Week 5

Class 5.1
Learnings-

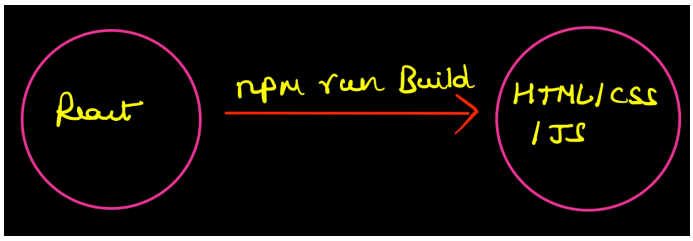
1.Jsx
2.class vs className
3.static vs dynamic websites
4.state
5.components
6.re-rendering

WHY DO YOU NEED REACT ?

BASICALLY, FOR STATIC WEBSITE YOU DON'T NEED REACT .(SHOULD WE REACT ? SURE IT WILL MAKE YOUR LIFE EASIER)

FOR DYNAMIC WEBSITES, THE LIBRARIES OF REACT MAKE IT EASIER TO DO DOM MANIPULATION .

React is just an easier way to write normal HTML/CSS/JS. Its a new syntax , that under the hood gets converted to HTML/CSS/JS.



Under the Hood Browser only understand HTML/CSS/JS . React is just a syntax which under the hood gets converted into HTML/CSS/JS.

Why React ?
1.People Realised its harder to do Dom manipulation the conventional way .
2. There were libraries that came into the picture that made it slightly easy ,but still for a very big app its very hard (Jquery).
3.Eventually ,React created a new syntax to do front-ends.
Under the hood, the react complier convert your code to HTML/CSS/JS.

USES OF INNERHTML -

1) You can use it to get the internal HTML content of any HTML element as an HTML string.
2) You can also use it to set or change elements innerHTML content.

To create a react application you need to worry about three things -
1.state
2.components
3.re-rendering

Creators of frontend frameworks realised that all websites can effectively be divided into two parts -

1. State
2. Components

STATE

In React, "state" is a built-in feature that represents the current condition or data of a component. Think of it as the memory of a component where data can be stored and modified over time. The state is an essential part of React components because it allows them to manage their own data and respond to user interactions.

Here's a breakdown of what you need to know about state:

1. **Component-specific data**: Each component in React can have its own state. This means that the data stored in the state is scoped to that particular component and doesn't affect other components.

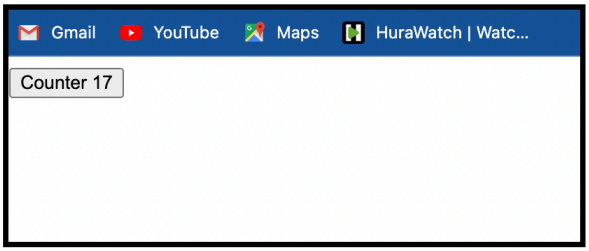
2. **Mutable and private**: State is mutable, meaning you can change its value throughout the lifecycle of a component. Also, it's private to the component, meaning that only the component itself can access and modify its state.

3. **Initialization**: State can be initialized in the constructor of a class component or by using the `useState` hook in a functional component.

4. **Updating state**: To update the state in a class component, you use the `setState()` method provided by React. In functional components, you use the function returned by `useState()` to update the state.

5. **Re-rendering**: Whenever the state of a component changes, React automatically re-renders the component, reflecting the updated state in the user interface.

In simple terms , An object that represents the current state of the app . It represents the dynamic things in your app(things that change)

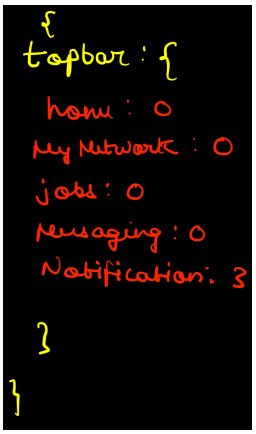


The state of this website will be the simple it would be the object that has count field . And rest all the element would be static in this website .

Example 2 -

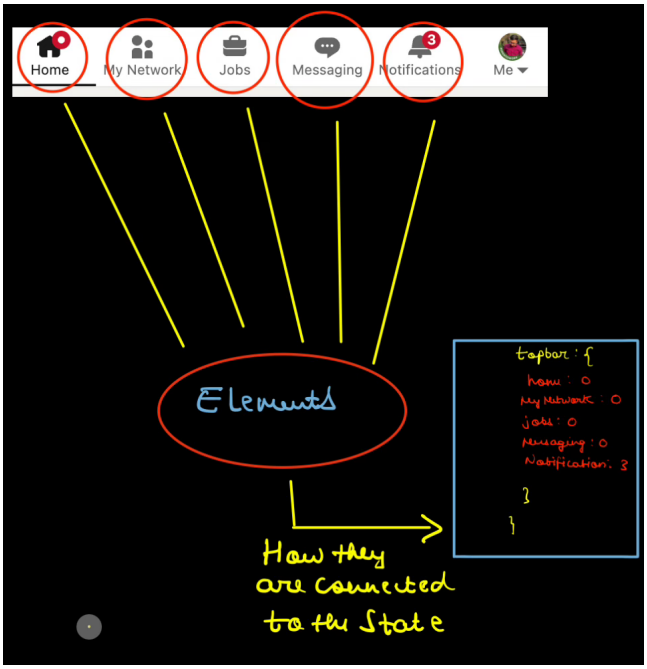
Linked In

State -



In react you maintain the state ,which represent your dynamic bits and then you just have to update the state .and then react takes care of actually updating the DOM elements .

COMPONENTS



This is what React needs -

React just need the elements , and how they are connected to the state .and rest things such as updating them when user interacts with this components . The React will take care of it .

Re-Rendering

Lets say I update the state and I change for example notification to 12 . Then the competent needs to render 12 as well. This thing of me updating the state .

When would the state update ?
Whenever the backend tells website that the number of notification as gone up.(there is some backend request that goes every 10 sec to check current state).

