React Foundation

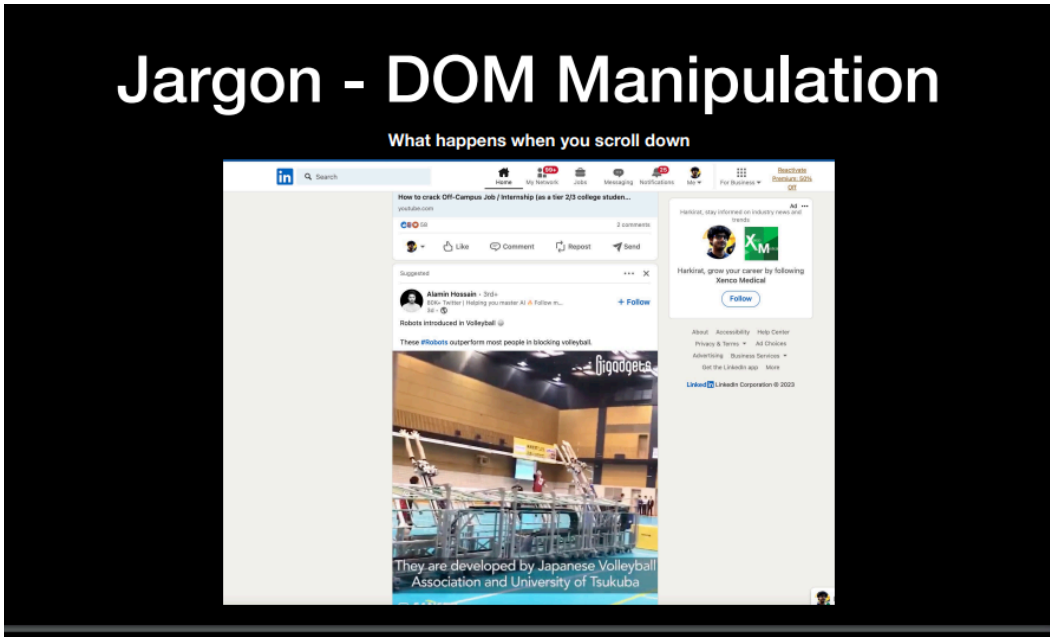
(Reconcilers and Intro to React)

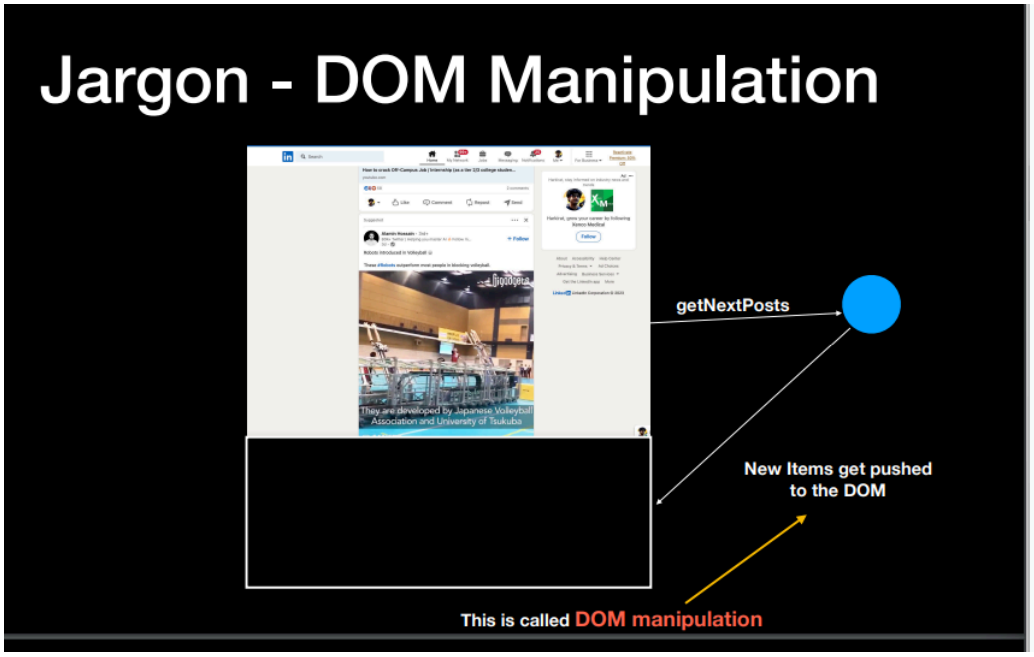
Why frontend frameworks were introduced?

Jargon-DOM Manipulation

LinkedIn is heavily dynamic

What is dynamic lets see an example





Basically the HTML which we got from the server is slowly getting updated.

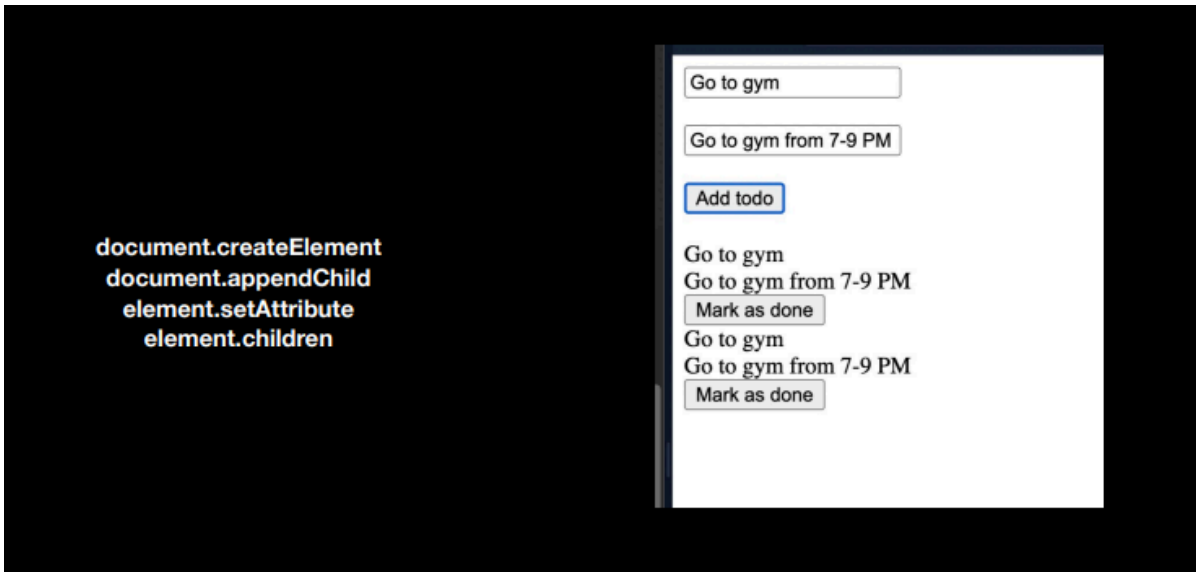
Before React came to picture through DOM manipulation people use to create the dynamic websites.

DOM manipulation is very hard to write as a developer

Making dynamic websites, with the premitive that DOM provides you is very hard.

React make it easier

Let's create a TODO application using DOM manipulation

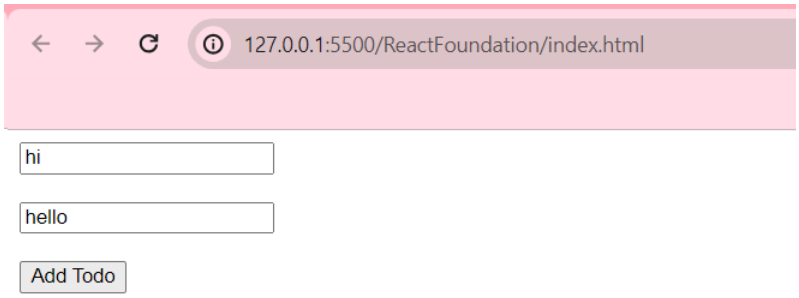


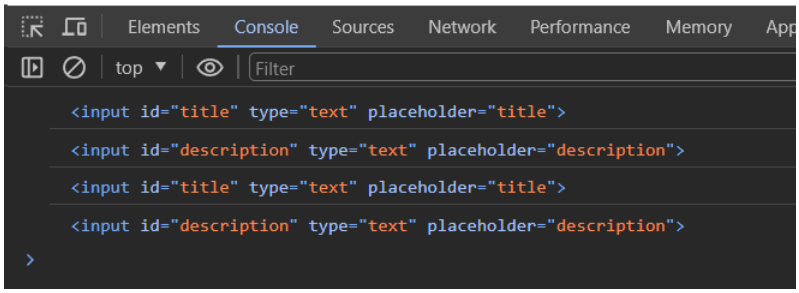
Basic HTML

```
<html>
    <div>
        <input type="text" placeholder="title"><br><br>
        <input type="text" placeholder="description"><br><br>
        <button>Add Todo</button>
    </div>
</html>
```

Adding a function when we click on the button and console.log title and description elements

```
<html>
    <script>
        function addTodo(){
            const title = document.getElementById("title");
            const description = document.getElementById("description");
            console.log(title);
            console.log(description);
        }
    </script>
    <div>
        <input id="title" type="text" placeholder="title"><br><br>
        <input id="description" type="text"
placeholder="description"><br><br>
        <button onclick="addTodo()">Add Todo</button>
    </div>
</html>
```





Basically when we write the todo heading and description and click the button we want the following manner

```
        <div id="container">
            <div>
                <div>Title of the todo1</div>
                <div>Description of the todo1</div>
                <button>Mark as done</button>
            </div>
        </div>
```

