```

```
}
export default App
```

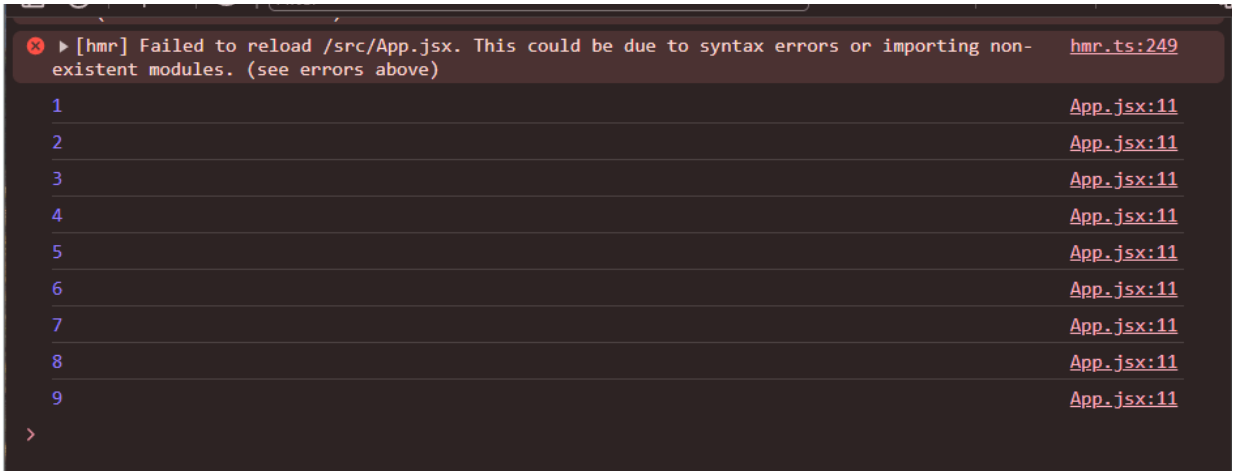


Clicking on the Counter does nothing

```
// state, components
// global state
let state = {
  count:0
}

function App() {

  function onClickHandler(){
    state.count = state.count + 1;
    console.log(state.count);
  }

  return (
      <div>
        {/* this is XML */}
        <button onClick={onClickHandler}>Counter {state.count}</button>
      </div>
  )
}
export default App
```

Although this much of code look correct but the problem is that it doesn't work correctly.

Lets see the console



So state.count value is increasing then why is it not re-rendering???
Purpose of react is that we give it state and component. And anytime the state is updated the component should be re rendered.
Why isn't component re rendering?

The reason is that React doesn't see every variable as the state variable and if we want to define the state variable we have to do it certain way react understand then only DOM will be updated.

Correct Code:

```
import { useState } from "react";

function App() {
//Defining the initial state
const [count,setCount] = useState(0);
  // [1,2]
  // setCount is a function , it dispatch something which rerender the DOM
  // Now we don't have global state variable


  function onClickHandler(){
    // it has to do something like count++ but in right way , in which
state variable are updated.
```

```
    setCount(count+1);
  }

  return (
      <div>
//defining dynamic html
        <button onClick={onClickHandler}>Counter {count}</button>
      </div>
  )
}
export default App
```

Lets see another better approach:

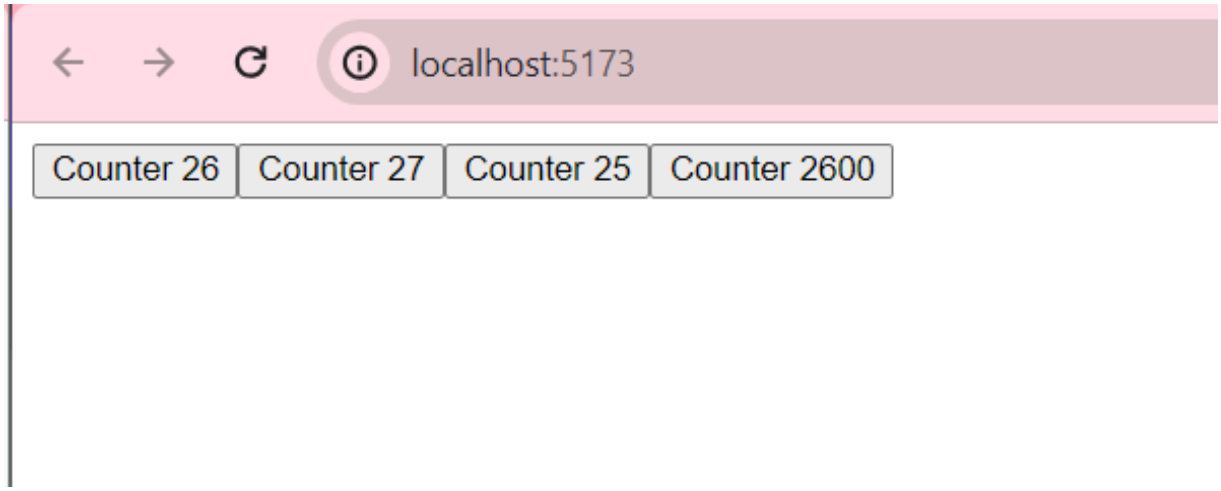
Now we can re use this component again and again