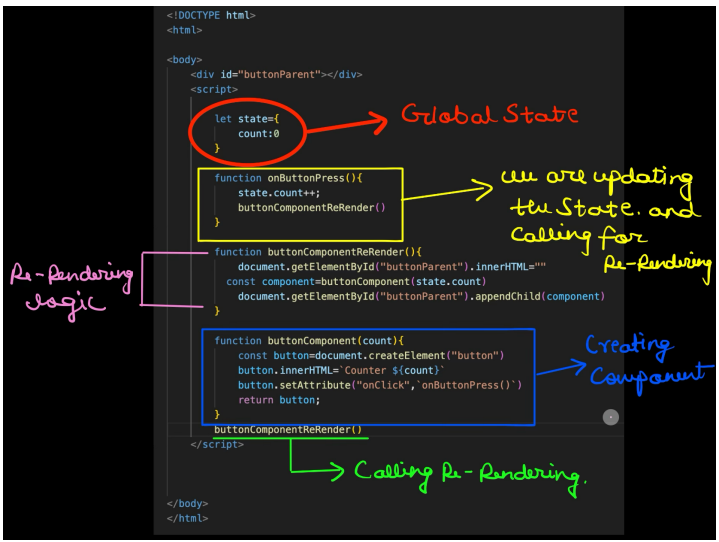
Then developer changes the notification field in the state . And then react re renders the Component or Dom Element .

So at high Level

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

For example lets consider the counter App we created lets create it having a global state ,component and a re re rendering function .

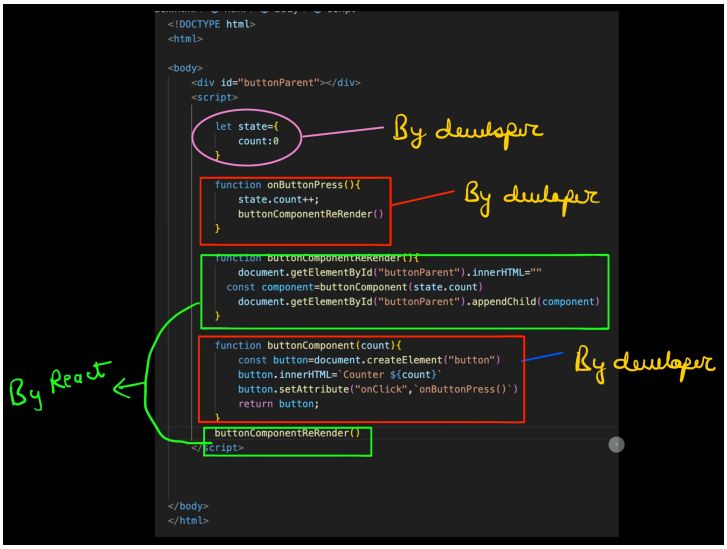
This what we need to do to simply create the counter button without react.



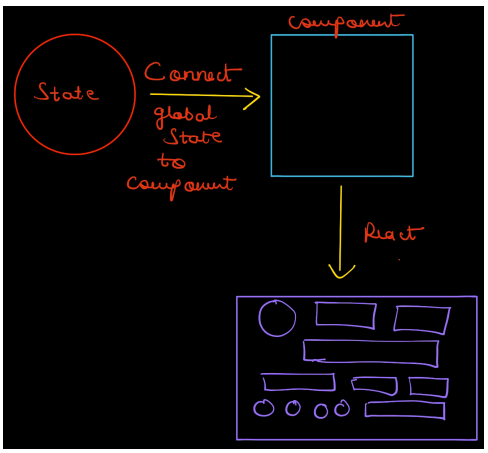
But with react it pretty simple we just have to -

1. Initialise the state object
2. Update the state object.
3. Create a component .(Given current state as input)

All the rendering logic will be taken care by the react .under the hood react converts into this big html/css/js code . But it make it easier for us.



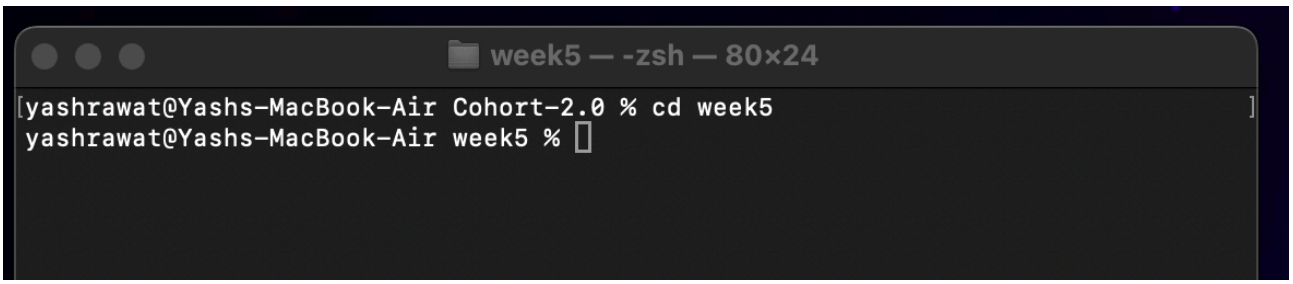
Q) WHAT REACT NEEDS ?



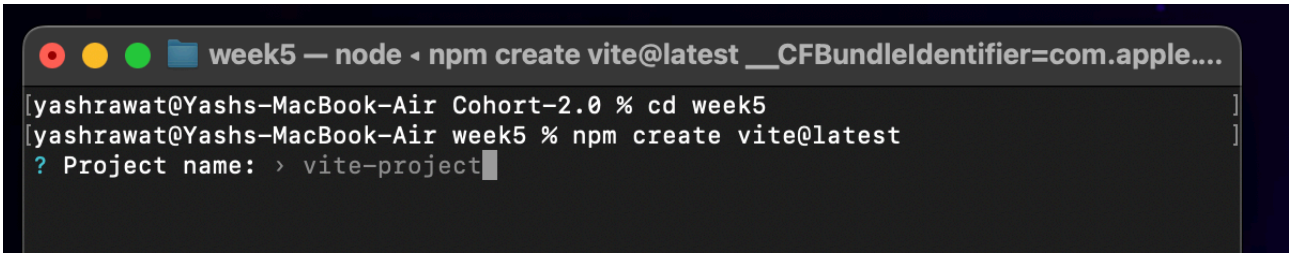
It just needs a state that is connected with the component .and then the react will take care of the rest of the things.