One way of doing this could be

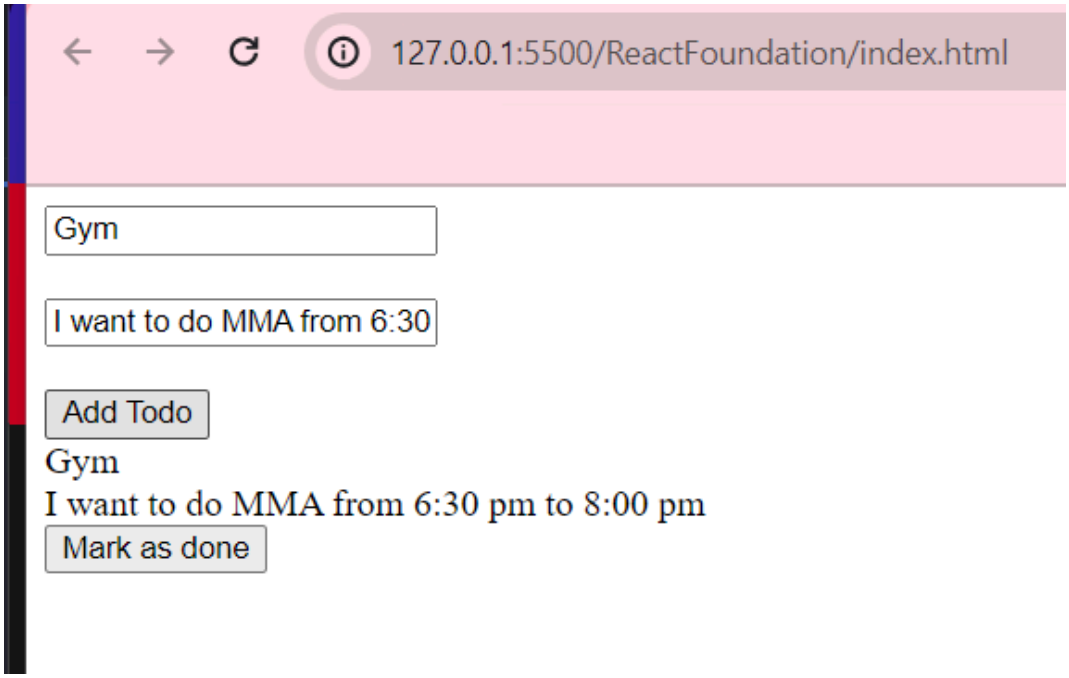
```
function addTodo(){
            const title = document.getElementById("title").value;
            const description =
document.getElementById("description").value;
            document.getElementById("container").innerHTML = `
            <div>
                <div>Title of todo1</div>
                <div>Description of todo1</div>
                <button>Mark as done</button>
```

```
            </div>
            `
        }
```

Ugly approach

```
    <script>
        function addTodo(){
            const title = document.getElementById("title").value;
            const description =
document.getElementById("description").value;
            document.getElementById("container").innerHTML = `
            <div>
                <div>${title}</div>
                <div>${description}</div>
                <button>Mark as done</button>
            </div>
            `
        }
    </script>
```

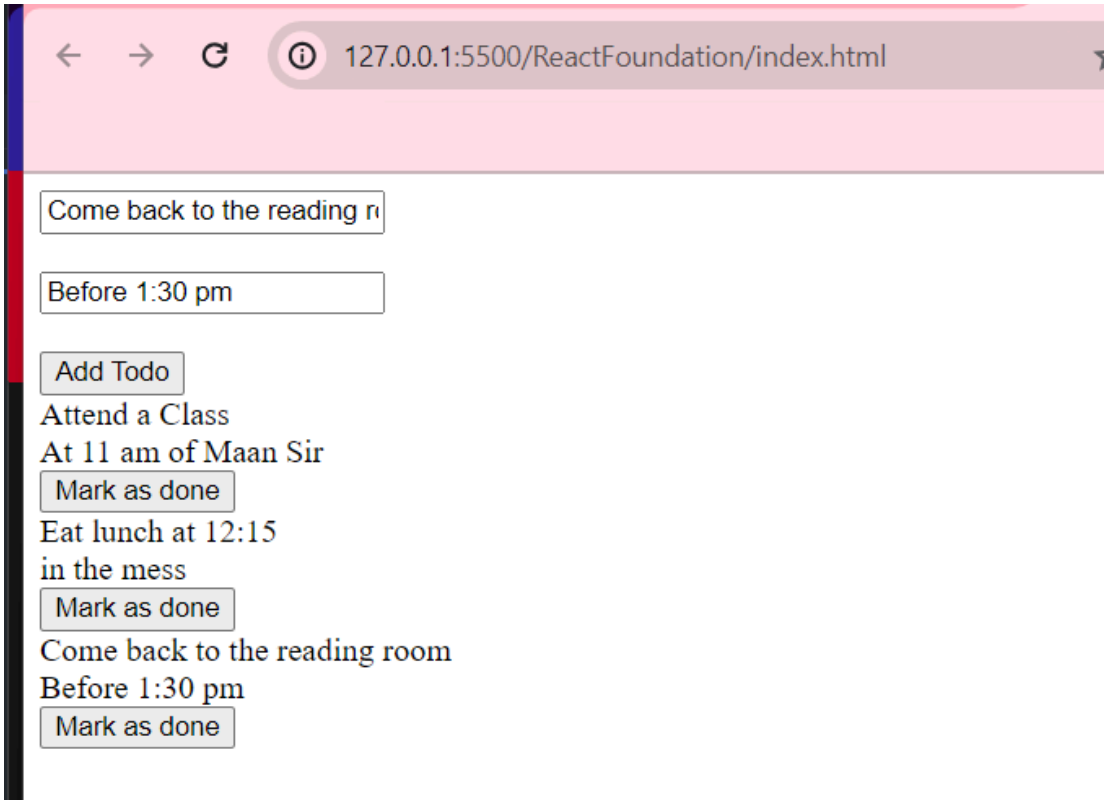
With tilda symbol (``) we can do templating.



But these will be overridden if we write another todo.

Lets correct this

```
    <script>
        function addTodo(){
            const title = document.getElementById("title").value;
            const description =
document.getElementById("description").value;
            const orginalHtml =
document.getElementById("container").innerHTML;
            document.getElementById("container").innerHTML = orginalHtml+`
            <div>
                <div>${title}</div>
                <div>${description}</div>
                <button>Mark as done</button>
            </div>
            `
        }
    </script>
```



Our container will look like

```
<button onclick="addTodo()">Add Todo</button>
▼<div id="container">
  ▼<div>
      <div>Attend a Class</div>
      <div>At 11 am of Maan Sir</div>
      <button>Mark as done</button>
    </div>
  ▼<div>
      <div>Eat lunch at 12:15</div>
      <div>in the mess</div>
      <button>Mark as done</button>
    </div>
  ▼<div>
      <div>Come back to the reading room</div>
      <div>Before 1:30 pm</div>
      <button>Mark as done</button>
    </div>
  </div>
</div>
```

Another approach

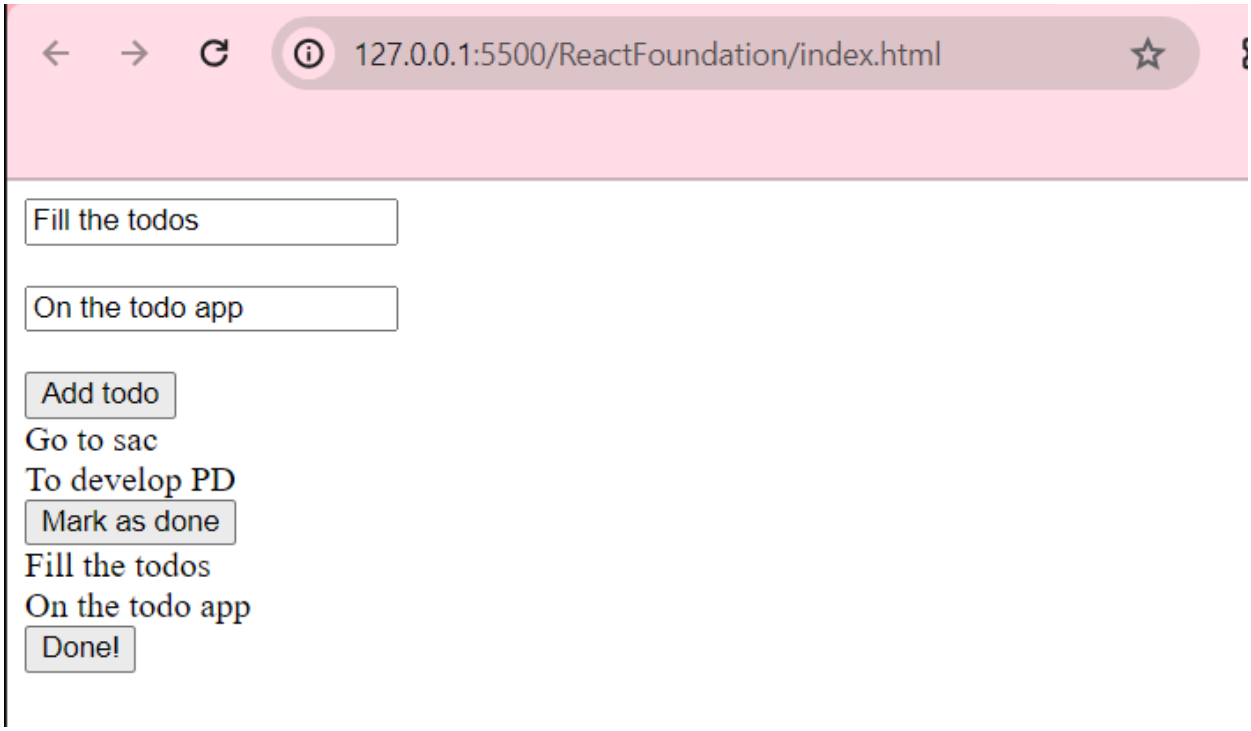
```
  // creating outer div
            const outerDiv = document.createElement("div")
            // creates a DOM div element which is currently not present in
the DOM
            // we can verify it in chrome developer tools.
            // createElement() goal is to first create an element in
memory and then eventually append it in to the DOM
```

As in the previous approach , here with the createElement() we want to create a parent div with three child : two div and a button.