```
import { useState } from "react";
function App() {
  const [count,setCount] = useState(0);
  return (
      <div>
        <CustomButton count={count} setCount={setCount}></CustomButton>
      </div>
  )
}

// component
function CustomButton(props){
  // buttonComponent take state as input
// {
//   count:
// }
  function onClickHandler(){
      props.setCount(props.count+1);
```

```
  }
 return <button onClick={onClickHandler}>
   Counter {props.count}
  </button>
}
export default App
```

Now our component is more reusable

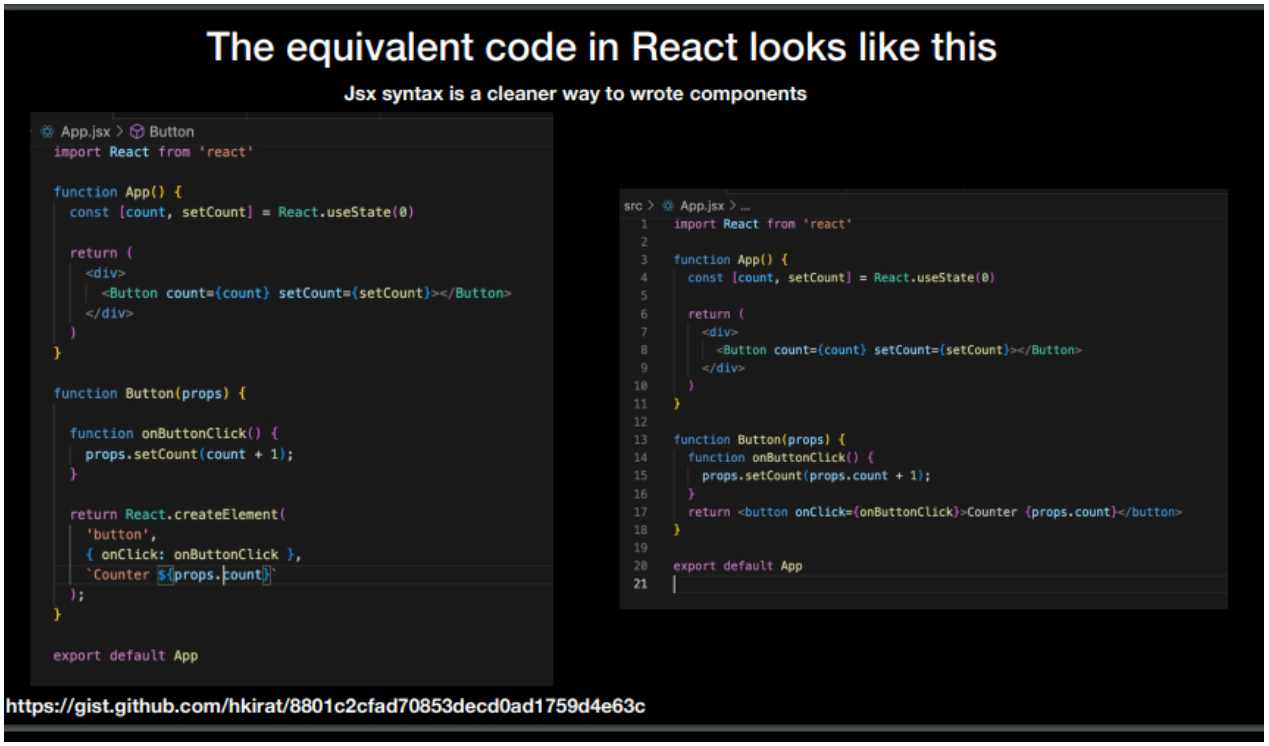
```
function App() {
  const [count,setCount] = useState(0);
  return (
      <div>
        <CustomButton count={count} setCount={setCount}></CustomButton>
        <CustomButton count={count + 1}
setCount={setCount}></CustomButton>
        <CustomButton count={count - 1}
setCount={setCount}></CustomButton>
        <CustomButton count={count * 100}
setCount={setCount}></CustomButton>
      </div>