Setting up React Project

Step 1 - open terminal and go to the folder where you save/put your projects .



Step 2 - now run this command in the file npm create vite@latest



Step 3 - give your project the name .

Step 4 - after giving the name select the framework .(which for us is react)

```
week5 — node ◂ npm create vite@latest __CFBundleIdentifier=com.apple....
yashrawat@Yashs-MacBook-Air Cohort-2.0 % cd week5
yashrawat@Yashs-MacBook-Air week5 % npm create vite@latest
✔ Project name: … counter-app
? Select a framework: › - Use arrow-keys. Return to submit.
❯   Vanilla
    Vue
    React
    Preact
    Lit
    Svelte
    Solid
    Qwik
    Others
```

Step 5 - after selecting the framework select variant (which for us is JavaScript)

```
week5 — node ◂ npm create vite@latest __CFBundleIdentifier=com.apple....
yashrawat@Yashs-MacBook-Air Cohort-2.0 % cd week5
yashrawat@Yashs-MacBook-Air week5 % npm create vite@latest
✔ Project name: … counter-app
✔ Select a framework: › React
? Select a variant: › - Use arrow-keys. Return to submit.
    TypeScript
    TypeScript + SWC
❯   JavaScript
    JavaScript + SWC
    Remix ↗
```

Step 6 - it will successfully create the boiler application of react .

```
week5 — -zsh — 80×24
yashrawat@Yashs-MacBook-Air Cohort-2.0 % cd week5
yashrawat@Yashs-MacBook-Air week5 % npm create vite@latest
✔ Project name: … counter-app
✔ Select a framework: › React
✔ Select a variant: › JavaScript

Scaffolding project in /Users/yashrawat/Desktop/Cohort/Cohort-2.0/week5/counter-
app...

Done. Now run:

  cd counter-app
  npm install
  npm run dev

yashrawat@Yashs-MacBook-Air week5 %
```

NOTE - Jsx file - A javaScript file in which you can write xml too.

Step 7 - open it in the VS code and run the following command in the terminal before running it -
1. Npm install
2. Npm run dev

After running the commands this will appear.

```
VITE v5.2.10  ready in 445 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
```

UNDERSTANDING THE STRUCTURE OF REACT FILE-

Index.jsx file -

```
index.html > html > body > div#root
<!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>
  </body>
</html>
```

1.We have div element having the id = root .

2. in the script section we can see the src to the main.jsx file.

Main.jsx file -

```
main.jsx U   index.html U   App.jsx U
src > 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 />,
)
```

As we can see we are calling out a div element with the id = root and rendering it .

App.jsx file -

```
src > App.jsx > App
import './App.css'

function App() {
  return (
    <div>
      hi there
    </div>
  )
}

export default App
```

Now in the app.jsx file we are returning a div.

index.css and App.css files -

```
> assets
# App.css      U
App.jsx        U
# index.css    U
```

Contain all the styling part of css.

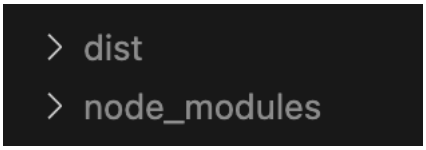
In the end all this react code gets converted into the HTML/CSS/JS files.

Now the Question is How does the react code gets converted into HTML/CSS/JS ?

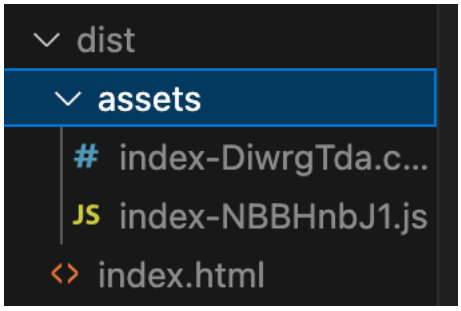
Step 1 - open the terminal .
Step 2 - run the command npm run build .

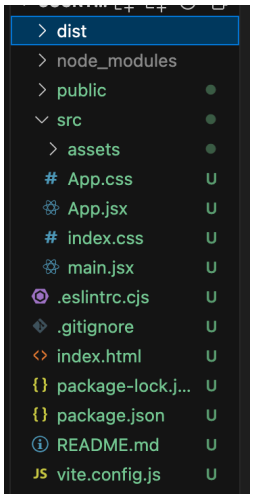
```
yashrawat@Yashs-MacBook-Air counter-app % npm run build

> counter-app@0.0.0 build
> vite build

vite v5.2.10 building for production...
✓ 32 modules transformed.
dist/index.html                  0.46 kB │ gzip:  0.29 kB
dist/assets/index-DiwrgTda.css   1.39 kB │ gzip:  0.72 kB
dist/assets/index-NBBHnbJ1.js  142.54 kB │ gzip: 45.76 kB
✓ built in 578ms
yashrawat@Yashs-MacBook-Air counter-app %
```

Step 3 - a dist and a node_modules folder will be created .



Inside Dist -

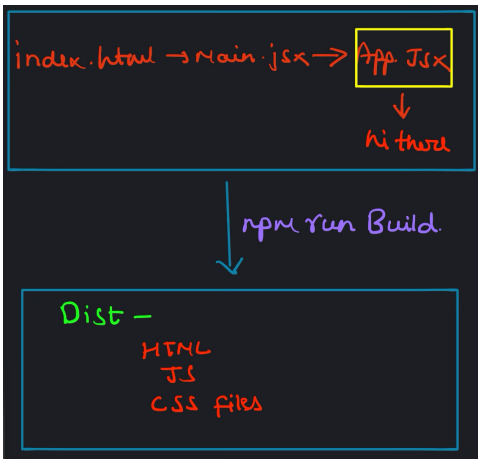




Command npm run build converts all your react code into html, css,js code . As you see above .

Now even if you delete all the file except . Dist file you will still have a fully functional website if you deploy it on aws.

RECAP



Now lets create a dynamic site until now we have only created a static website .

So to create a dynamic website we need two things state , components .

So we now defined the state in the app.jsx file.

```
src > App.jsx > App

let state={
  count:0
}

function App() {
```

Now for component we already have it the react component .

```
function App() {
  return (
    <div>
      hi there
    </div>
  )
}
```

Now inside these component I have to define the button component .