FInal code:

```
<!DOCTYPE html>
<html>

<head>
  <script>
    let globalId = 1;
    //   random id so that we know which button to target

    function markAsDone(todoId) {
      const parent = document.getElementById(todoId);
      parent.children[2].innerHTML = "Done!"
    //   thirdChild of a todo
    }

    function createChild(title, description, id) {
    // Here we are creating div with three children
    // intialize empty div object
      const child = document.createElement("div");
    // if we log it we see empty div
    // <div></div>
      const firstGrandChild = document.createElement("div");
      firstGrandChild.innerHTML = title;
      const secondGrandChild = document.createElement("div");
      secondGrandChild.innerHTML = description;
      const thirdGrandChild = document.createElement("button");
```

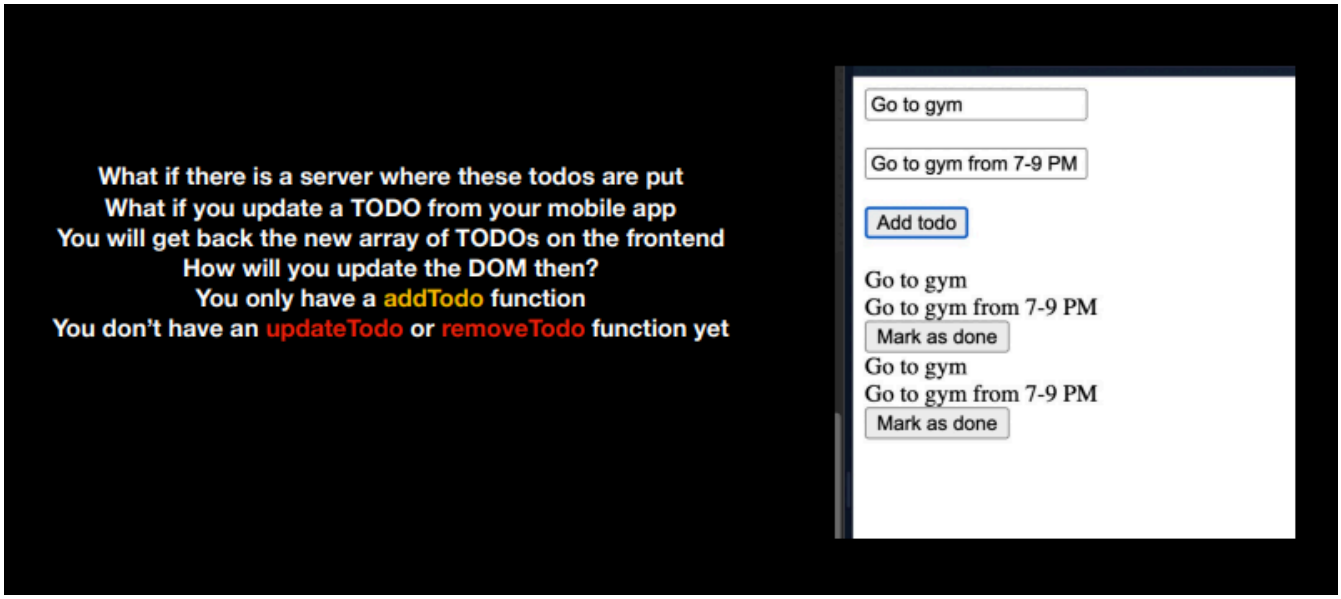
```
      thirdGrandChild.innerHTML = "Mark as done";
      thirdGrandChild.setAttribute("onclick", `markAsDone(${id})`);
    //   append to the child Which is the orginal child of the container
      child.appendChild(firstGrandChild);
      child.appendChild(secondGrandChild);
      child.appendChild(thirdGrandChild)
      child.setAttribute("id", id);
      return child;
    }
    function addTodo() {
      const title = document.getElementById("title").value;
      const description = document.getElementById("description").value;
      const parent = document.getElementById("container");

    // globalId++ , giving id to each container (todo)
      parent.appendChild(createChild(title, description, globalId++));
    // createChild create structure of our div
    }
  </script>
</head>

<body>
  <input type="text" id="title" placeholder="Todo title"></input> <br
/><br />
  <input type="text" id="description" placeholder="Todo
description"></input> <br /><br />
  <button onclick="addTodo()">Add todo</button>

  <div id="container">
  </div>
</body>
</html>
```



Problem with this approach

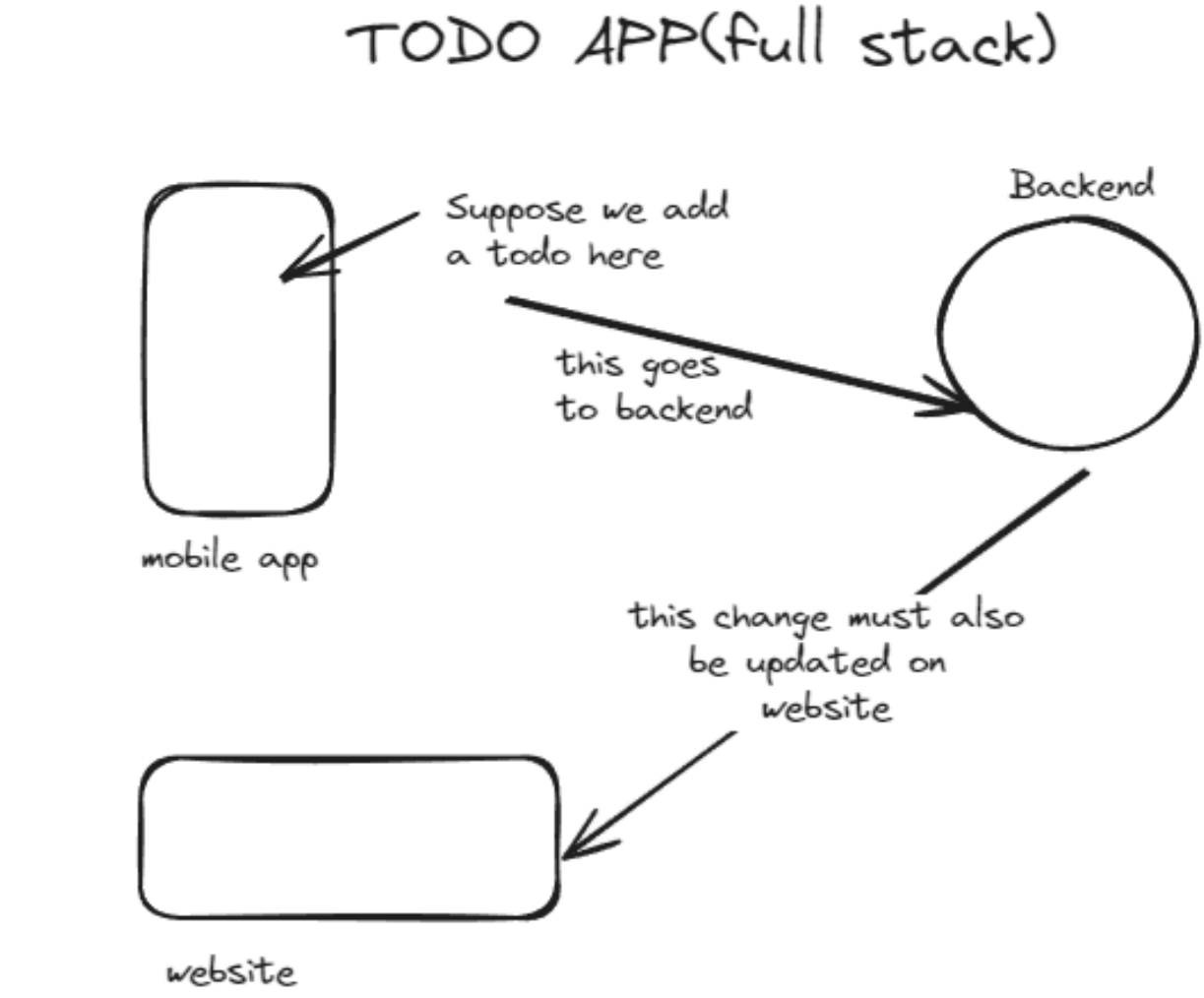
Very hard to add and remove elements.

No central state.