  )
}
```



Initially React was written as

return React.createElement(‘ ’)

React was not originally written in xml way .



It is just syntactic sugar Both code works same way.

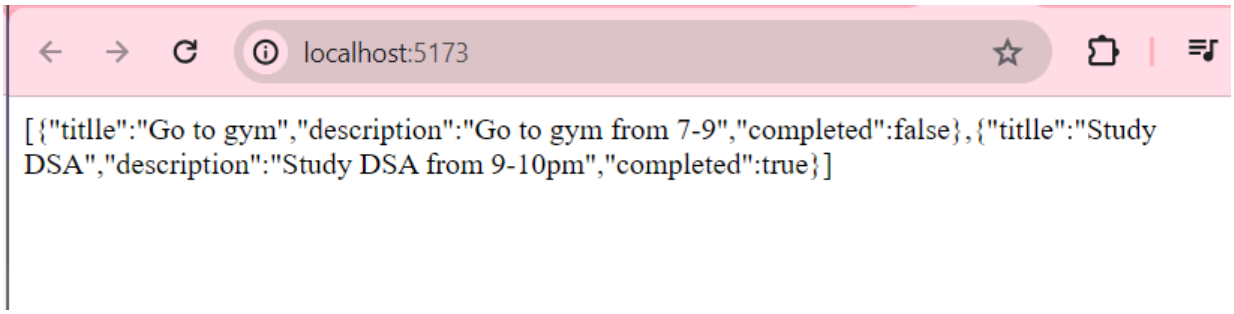
Now Lets make TODO APP

```
// todo application
// todo
// {
//    todos:[{title:"todo1",description:"first Todo",completed:false}]
// }
```

```
import { useState } from "react";

function App() {
  // complicated state
  const [todos,setTodos] = useState([{
    titlle:"Go to gym",
    description:"Go to gym from 7-9",
    completed: false
```

```
  },{
    titlle:"Study DSA",
    description:"Study DSA from 9-10pm",
    completed: true
  }]);
// How do i render it one after another
  return (
      <div>
{/* dumbs the whole array */}
        {JSON.stringify(todos)}
      </div>
  )
}

export default App
```

Currently we are rendering the State variable in a not a good way.



Lets see another approach

```
import { useState } from "react";

function App() {
  // complicated state
  const [todos,setTodos] = useState([{
    title:"Go to gym",
    description:"Go to gym from 7-9",
    completed: false
  },{
    title:"Study DSA",
    description:"Study DSA from 9-10pm",
    completed: true
  }]);
```

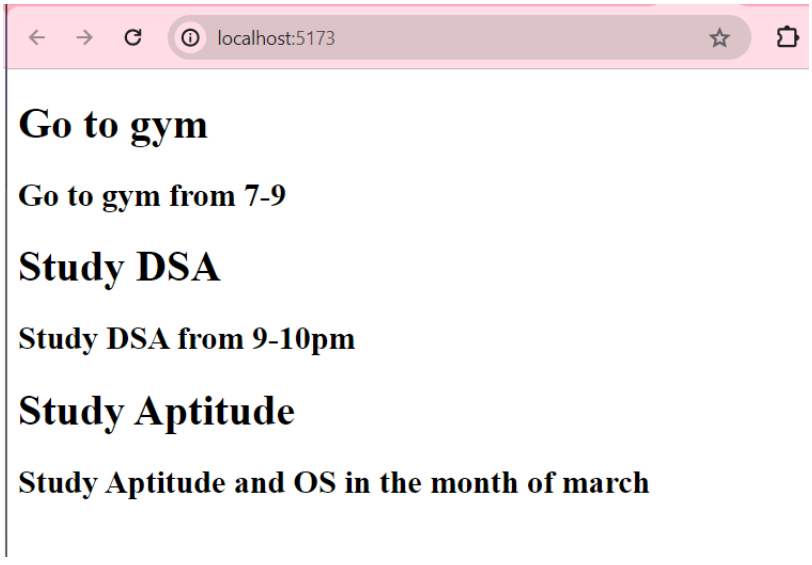
```
// How do i render it one after another
  return (
      <div>
        <Todo title={todos[0].title}
description={todos[0].description}></Todo>
        <Todo title={todos[1].title}
description={todos[1].description}></Todo>
      </div>
  )
}