NOTE - whenever you write JavaScript code in the React component you have to enclose it in the curly braces.

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

In react the onClick attribute will be defined like these -

```
<button onClick={onClickHandler}>Counter {state.count}</button>
```

onClickHandler is a function . Which will be called or triggered when the button is clicked .

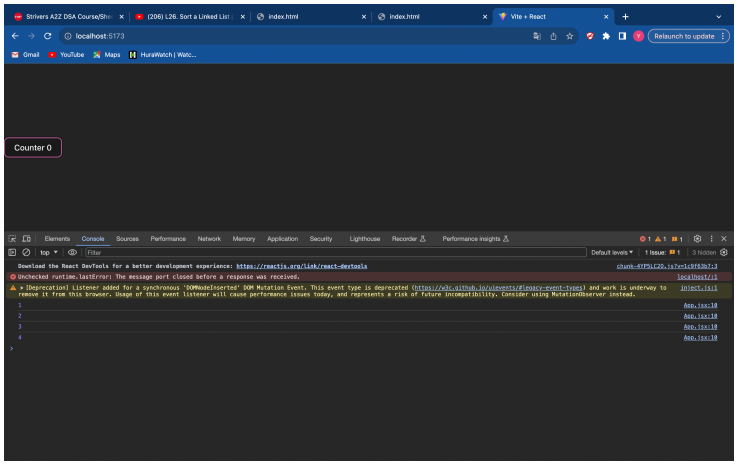
Now you will question why we are writing it like these ?

Because in Xml this is the way onlclick attribute is defined .

Now as we did all the things react needs but after that why counter is not getting re rendered while I am clicking on the button .

```
// we first initialised the state .
let state={
  count:0
}

function App() {
// updating the state .
  function onClickHandler(){
    state.count=state.count + 1;
    console.log(state.count)
  }
  return (
    // defined the component .
    <div>
      <button onClick={onClickHandler}>Counter {state.count}</button>
    </div>
  )
}

export default App
```

As we can see that internally the state getting updating but externally it is not re rendering the component .



It is because react says I will not watch every state variable . If you want me to watch state variable you have to define the state in such a way that I am watching it .

If you want me to watch do it like these -

Step 1 - you have to import the hook name useState.

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

Step 2 - initialise a state such as -

```
const [count,setCount]=useState(0);
```

1. useState(): useState is a React Hook that allows functional components to have state. It takes one argument, which is the initial state value.
2. Array Destructuring: The left side of the expression uses array destructuring to declare two variables, count and setCount. These variables are being assigned the return values of useState(0).
3. Initial State: In useState(0), 0 is the initial state value. It means that count will initially be set to 0.
4. State Variable (count): The count variable holds the current state value. You can think of it as the equivalent of the state variable you would use in a class component.
5. State Updater Function (setCount): The setCount variable is a function that allows you to update the state. When you call setCount(newValue), it updates the count state to newValue, and then triggers a re-render of the component to reflect the updated state.

ARRAY DESTRUCTURING

Array destructuring is a JavaScript feature that allows you to unpack values from arrays or iterable objects into separate variables. It provides a more concise and readable way to extract data from arrays or objects.

```
const numbers = [1, 2, 3, 4, 5];

// Using array destructuring to unpack values from the numbers array
const [first, second, third] = numbers;

console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(third); // Output: 3
```

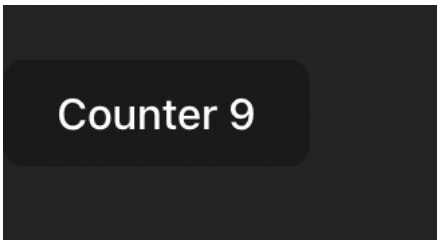
Step 3 - update the state.

```
function onClickHandler(){
  setCount(count+1)
}
```

Step 4 - inside button instead os state.count you simply have to write the count .

```
<button onClick={onClickHandler}>Counter {count}</button>
```

Result -




```
// we first initialised the state .
import { useState } from "react";


function App() {
//initialising the state .
const [count,setCount]=useState(0);
//updating the state
  function onClickHandler(){
    setCount(count+1)
  }
  return (
    // defining component .
    <div>
      <button onClick={onClickHandler}>Counter {count}</button>
    </div>
  )
}

export default App
```

This much code base is enough to write the counter application using react.

This code base is equivalent to the code what we have written in the past counter app.

This is mostly all the react code basically looks like a state which is defined above . And the component which are using these state . And the function updating these state .

Rest all the logic of re rendering is taken care by the react .

PROPS

When you render a React component, you can pass arbitrary data to it as attributes. These attributes are called props.

```
src > App.jsx > default
function App() {
//initialising the state .
const [count,setCount]=useState(0);
  return (
    <div>
      <CustomButton count={count} setCount={setCount}></CustomButton>
    </div>
  )
}
function CustomButton(props){
  function onClickHandler(){
    props.setCount(props.count+1);
  }
  return <button onClick={onClickHandler}>Counter {props.count}</button>
}

export default App
```

Accessing Props

Props are fundamental to React's component architecture and are essential for building flexible and reusable UI components. They enable communication between components in a React application, allowing you to create a hierarchy of components that can share and display data effectively.

These is the more cleaner way to write react component . And reusing them without dealing with the logic again and again .