Todo app like have usually multiple clients

Lets say we have full stack todo app



Change must propagate(Maybe something get updated, deleted,added)

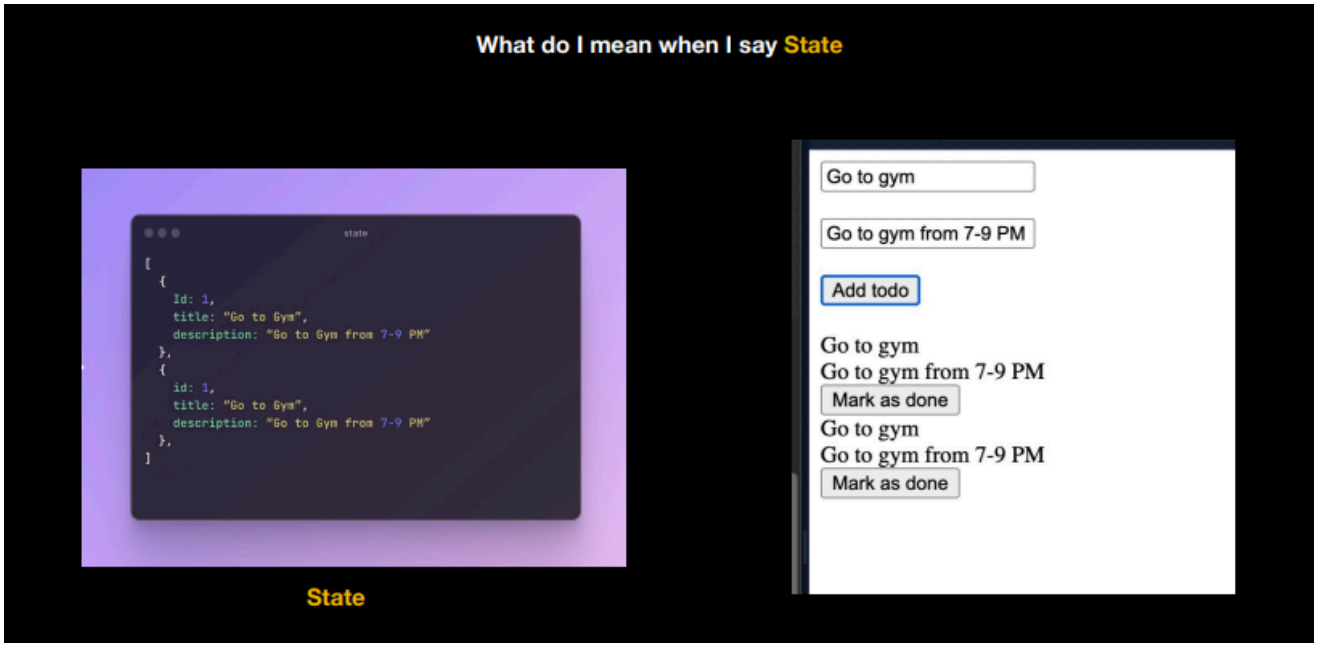
Backend will send the updated(new) list (to website)
It wont tell what got updated/ deleted etc.

This time we are given the todo array by suppose(backend) and we have to support this functionality

Our new problem statement is that we will be given an array of todos from backend we have to render it and display it. We will continuously get these array of todos.

Here we call out input as state.

And of the function is called again with new state then they have to remove the already append todos and fill with the new one.

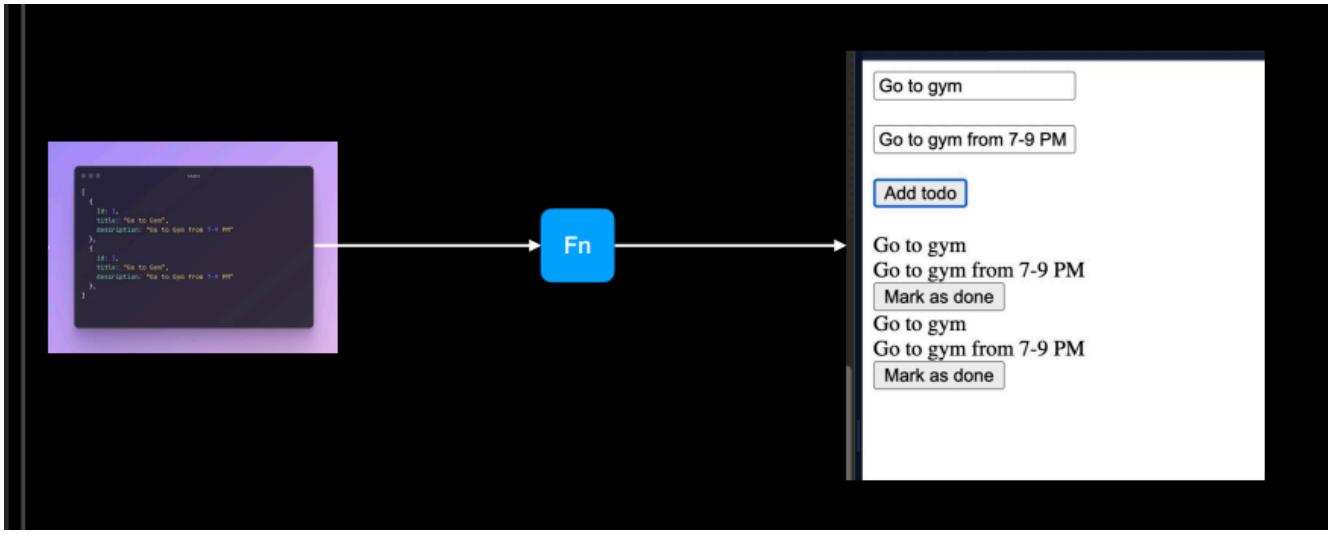


This approach of taking **state** as input and creates the output, is more applicable.

In react we have what called the global state.

Array

List of string.



Q/Na

- **State is variable which has how our application should look**

For example: LinkedIn is an array of post. User notification count .

We can convert a LinkedIn page into a huge LinkedIn object.

- This is how website like LinkedIn were created before react.

Jquery and backbone.js were used to help better.

- React also under the hood does DOM manipulation

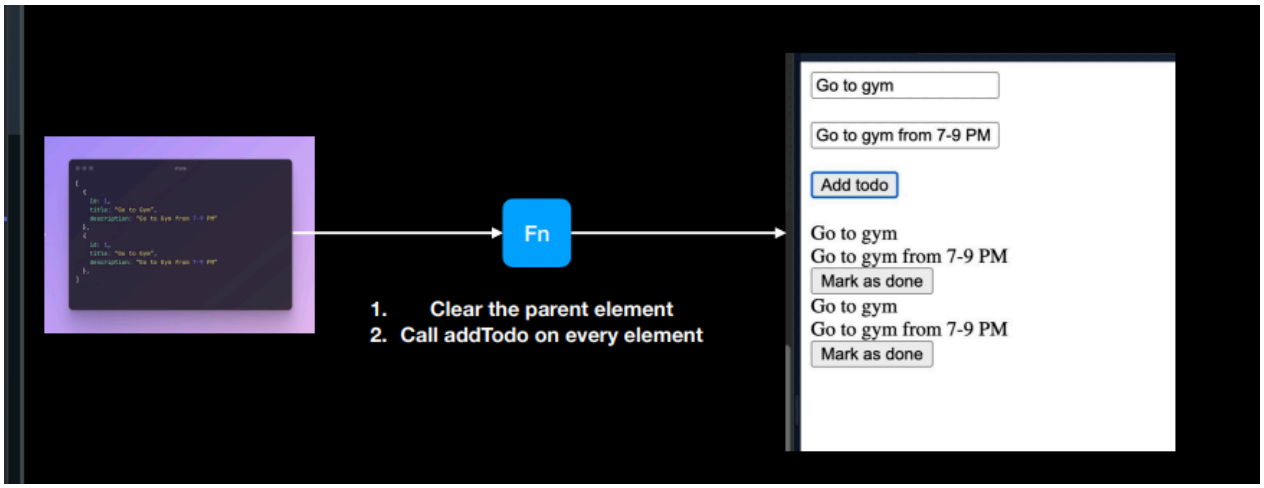
Browser were build in such a way that if we want to make our website dynamic there are five primitive

1. appendChild()
2. createElement()
3. removeChild() etc

Problem Statement

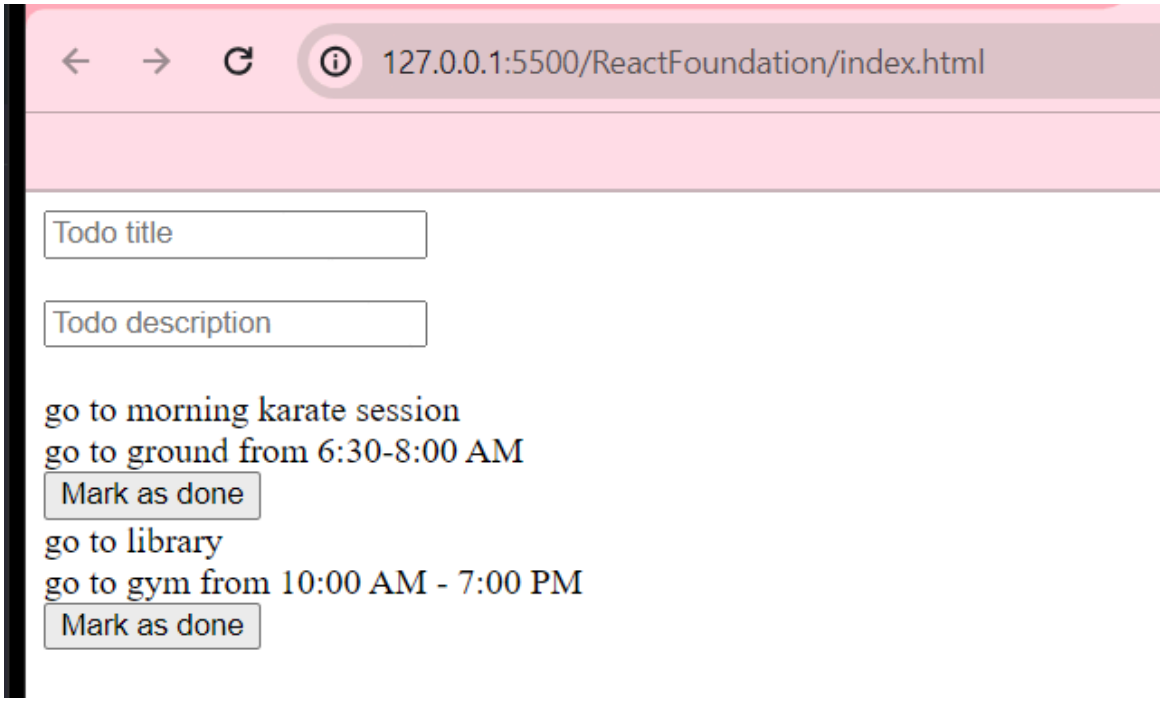
Our new problem statement is that we will be given an array of todos from backend we have to render it and display it. We will continuously get these array of todos.