// lets make a function which render a todo given,
// {
//   title,
//   description,
//   completed
// }

function Todo(props){
  return <div>
    <h1>{props.title}</h1>
    <h2>{props.description}</h2>
  </div>
}
export default App
```

The problem with this approach is that if we add third todo then we have to add manually <Todo> tag

How can i iterate over the todos state.

```
import { useState } from "react";

function App() {
  // complicated state
  const [todos,setTodos] = useState([{
    title:"Go to gym",
    description:"Go to gym from 7-9",
    completed: false
  },{
```

```
    title:"Study DSA",
    description:"Study DSA from 9-10pm",
    completed: true
  },{
    title:"Study Aptitude",
    description:"Study Aptitude and OS in the month of march",
    completed: false
  }]);
// How do i render it one after another
  return (
      <div>
        {/* <Todo title={todos[0].title}
description={todos[0].description}></Todo>
        <Todo title={todos[1].title}
description={todos[1].description}></Todo> */}
        {todos.map(function(todo){
          return <Todo title={todo.title}
description={todo.description}></Todo>
        })}
      </div>
  )
}
```



Lets add a button on clicking the button will add a todo

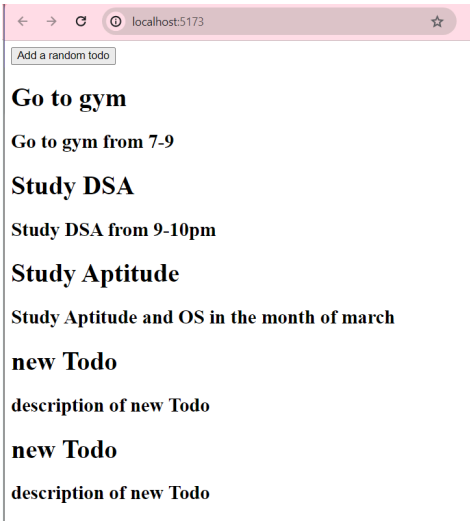
```
// this function adds a todo
  function addTodo(){
    setTodos([...todos,{
      title:"new Todo",
      description:"description of new Todo"
    }])
  }
  return (
      <div>
        <button onClick={addTodo}>Add a random todo</button>
```

Suppose todos is initially [1,2]

Now if we do [...todos] it is basically [1,2]

But if we do [...todos,3] it is [1,2,3]

It basically spread the value of array



Code:

```
import { useState } from "react";

function App() {
  const [todos,setTodos] = useState([{
    title:"Go to gym",
    description:"Go to gym from 7-9",
```

```
    completed: false
  },{
    title:"Study DSA",
    description:"Study DSA from 9-10pm",
    completed: true
  },{
    title:"Study Aptitude",
    description:"Study Aptitude and OS in the month of march",
    completed: false
  }]);