So lets create a simple todo Application using React .
DUMB WAY -

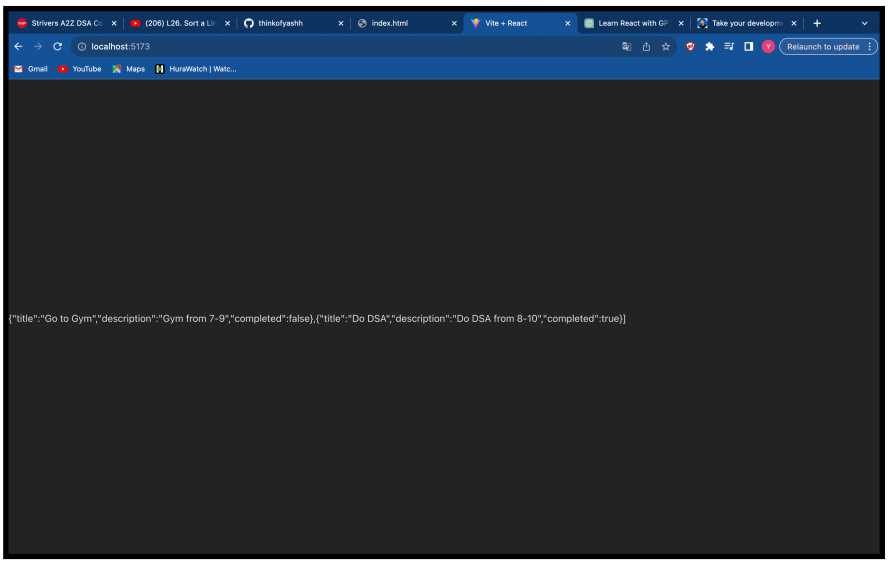
Step 1 - define/initialise state .

```
const [todos, setTodos] = useState([
  {
    title: "Go to Gym",
    description: "Gym from 7-9",
    completed: false
  },
  {
    title: "Do DSA",
    description: "Do DSA from 8-10",
    completed: true
  }
]);
```

Step 2 -

```
return (
  <div>
    {JSON.stringify(todos)}
  </div>
);
```

Result -



Another way of Doing it (better than previous) :

Step 1- Define the component .

```
// todos:
//title
//description
// define a component
function Todo(props){
  return <div>
    <h1>{props.title}</h1>
    <h2>{props.description}</h2>
  </div>
```

Step 2 -

```
function onClickHandler(){
  setTodos([...todos,{
    title:"another title",
    description:"another description"
  }])
}

return (
  <div>
    <button onClick={onClickHandler}>Random ToDo adder</button>

    <Todo title="Gym" description="at 2-3 am"></Todo>

    <Todo title={todos[0].title} description={todos[0].description}></Todo>

    {todos.map(function(t){
      return  <Todo title={t.title} description={t.description}></Todo>
    })}

  </div>
);
```

... → It is just a Spread operator.

You can create a Button.

Simplest way

Iterating over todos object.

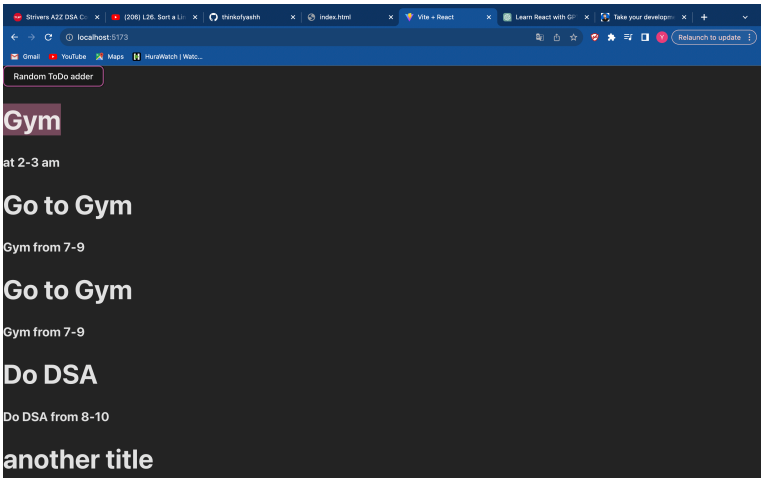
In the App component of react we can render it in direct ways -

1.We can create a button.

2.Simply creating a Todo component

3.Using map to iterate over initial state and rendering it .

Result -



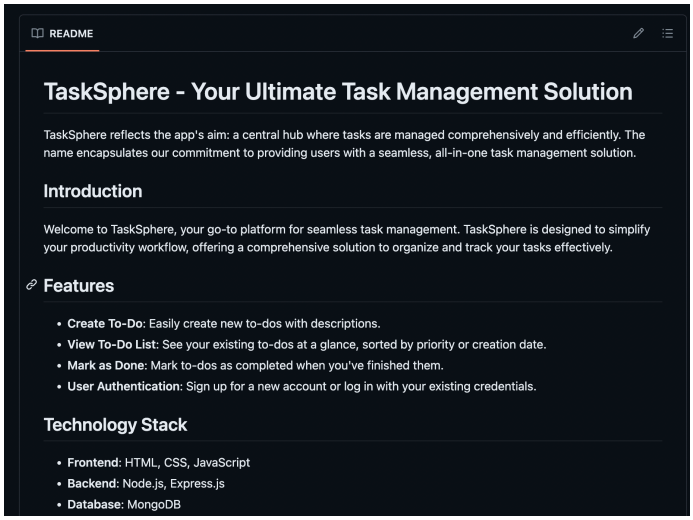
Class 5.2 (Creating a ToDo application)

STEP - 1

Generally when you start a project you create a new Directory for it .

After Creating a New Directory for it then you first Create a readme file for it .

Readme file - is the first thing the client will see . It should tell about the project . How to setup it and etc .



README

TaskSphere - Your Ultimate Task Management Solution

TaskSphere reflects the app's aim: a central hub where tasks are managed comprehensively and efficiently. The name encapsulates our commitment to providing users with a seamless, all-in-one task management solution.

Introduction

Welcome to TaskSphere, your go-to platform for seamless task management. TaskSphere is designed to simplify your productivity workflow, offering a comprehensive solution to organize and track your tasks effectively.

Features

- **Create To-Do**: Easily create new to-dos with descriptions.
- **View To-Do List**: See your existing to-dos at a glance, sorted by priority or creation date.
- **Mark as Done**: Mark to-dos as completed when you've finished them.
- **User Authentication**: Sign up for a new account or log in with your existing credentials.

Technology Stack

- **Frontend**: HTML, CSS, JavaScript
- **Backend**: Node.js, Express.js
- **Database**: MongoDB

Something like this.

STEP 2

Create a Backend Directory.

After Creating the Backend Directory the first thing we need to do it initialise a node project is to put package.json file .

Package.json file

The **package.json** file is a key component of Node.js projects, particularly those developed using npm (Node Package Manager) or yarn. It's a metadata file that contains various information about the project, such as its name, version, dependencies, scripts, and more.