Approach - 1




```
<!DOCTYPE html>
<html>
<body>
  <input type="text" id="title" placeholder="Todo title"></input> <br
/><br />
  <input type="text" id="description" placeholder="Todo
description"></input> <br /><br />
  <div id="container">
  </div>

  <!-- the problem with script being on the top is that the function is
being called even when the container is not defined -->
  <script>
    function createChild(title, description, id) {
      const child = document.createElement("div");
      const firstGrandChild = document.createElement("div");
      firstGrandChild.innerHTML = title;
      const secondGrandChild = document.createElement("div");
      secondGrandChild.innerHTML = description;
      const thirdGrandChild = document.createElement("button");
      thirdGrandChild.innerHTML = "Mark as done";
      thirdGrandChild.setAttribute("onclick", `markAsDone(${id})`);
      child.appendChild(firstGrandChild);
      child.appendChild(secondGrandChild);
      child.appendChild(thirdGrandChild)
```

```
      child.setAttribute("id", id);
      return child;
    }
    // state constraints
    // every element of state would have a title, description and id
    function updateDOMAccToState(state){
        const parent = document.getElementById("container");
        parent.innerHTML = "";
        for( let i=0; i<state.length; i++){
            const child = createChild(state[i].title,
state[i].description,state[i].id)
            parent.appendChild(child);
        }
    }
    // Question may arise that if we have these data from upfront then why
dont we use backtiks and declare div
    // in real world project we wont have access to these data from
upfront and will come from the server.
    // Our browser need to refresh after sometime to check whether we got
any latest data.

    updateDOMAccToState([{
        title: "go to morning karate session",
        description:"go to ground from 6:30-8:00 AM",
        id:1
    },{
        title: "go to library",
        description:"go to gym from 10:00 AM - 7:00 PM",
        id:2
    }]);
  </script>
</body>
</html>
```

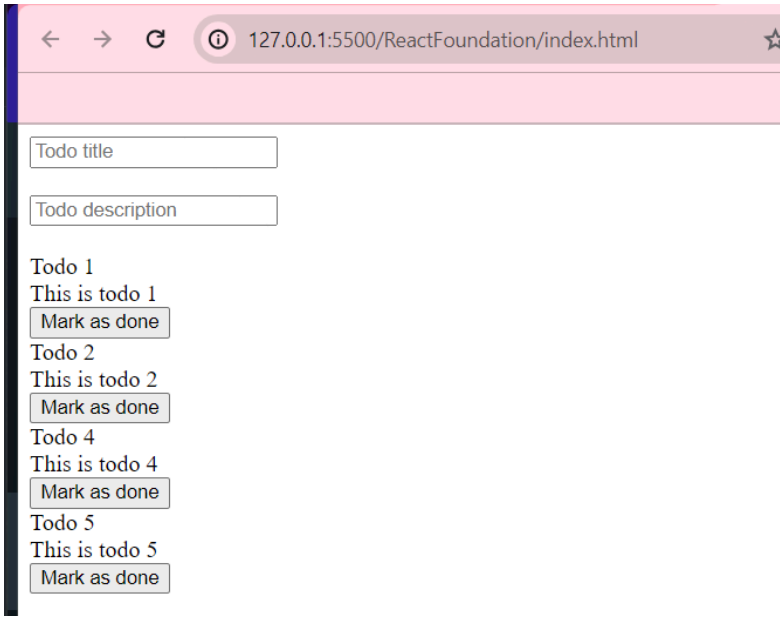


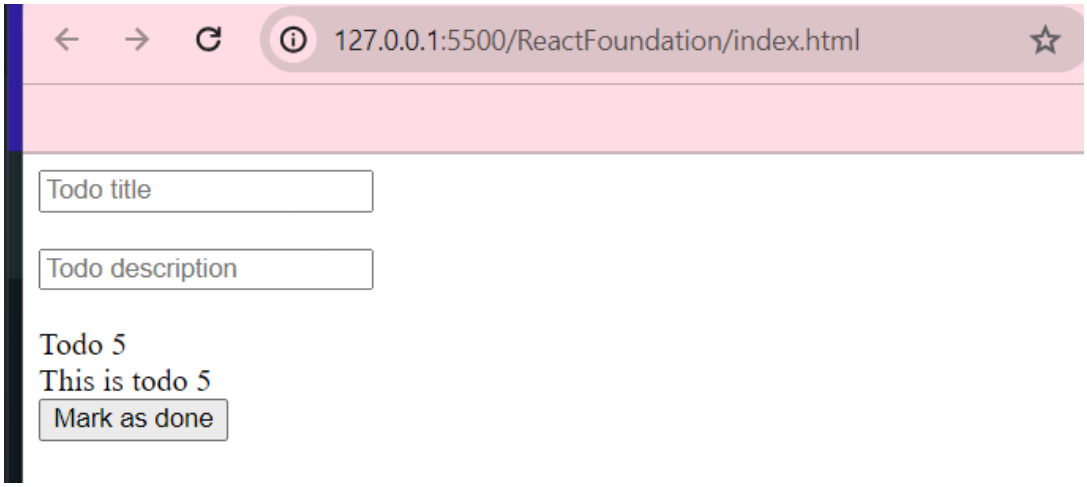
Now in next approach we will get our state from the backend server.

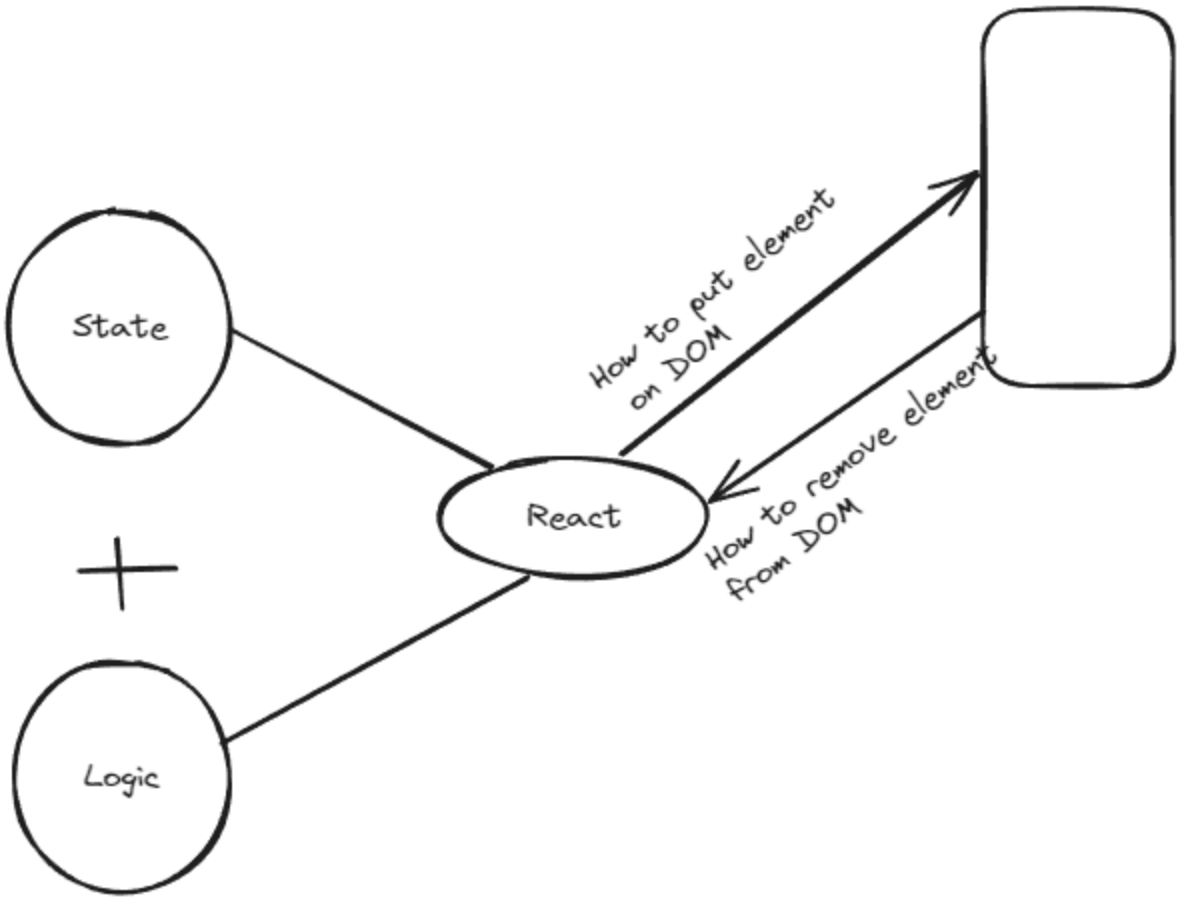
```
<!DOCTYPE html>
<html>
<body>
  <input type="text" id="title" placeholder="Todo title"></input> <br
/><br />
  <input type="text" id="description" placeholder="Todo
description"></input> <br /><br />
  <div id="container">
  </div>

  <!-- the problem with script being on the top is that the function is
being called even when the container is not defined -->
  <script>
    function createChild(title, description, id) {
      const child = document.createElement("div");
      const firstGrandChild = document.createElement("div");
      firstGrandChild.innerHTML = title;
      const secondGrandChild = document.createElement("div");
      secondGrandChild.innerHTML = description;
      const thirdGrandChild = document.createElement("button");
      thirdGrandChild.innerHTML = "Mark as done";
      thirdGrandChild.setAttribute("onclick", `markAsDone(${id})`);
```

```
      child.appendChild(firstGrandChild);
      child.appendChild(secondGrandChild);
      child.appendChild(thirdGrandChild)
      child.setAttribute("id", id);
      return child;
    }
    // state constraints
    // every element of state would have a title, description and id
    function updateDOMAccToState(state){
        const parent = document.getElementById("container");
        parent.innerHTML = "";
        for( let i=0; i<state.length; i++){
            const child = createChild(state[i].title,
state[i].description,state[i].id)
            parent.appendChild(child);
        }
    }
    window.setInterval(async function(){
        // this backend server send random value of state at each refresh
        const res = await fetch("https://sum-server.100xdevs.com/todos")
        const json = await res.json();
        updateDOMAccToState(json.todos);
        // React will ask for state
    },2000)
    //  If all this create child function is done in the react we have to
majorly only handle the window.setInterval()
    // We dont have to write the logic of creating grandchild etc.
  </script>
</body>
</html>
```