  function addTodo(){
    setTodos([...todos,{
      title:"new Todo",
      description:"description of new Todo"
    }])
  }
  return (
      <div>
        <button onClick={addTodo}>Add a random todo</button>
        {todos.map(function(todo){
          return <Todo title={todo.title}
description={todo.description}></Todo>
        })}
      </div>
  )
}
function Todo(props){
  return <div>
    <h1>{props.title}</h1>
    <h2>{props.description}</h2>
  </div>
}
export default App
```

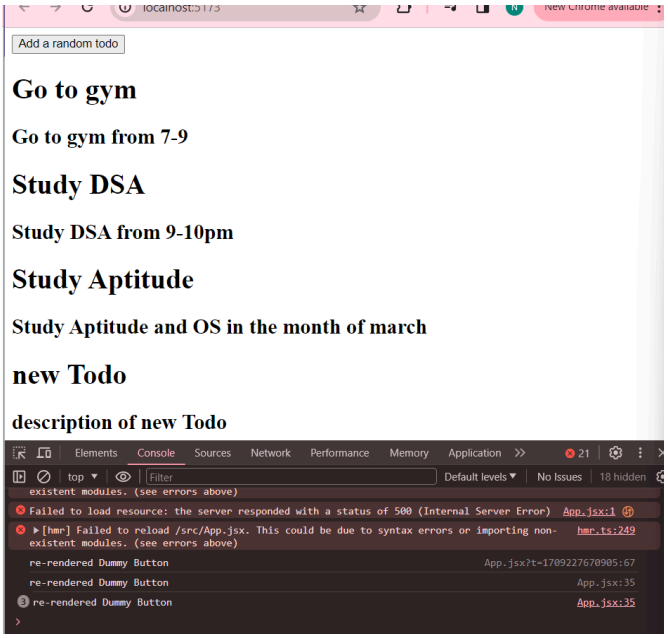
Q/Na:

1. **Any time a parent re-render its child re-render as well.**

Even a dummy button which doesn't take any input/ state still re-renders the dummybutton.

```
function App() {
  const [todos,setTodos] = useState([{
    title:"Go to gym",
    description:"Go to gym from 7-9",
    completed: false
  },{
    title:"Study DSA",
    description:"Study DSA from 9-10pm",
    completed: true
  },{
    title:"Study Aptitude",
    description:"Study Aptitude and OS in the month of march",
    completed: false
  }]);

  function addTodo(){
    setTodos([...todos,{
      title:"new Todo",
      description:"description of new Todo"
    }])
  }
  return (
      <div>
        <button onClick={addTodo}>Add a random todo</button>
        {todos.map(function(todo){
          return <Todo title={todo.title}
description={todo.description}></Todo>
        })}
        <DummyButton></DummyButton>
      </div>
  )
}
function DummyButton(){
  console.log("re-rendered Dummy Button")
  return <button>DSDSVF</button>
}
```

On pressing the Add a random todo , or re-rendering the parent the dummy button which doesn't even take a state variable as input still re-renders



React.memo() will prevent this

2. How does it monitor props?

```
  let a=1;
  window.setInterval(()=>{
    a++;
  },1000)
return (
      <div>
     <DummyButton a={a}></DummyButton>
        {/* this willl not result in re-rendering even if we continously
change the value of "a" at a set interval. Since it is not a state
variable */}
      </div>
  )
```

React is not monitoring props its monitoring state only, if we pass state as a props it will re-render.

3. **Adding css in React**

```
<button style={{
          border:10,
          padding:100
        }} onClick={addTodo}>Add a random todo</button>
```

Here we are using double curly braces, outer curly braces is for specifying Js and inner curly braces are for object.

4. **React context ,recoil , redux**
5. **State ->> useState hook**

React need to watch , we need to use , useState

[todos, setTodos] = useState([])

Todos = [setTodos([

])]

React is not watching variable, it is listening to the setTodos calls, and anytime setTodo is called with new input , it know component need to re-render, it will start the process of finding the new DOM element put it,

```
function App() {
  const [todos,setTodos] = useState([]);