1. **Project Name and Version**: Just like a title and version number for your project.
2. **Dependencies**: Other code libraries your project relies on. It's like saying, "I need these tools to work properly."
3. **Scripts**: Custom commands or actions you can run to do things like start your project, run tests, or build it for deployment.
4. **Main File**: The starting point of your project, where it all begins.
5. **Author, License, Repository**: Information about who created the project, how it can be used (license), and where the source code lives (repository).

Go to the Backend directory and run the command **npm init**

Now you need to answer few questions.

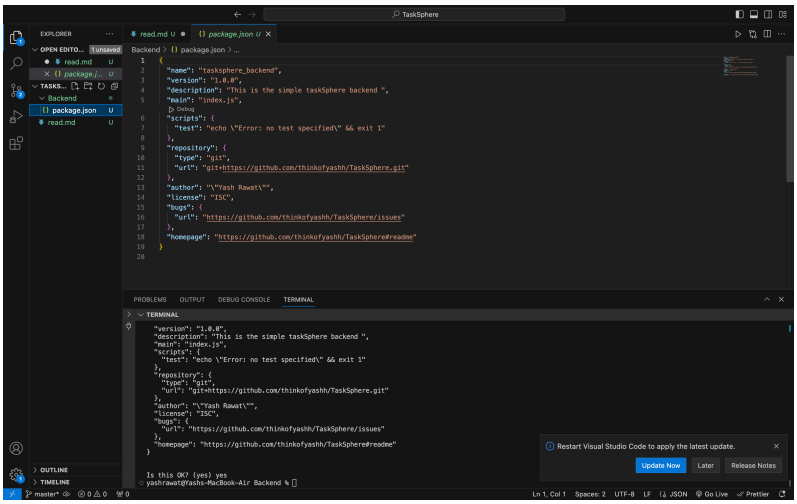
QUESTION 1 - PACKAGE NAME - WE SHOULD NAME IT AS **PROJECT_NAME_BACKEND .**
QUESTION 2 - THEN IT WILL ASK YOU FOR THE VERSION.(IT HAS A BIG SIGNIFICANT IN DEPLOYMENT.{WE WILL SEE IT LATER.}) FOR NOW JUST ENTER .
QUESTION 3 - NOW IT NEED A DESCRIPTION.
Question 4 - it will ask you about the entry point . (index.js) is fine .

And some bunch of questions you can easily answer .

```
  "bugs": {
    "url": "https://github.com/thinkofyashh/TaskSphere/issues"
  },
  "homepage": "https://github.com/thinkofyashh/TaskSphere#readme"
}


Is this OK? (yes)
```

At the end it will ask you about is this ok ?
Type yes .

And then it will create a file name **package.json** .



Now run npm install express in the terminal (in the backend directory)

```
"dependencies": {
  "express": "^4.19.2"
}
```

A dependencies section is created .
Finally a place where your all external dependencies are listed .

OK what happens when you run the command npm install something

These folders gets created and it brings lot of library from internet to your machine .

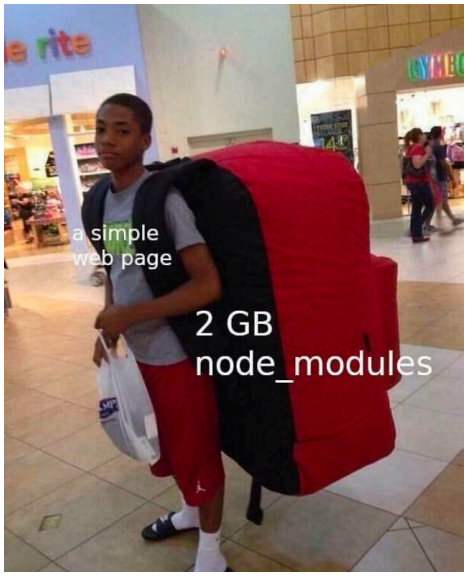
Now these libraries has their own dependencies .

So its a very heavy file .

Significance of Package.json file

We don't need to share the node modules file . We only send them a package.json file . And that person on their machine runs npm install. And gets the dependencies locally .

NODE_MODULES



STEP 3

SETTING UP ROUTES IN TASKSPHERES

There will be three routes -

1. To post new todo on the server.
2. To get todos from the backend
3. To mark todo when they are completed .

```
app.post("/todo",function(req,res){

})

app.get("/todos",function(req,res){

})
app.put("/completed",function(req,res){

})
```

STEP 4 -

Understanding what inputs does these routes gets .we need to validate the inputs .generally it is a good practice to validate these inputs . So to make user that user is sending the right input to this routes.

To make it clear you can comment each input you can expect in these routes.for example -

```
//body {
  //  title:string
   // description:String
//}
app.post("/todo",function(req,res){

})
```

So to validate the input we need Zod library .

So for that we can create a separate file for that in the backend .

In the file now we have to define the schema for different routes .for example -

```
Backend > JS types.js > [@] <unknown>
const zod=require("zod")
/*

1. route
body{
    title:String
    Description:String
}

2. we dont need input we simply have to return the list of todos from the backend.

3.
{
    id:String
}

*/
    const CreateTodo =Zod.object({
        title:Zod.string(),
        description:Zod.string()
    })

    const UpdateTodo=Zod.object({
        id:zod.string()
    })

    module.exports={
        createTodo : CreateTodo,
        updateTodo : UpdateTodo
    }
```

1.First we imported Zod library .

2.We defined what type of input does different routes want .

3.We created those Schemas according to routes need .

4.We exported the module .

Now we have to parse it into different route .

```
app.post("/todo",function(req,res){
    const newtodo=req.body.newtodo
    const validatetodo=createTodo.safeParse(newtodo)
    if(!validatetodo.success){
        res.status(411).json({
            msg:"Invalid Todo  "
        })
    }else{

    }
```

```
app.put("/completed",function(req,res){
    const todoId=req.body.todoId
    const validateiD=updateTodo.safeParse(todoId)
    if(!validateiD.success){
        res.status(411).json({
            msg:"Invalid Todo id"
        })
    }else{

    }

})
```

Step 4 -

Now you have to create the Schema for the database . So for that we have to create new file in the backend directory name db.js .