This is not good approach since we are changing the whole todos , without checking which todos were updated.


127.0.0.1:5500/ReactFoundation/index.html
Todo title
Todo description
Todo 1
This is todo 1
Mark as done
Todo 2
This is todo 2
Mark as done
Todo 4
This is todo 4
Mark as done
Todo 5
This is todo 5
Mark as done


127.0.0.1:5500/ReactFoundation/index.html
Todo title
Todo description
Todo 5
This is todo 5
Mark as done



Creating something like React is like creating function which take current state as input and update the DOM.

Lets see some flaws with this approach:
Clearing DOM and then adding new thing to the DOM , We can see some flash like thing.

Better Solution

Don’t clear the DOM upfront, update it based on what has changed.
Question is, how does it calculate what all has changed?
Has a todo been marked as complete?

Has a todo been removed from the backend?

By remembering the old todos in a variable (Virtual DOM)

We calculate the difference between Virtual DOM and state.

If suppose first element change then we will update it.

Not optimal(previous approach) what if backend send the same data?

React does the same thing.

Starter Code:

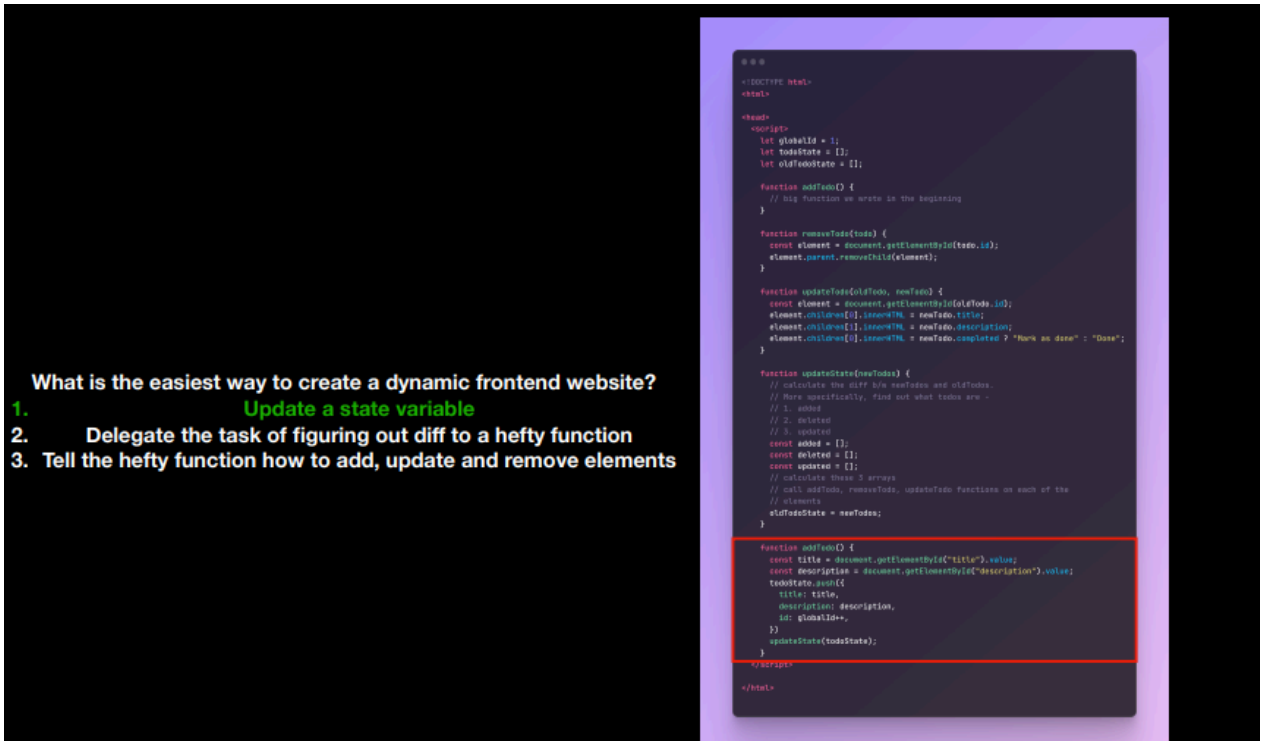
```
<!DOCTYPE html>
<html>

<head>
  <script>
    let globalId = 1;
    let todoState = [];
    let oldTodoState = [];
//Keep track of the old todo
    function addTodo() {
      // big function we wrote in the beginning
    }

    function removeTodo(todo) {
      const element = document.getElementById(todo.id);
      element.parent.removeChild(element);
    }

    function updateTodo(oldTodo, newTodo) {
      const element = document.getElementById(oldTodo.id);
      element.children[0].innerHTML = newTodo.title;
      element.children[1].innerHTML = newTodo.description;
      element.children[0].innerHTML = newTodo.completed ? "Mark as done" :
"Done";
    }
```

```
    function updateState(newTodos) {
      // calculate the diff b/w newTodos and oldTodos.
      // More specifically, find out what todos are -
      // 1. added
      // 2. deleted
      // 3. updated
      const added = [];
      const deleted = [];
      const updated = [];
      // calculate these 3 arrays
      // call addTodo, removeTodo, updateTodo functions on each of the
      // elements
      oldTodoState = newTodos;
    }

    function addTodo() {
      const title = document.getElementById("title").value;
      const description = document.getElementById("description").value;
      todoState.push({
        title: title,
        description: description,
        id: globalId++,
      })
      updateState(todoState);

    }
  </script>
</head>

<body>
  <input type="text" id="title" placeholder="Todo title"></input> <br
/><br />
  <input type="text" id="description" placeholder="Todo
description"></input> <br /><br />
  <button onclick="addTodo()">Add todo</button>
  <br /> <br />

  <div id="todos">

  </div>
```

```
</body>

</html>
```

What is the easiest way to create a dynamic frontend website??

1. **Update a state variable.**
2. **Delegate the task of figuring out the difference to a hefty function.**
3. **Tell the hefty function how to add, update and remove elements.**



What is the easiest way to create a dynamic frontend website?

1. Update a state variable
2. Delegate the task of figuring out diff to a hefty function
3. Tell the hefty function how to add, update and remove elements

```
<!DOCTYPE html>
<html>

<head>
  <script>
    let globalId = 1;
    let todoState = [];
    let oldTodoState = [];

    function addTodo() {
      // big function we wrote in the beginning
    }

    function removeTodo(todo) {
      const element = document.getElementById(todo.id);
      element.parent.removeChild(element);
    }

    function updateTodo(oldTodo, newTodo) {
      const element = document.getElementById(oldTodo.id);
      element.children[0].innerHTML = newTodo.title;
      element.children[1].innerHTML = newTodo.description;
      element.children[2].innerHTML = newTodo.completed ? "Mark as done" : "Done";
    }

    function updateState(newTodos) {
      // calculate the diff b/w newTodos and oldTodos.
      // More specifically, find out what todos are -
      // 1. added
      // 2. deleted
      // 3. updated
      const added = [];
      const deleted = [];
      const updated = [];
      // calculate these 3 arrays
      // call addTodo, removeTodo, updateTodo functions on each of the
      // elements
      oldTodoState = newTodos;
    }

    function addTodo() {
      const title = document.getElementById("title").value;
      const description = document.getElementById("description").value;
      todoState.push({
        title: title,
        description: description,
        id: globalId++,
      })
      updateState(todoState);
    }
  </script>

</html>
```

What is the easiest way to create a dynamic frontend website?