  function addTodo(){
    let newTodos = [];
    for(let i=0; i<todos.length; i++){
      newTodos.push(todos[i])
    }
    newTodos.push({
      title:"asda",
      description:"asvsdafdsd"
    })
  setTodos(newTodos)
  }
  return (
      <div>
```

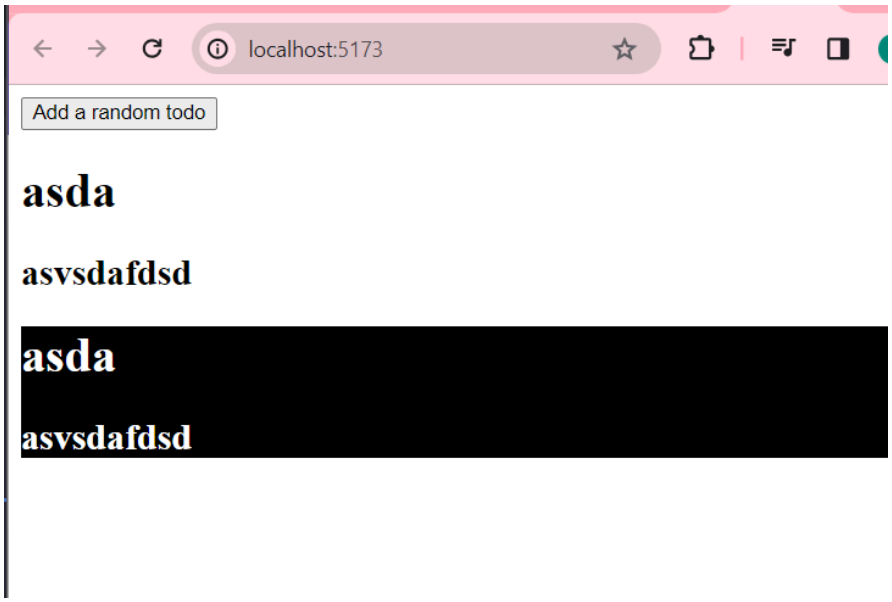
```
        <button onClick={addTodo}>Add a random todo</button>
        {todos.map(function(todo){
          return <Todo title={todo.title}
description={todo.description}></Todo>
        })}
      </div>
  )
}
```

Child variable also need to access the global state, we pass them as props

```
        <NewTodos todos={todos}></NewTodos>
        <DarkNewTodos todos={todos}></DarkNewTodos>
```

We can pass state variable as props

```
function DarkNewTodos(props){
  return <div>
    {props.todos.map(function(todo){
      return <div style={{background:"black",color:"white"}}>
      <Todo title={todo.title} description={todo.description} />
      </div>
    })}
  </div>
}
```



Re-rendering when whenever we are adding a random todo.

6. **Control flow**

App got called

addTodo got called (realize Todos state is changed)

App is again called

Todo component get called

Npm run dev : used when we are developing

(hot reloading) We want to immediately show changes

npm run build : we can distribute it.

7. **const [count, setCount] = useState(1);**

   count++;

This will generate error

Always call the setCount function if the variable is not constant dont do count++.

8. example

```
const [a, setA] = useState(0);
<Sum a={a} b={2}/>
function Sum(a, b){
  return <div>

  </div>
}
```

At high level anytime state variable changes app will re-render.

We will learn state management library to prevent unnecessary re-renders.

9. Another example

```
const [notificationCount,NotificationComponent] =useState(0);
notificationCount = notificationCount + 2;
setCNotification();
```

10. Since we know that the button count will be use in button only hence we can define it in the component itself.

```
function App() {
  const [todos,setTodos] = useState([]);

  return (
      <div>
        <Button/>
        <Button/>
        <Button/>
        <Button/>
        <Button/>
      </div>
  )
}

function Button(){
  const [count, setCount] = useState(0);
  return <button>
    </button>
  // self containerd
}
```

11. const [todos, setTodos] = useState([]);
    function addTodo(){
    let newTodos = [...todos,{
    title:”acvdsa”,
    description:”Musu musu hasai dieu malai lai ….”
    }]
    setTodos(newTodos);
    }

{} are only for expression not running whole for loop etc

```
ReactDOM.createRoot(document.getElementById('root')).render(
    <App />
)
// All our logic is inside the root
// Anything above and below we cant add or remove it
```

12. Why do we use props??

```
import { useState } from "react";

function App() {
  const [count,setCount]=useState(0)
  return (
      <div>

      </div>
  )
}

function Todo(props){
  return <div>
  asdfg
  </div>
}
export default App
```

Suppose we want to use state variable which is defined on App function inside the Todo function then we will use props

Const a = [1,2,3,4]

a[0] = 100;

Array is constant means its address is constant.

13. hook we have to start with use while naming it.