Which contain the schema for the database .

```
Backend > JS db.js > [@] <unknown> > todos
const mongoose=require("mongoose")
const { boolean } = require("zod")

mongoose.connect("mongodb+srv://thinkofyash:18NovMonday@cluster0.xay1cqj.mongodb.net/TaskSphereDb")

const todoSchema=mongoose.Schema({
    title:String,
    description:String,
    completed:Boolean
})


const todo=mongoose.model("todos",todoSchema)


module.exports={
    todos:todo
}
```

Step 5 -

Complete all the routes .

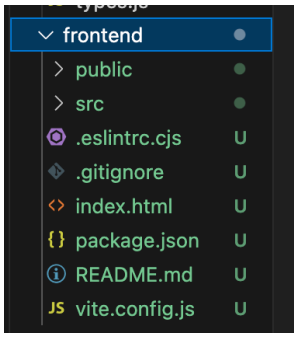
Now we are Done with backend .

Step 6 -

Setting up frontend .

Initialise a frontend project for it .

```
yashrawat@Yashs-MacBook-Air TaskSphere % npm create vite@latest
✔ Project name: … frontend
✔ Select a framework: › React
✔ Select a variant: › JavaScript

Scaffolding project in /Users/yashrawat/Desktop/project/TaskSphere/frontend...

Done. Now run:

  cd frontend
  npm install
  npm run dev

yashrawat@Yashs-MacBook-Air TaskSphere %
```

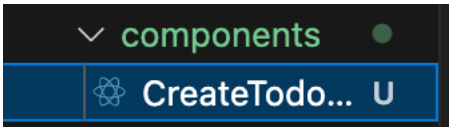


After setting it up .run **npm install** in the frontend directory so that all the dependencies will be downloaded .

STEP 7 -

Creating Components in react one for creating a todo and other for updating a **todo.so** for that we will create different directory for it .

And inside it create a file who's name start with capital letter and has a extension of jsx file .



Now inside this file we will create a function name CreateTodo and export it which returns us Xml .

```
frontend > src > components > CreateTodo.jsx > CreateTodo
1   export function CreateTodo(){
2       return <div>
3
4           <input type="text" placeholder="Title" /> <br />
5           <input type="text" placeholder="Description" /> <br />
6
7           <button>Add a ToDo</button>
8
9
10      </div>
11  }
```

Now after this running command npm run dev in the react directory and saving it .

Now lets import it into the App.jsx

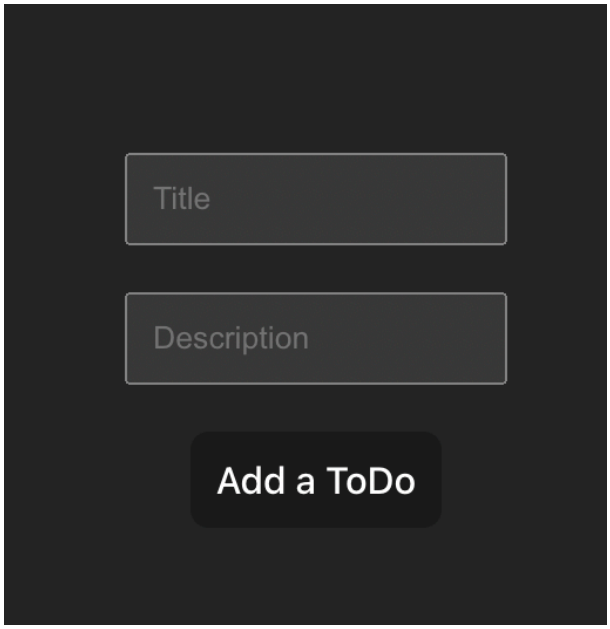
```
frontend > src > App.jsx > App
1   import { useState } from 'react'
2   import reactLogo from './assets/react.svg'
3   import viteLogo from '/vite.svg'
4   import './App.css'
5   import { CreateTodo } from './components/CreateTodo'
6
7   function App() {
8     const [count, setCount] = useState(0)
9
10    return (
11      <div>
12        <CreateTodo></CreateTodo>
13      </div>
14    )
15  }
16
17  export default App
18
```

Result -

Now you will be like how do I style these components . The one way of doing it is by -

```
<input style={{}} type="text" placeholder="Title" /> <br />
```

```
frontend > src > components > CreateTodo.jsx > CreateTodo
1   export function CreateTodo(){
2       return <div>
3
4           <input style={{
5               padding:10,
6               margin:10
7           }} type="text" placeholder="Title" /> <br />
8
9           <input style={{
10              padding:10,
11              margin:10
12          }} type="text" placeholder="Description" /> <br />
13
14          <button style={{
15              padding:10,
16              margin:10,
17
18          }} >Add a ToDo</button>
19
20
21      </div>
22  }
```



Step 8 -

Rendering a Todos list .but before that creating a component for it first .

Creating a new file in the components . **Name - Todo.jsx**

```
frontend > src > components > Todos.jsx > Todos
1   export function Todos(){
2       return <div>
3           <h1>Go to the Gym</h1>
4           <h2>From 8am to 10 am</h2>
5           <button>Mark as Completed</button>
6       </div>
7   }
```

Now the thing is you don't have it statically . It will be dynamic .

So there are two ways to make it dynamic -
1. Props way .
2. Array Restructuring .

PROPS

In React, "props" is short for "properties," and it's a way to pass data from a parent component to a child component. Props are read-only, meaning that a child component cannot modify the props it receives from its parent.

```jsx
<ChildComponent name="John" age={30} />
```

```jsx
function ChildComponent(props) {
  return <div>{props.name} is {props.age} years old.</div>;
}
```

```
export function Todos(props){
    const todos=props.todos;
    return <div>
        <h1>Go to the Gym</h1>
        <h2>From 8am to 10 am</h2>
        <button>Mark as Completed</button>
    </div>
}
```

Array Destructuring

In modern way it is done through array destructuring .