1. Update a state variable
2. Delegate the task of figuring out diff to a hefty function
3. Tell the hefty function how to add, update and remove elements

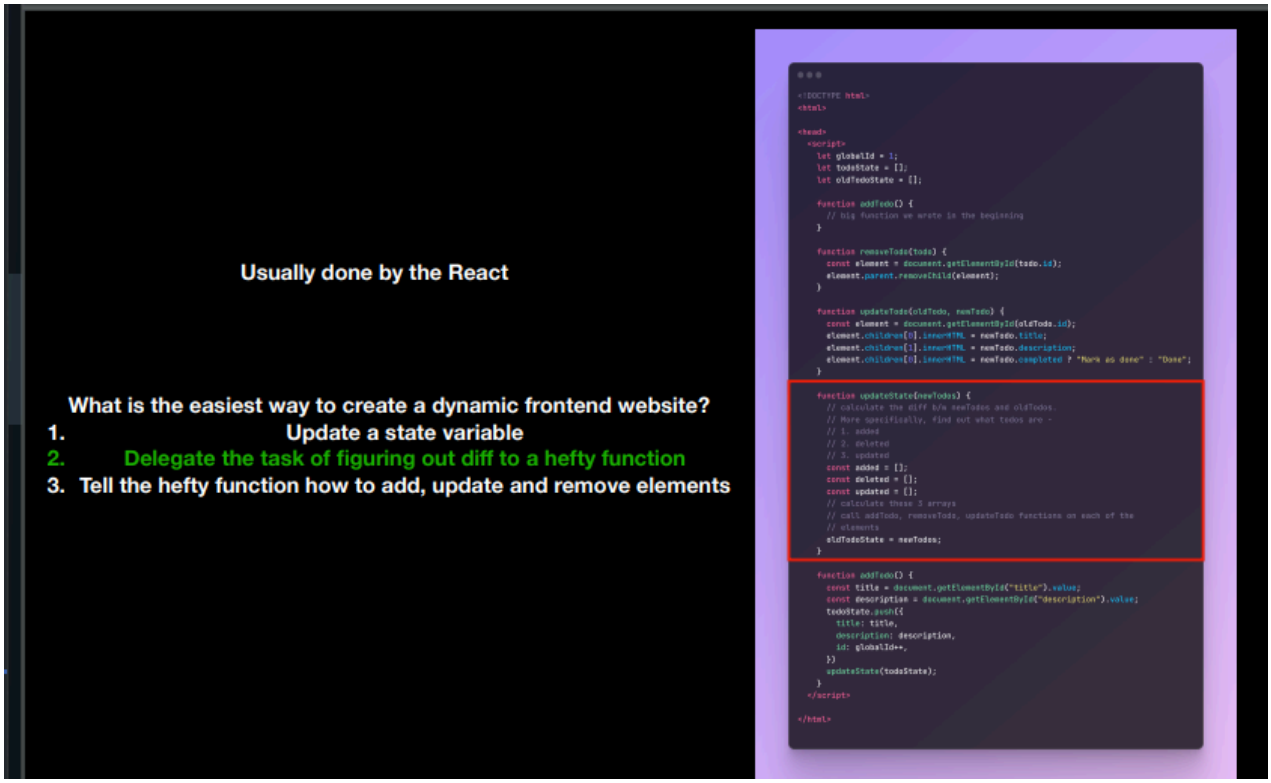
```
<!DOCTYPE html>
<html>

<head>
  <script>
    let globalId = 1;
    let todoState = [];
    let oldTodoState = [];

    function addTodo() {
      // big function we wrote in the beginning
    }

    function removeTodo(todo) {
      const element = document.getElementById(todo.id);
      element.parent.removeChild(element);
    }

    function updateTodo(oldTodo, newTodo) {
      const element = document.getElementById(oldTodo.id);
      element.children[0].innerHTML = newTodo.title;
      element.children[1].innerHTML = newTodo.description;
      element.children[2].innerHTML = newTodo.completed ? "Mark as done" : "Done";
    }

    function updateState(newTodos) {
      // calculate the diff b/w newTodos and oldTodos.
      // More specifically, find out what todos are -
      // 1. added
      // 2. deleted
      // 3. updated
      const added = [];
      const deleted = [];
      const updated = [];
      // calculate these 3 arrays
      // call addTodo, removeTodo, updateTodo functions on each of the
      // elements
      oldTodoState = newTodos;
    }

    function addTodo() {
      const title = document.getElementById("title").value;
      const description = document.getElementById("description").value;
      todoState.push({
        title: title,
        description: description,
        id: globalId++,
      })
      updateState(todoState);
    }
  </script>

</html>
```

What is done by React??

React just calculates the difference between old and new state.

ReactDOM (tell react how to remove or put things in DOM) similarly

ReactNative tell in how to put and remove thing in native applications



React can be used to create not only websites but also mobile apps.

Setting up a new React project

C:\Users\NTC\Desktop\Dev\Week4>**npm create vite@latest**

Need to install the following packages:

create-vite@5.2.0

Ok to proceed? (y) **y**

√ Project name: ... **React_Basic**

√ Package name: ... **react-basic**

? Select a framework: » - Use arrow-keys. Return to submit.

> Vanilla

Vue

√ Select a framework: » **React**

√ Select a variant: » **JavaScript**

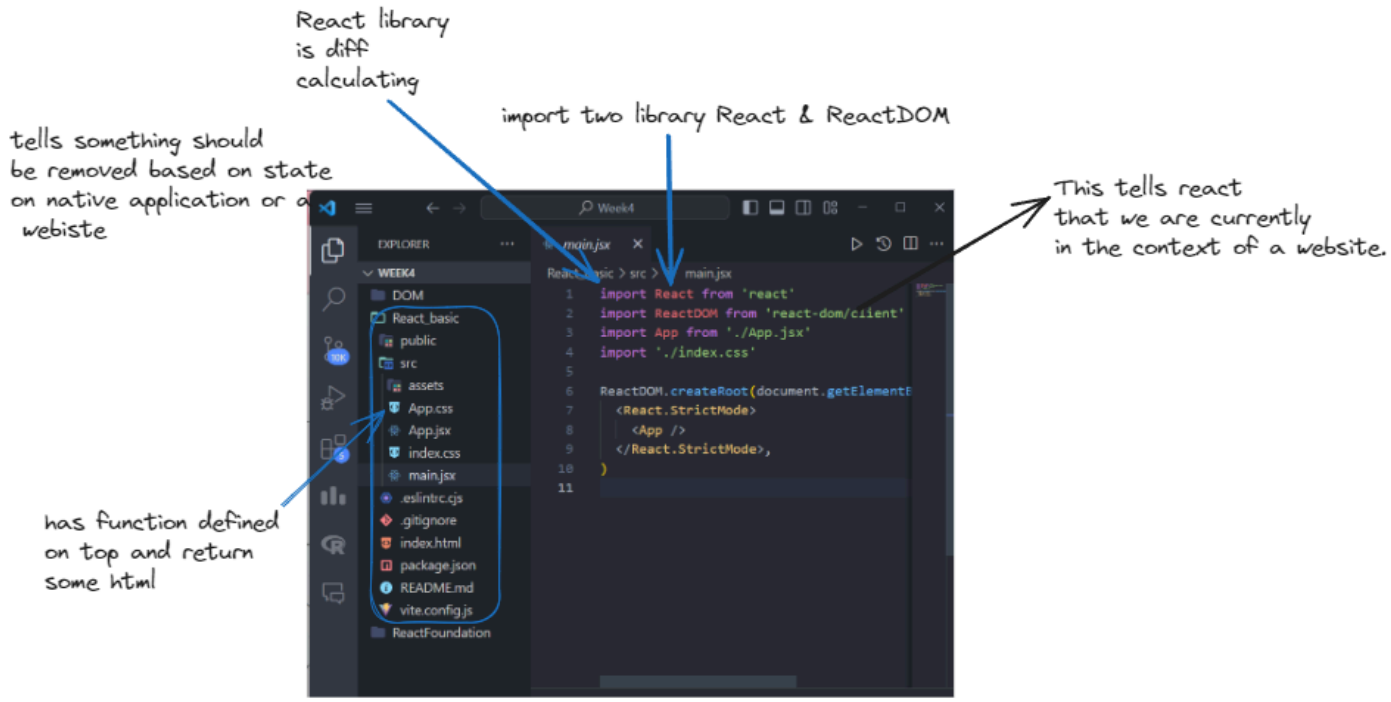
Scaffolding project in C:\Users\NTC\Desktop\Dev\Week4\React_Basic...