Array destructuring is a feature in JavaScript that allows you to extract values from arrays and assign them to variables in a more concise way. It's particularly useful when working with functions that return arrays or objects.

```javascript
const numbers = [1, 2, 3];

// Extracting values from the array using array destructuring
const [a, b, c] = numbers;

console.log(a); // Output: 1
console.log(b); // Output: 2
console.log(c); // Output: 3
```

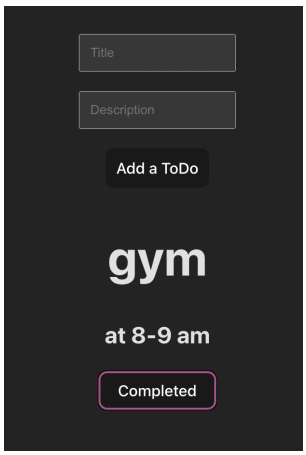
This is how you do array destructuring .
You use map to iterate over loop .

```
frontend > src > components > Todos.jsx > Todos > todos.map() callback
export function Todos({todos}){

    return <div>
        {todos.map(function(todo){
            return <div>
                <h1>{todo.title}</h1>
                <h2>{todo.description}</h2>
                <button>{todo.completed == true? "Completed":"Mark as Complete"}</button>
            </div>
        })}
    </div>
}
```

Note - You can only return one element . Not many element simultaneously.

```
<Todos todos={[
    {
      title:"gym",
      description:"at 8-9 am",
      completed:true
    }
  ]}></Todos>
```

—->



Step 9 -

The question is now how do you hit the backend and put it into the state.

Step 1 - Initialise the state first .

```
const [todo,setTodo]=useState([])
```

Step 2 - Now using the fetch api . Fetch all the todos .

```
const output=await fetch("http://localhost:3000/todos")
const res=await output.json()
```

Step 3 - now update the state .

```
setTodo(res.json())
```

Now you will an error name - Cors error

todos	CORS error	fetch	App.jsx:10	0 B	183
todos	CORS error	fetch	App.jsx:10	0 B	92
todos	CORS error	fetch	App.jsx:10	0 B	119
todos	CORS error	fetch	App.jsx:10	0 B	262
todos	CORS error	fetch	App.jsx:10	0 B	147

So to solve these error go to your backend file and run npm install cors and import it .

It will solve the previous cors error but it still doesn't render the todo .instead all the screen will be blank.

Is because we can't use async ,await syntax directly in the component . We need to use the useEffects hook for it (we will come to it later)

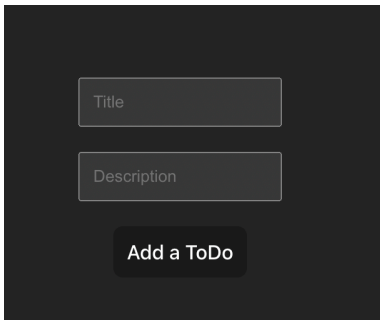
But for now to fetch things we can use .then method .and then set the state .

```
fetch("http://localhost:3000/todos").then(
  async function(res){
    const data=await res.json()
    setTodo(data.Todos)

  }
)
```

But this is not the right way to send the request to the backend application .Whats happening is we keep on sending the request to the express application and re rendering it again and again .

The right way is **useEffect hook .**

Step 10 -



Every time the button Add ToDo is clicked I want it to render it on the screen .

So for that we have to go to the components directory .and then to the CreateTodo file.

Now in button element we have to add onClick attribute .

```
}} onClick={()=>{fetch("http://localhost:3000/todos",{
    method:"POST",
    body: JSON.stringify({
        title:document.getElementById("title").innerHTML,
        description:document.getElementById("description").innerHTML
    }),
    headers:{
        "contentType":"application/json"
    }
}).then(async function(e){
    const data=await e.json()
    alert("Todo Added")
})}} >Add a ToDo</button>
```

1. **onClick Event**: This function is attached to the `onClick` event of an HTML element. When the element is clicked, this function will be executed.

2. **Fetch API**: The `fetch` function is used to make an HTTP request to the specified URL (`http://localhost:3000/todos`). This request is a POST request, indicating that it's intended to create a new resource on the server.

3. **Request Body**: The `body` property of the fetch options is set to a JSON string representing the data to be sent to the server. This data includes a `title` and `description`, which are extracted from HTML elements with the IDs `"title"` and `"description"` respectively. The `innerHTML` property retrieves the content within those HTML elements.

4. **Request Headers**: The `headers` property specifies the content type of the request body as JSON using the `"Content-Type"` header.

5. **Response Handling**: After the fetch request is made, a promise is returned. The `.then` method is used to handle the asynchronous response from the server. Inside the callback

function, the response is converted to JSON format using the `.json()` method. Once the response data is available, an alert is displayed with the message "Todo Added".

But this is wrong way of doing it . Because the whole idea of using react is to get rid of using DOM.

Better way of doing it

Step 1 - creating the local state of both title and the description

```
const [title,setTitle]=useState("")
const [description,setDescription]=useState("")
```

Step 2 - Setting the state using onChange attribute .

```
<input style={{
    padding:10,
    margin:10
}} type="text" placeholder="Title" onChange={function(e){
    const value=e.target.value
    setTitle(e.target.value)
}} /> <br />

<input style={{
    padding:10,
    margin:10
}} type="text" placeholder="Description" onChange={function(e){
    const value=e.target.value
    setDescription(e.target.value)
}} /> <br />
```

1. **onChange Event**: This function is attached to the `onChange` event of an input element. The `onChange` event is triggered whenever the value of the input element changes, such as when a user types in new text.

2. **Event Object (`e`)**: When the `onChange` event is triggered, it passes an event object (`e`) to the event handler function. This event object contains information about the event, such as the target element that triggered the event.

3. **Extracting Value**: Inside the event handler function, `e.target.value` is used to extract the current value of the input element that triggered the event. This represents the new value entered by the user.

4. **Updating State**: The extracted value (`value`) is then used to update the state of the component. In this example, `setTitle` is a state setter function (likely obtained from the `useState` hook), which updates the state variable responsible for storing the title value.

Step 3 - Using this state while sending the post request .

```
onClick={()=>{fetch("http://localhost:3000/todo",{
 method:"POST",
 body: JSON.stringify({
     title:title,
     description:description
 }),
 headers:{
     "Content-Type":"application/json"
 }
```

Now what's left to fix is -

1. Infinte request to get todos
2. Not yet implemented update todos.

End of week 5

Assignment

React