Done. Now run:

cd React_Basic

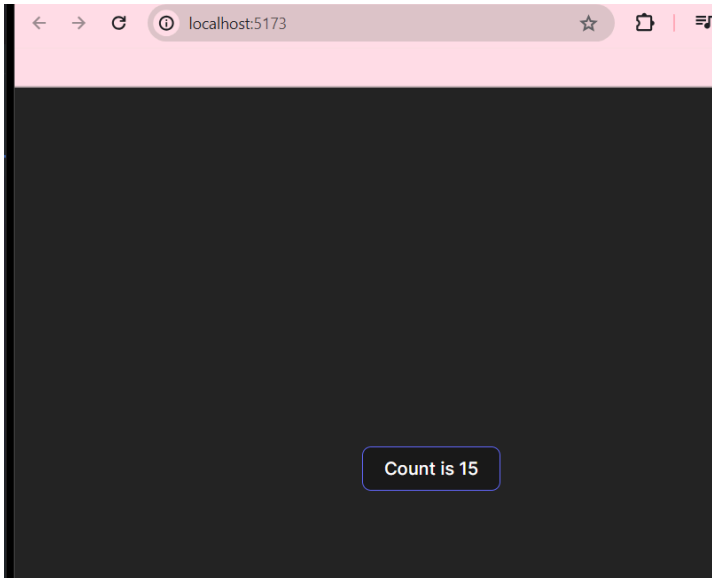
npm install

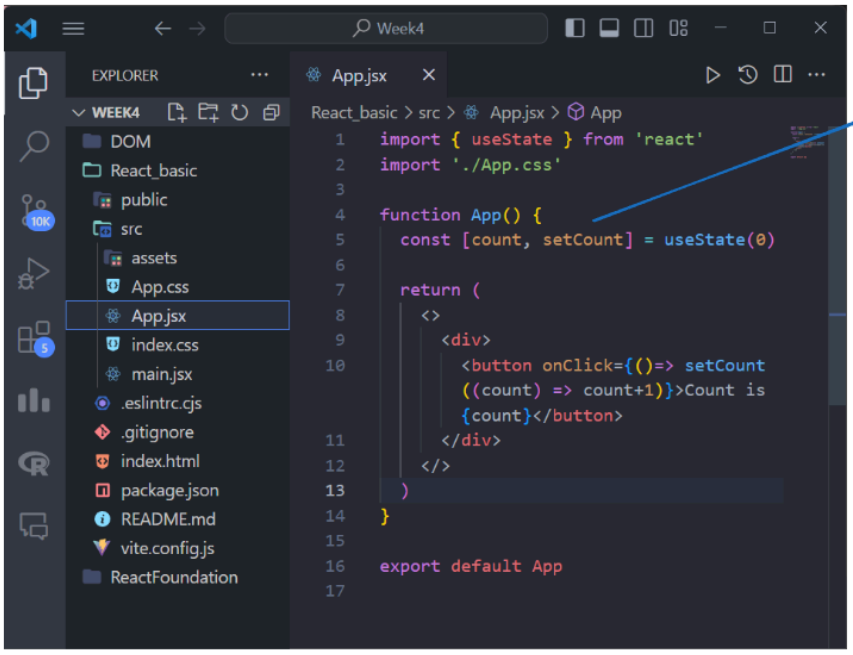
npm run dev



Steps to run the program

- **Go to the project directory**
- **npm install**
- **npm run dev**





A function returning html

We havent return div yet VS code is not complaining Since it is jsx file. jsx file combo of javascript and html

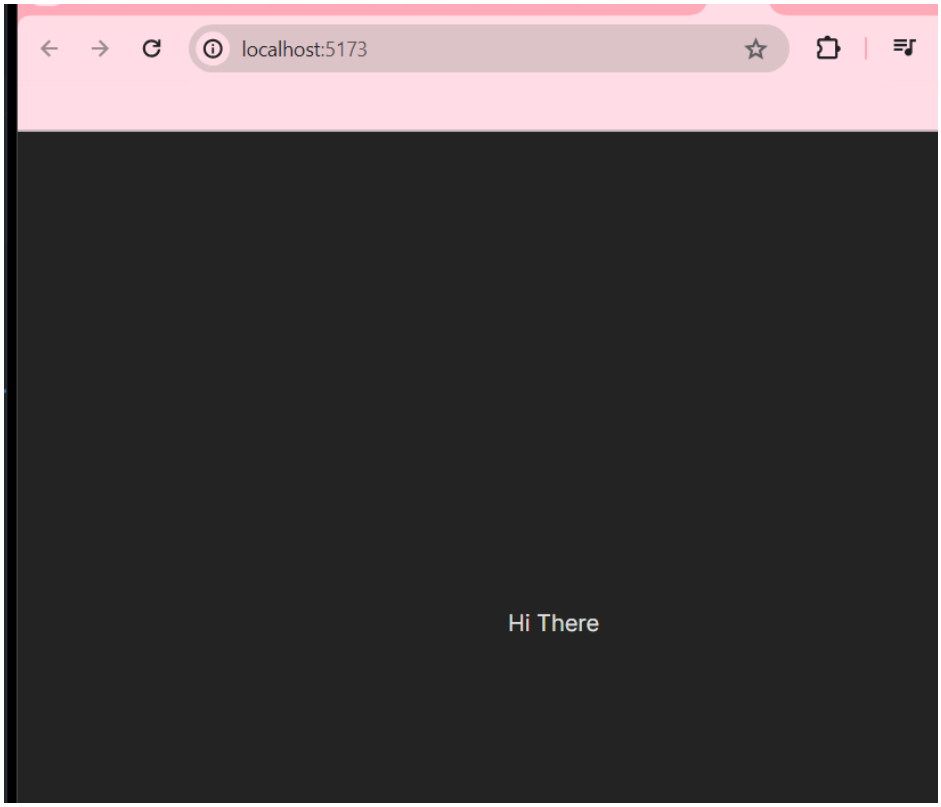
How does this website is rendered

App.jsx

```jsx
import { useState } from 'react'
import './App.css'

function App() {
  return (
    <>
      <div>
```

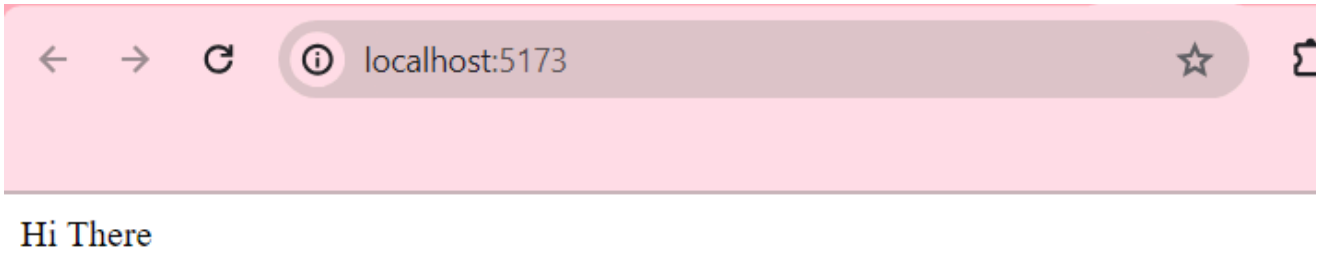
```
        Hi There
      </div>
    </>
  )
}

export default App
```

Our website will look like this:



It is due to App.css and index.css

Now let's empty both of these files.

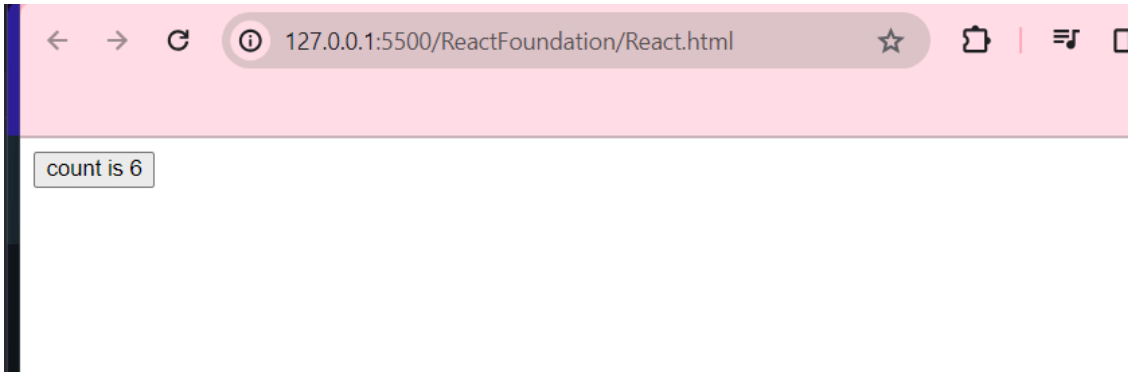


Why is it being returned from function?
Why not part of HTML?

Till now we have discussed dynamic websites.
When we create dynamic websites , we write lot of Js code which does the DOM manipulation
When we create dynamic websites most of the HTML are injected by Javascript only.

Technically in DOM manipulation we are also returning the html , if we see it carefully.
We can also use the previous syntax

```
function App() {
  React.createElement("div")
```



Making this using raw js and html.

```
<html>
    <script>
        let count = 0;
        function updateDomValue(){
            count = count+1;
            document.getElementById("btn").innerHTML = 'count is '+count;
        }
    </script>
    <div>
        <button id="btn" onclick="updateDomValue()">
          count is 0
        </button>
    </div>
</html>
```

We can think count as state variable

Another example

```
import { useState } from 'react'
import './App.css'

function App() {
  const [count,setCount] = useState(0)
  return (
    <div>
      <button onClick={function(){
        // calling global function with new state variable
        // React take care of re-rendering it
        setCount(Math.random())
```

```
      }}>
        count is {count}
      </button>
    </div>
  )
}
export default App
```

Q/Na

- **To propagate the changes in react someone has created dev server which uses express (**maybe**)**
- **Go to 1hour:52 minute marks how react doesn't need server when we did npm run build, How to run it without server.**
- **Reconciller:** given the input, how the DOM look like?
- **React expect the function signature instead of calling it.**
- **Way to create React-app without vite**

Mkdir react-app

cd react-app

npm init

npm init -y

npm install react react-dom

- **useEffect :** to connect this to a backend
- Redux and recoil
- Const a = [1,2]

Reference to the first element is constant.

- const obj = {name: “Thapa”}

Obj.name = “Thapu”