

# Complete React Notes

11 September 2023 08:14 PM

What is Library and framework?

React is a library made by facebook, similar to jQuery

Carousel is a JS library

Framework is complete in itself, it has everything needed to create an app

**A framework is a set of pre-written code that provides a structure for developing software applications.** A library, on the other hand, is a collection of pre-written code that can be used to perform specific tasks

Both the framework vs library is pre-coded support programs to develop complex software applications. However, **libraries target a specific functionality, while a framework tries to provide everything required to develop a complete application.**

It takes minimum effort for a library to put in inside our code

What is emmet?

Emmet is a set of plug-ins for text editors that allows for high-speed coding and editing in HTML, XML, XSLT, and other structured code formats via content assist.

How to create a H1 using JS and put it inside a div with id root?

Browser has a JS engine which interprets the code written below and react accordingly

```
<title>Namaste React</title>
</head>
<body>
  <div id="root"></div>
</body>
<script>

  const heading = document.createElement("h1");

  heading.innerHTML = "Namaste Everyone from JavaScript!";

  const root = document.getElementById("root");

  root.appendChild(heading);
```

What is React CDN?

Content delivery/distribution network

A content delivery network (CDN) is a **network of interconnected servers that speeds up webpage loading for data-heavy applications.** CDN can stand for content delivery network or content distribution network.

What is cross origin in Script tag?

```
<script>
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
</script>
<script>
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
</script>
```

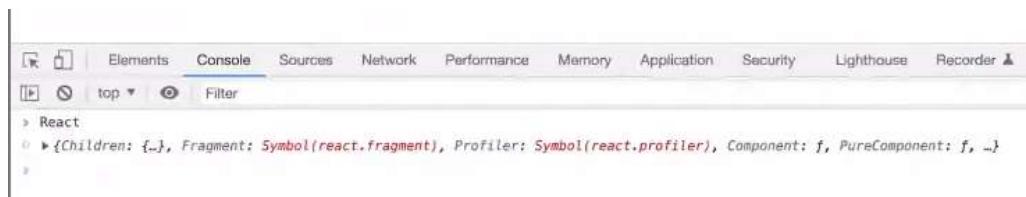
The crossorigin attribute **sets the mode of the request to an HTTP CORS Request**. Web pages often make requests to load resources on other servers. Here is where CORS comes in. A cross-origin request is a request for a resource (e.g. style sheets, iframes, images, fonts, or scripts) from another domain

### Shortest Program of React?

The below is shortest program of react, we have just injected react CDN into our document

```
index.html X
index.html > html > script
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Namaste React</title>
</head>
<body>
  <div id="root"></div>
</body>
<script>
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
</script>
<script>
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
</script>
</html>
```

Now if we write React in console we get this



We can also use React.createContext etc etc in console now.

React is a global object and it can be used anywhere using React.something anywhere because we have added react CDN now

We also access to React DOM now

```
> ReactDOM
✖ ↳ _SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED: {...}, createPortal: f, createRoot: f, findDOMNode: f, flushSync: f, ...
  ↳ createPortal: f createPortal$1(children, container)
  ↳ createRoot: f createRoot$1(container, options)
  ↳ findDOMNode: f findDOMNode(componentOrElement)
  ↳ flushSync: f flushSync$1(fn)
  ↳ hydrate: f hydrate(element, container, callback)
  ↳ hydrateRoot: f hydrateRoot$1(container, initialChildren, options)
  ↳ render: f render(element, container, callback)
  ↳ unmountComponentAtNode: f unmountComponentAtNode(container)
  ↳ unstable_batchedUpdates: f batchedUpdates$1(fn, a)
  ↳ unstable_renderSubtreeIntoContainer: f renderSubtreeIntoContainer(parentComponent, element, containerNode, cai)
  ↳ version: "18.2.0"
  ↳ _SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED: {usingClientEntryPoint: false, Events: Array(6)}
  ↳ [[Prototype]]: Object
:
: Console What's New ×
```

Use of those 2 CDN links were

```
> React
✖ ↳ {Children: {}}, Fragment: Symbol/react.fragment, Profiler: Symbol/react.profiler, Component: f, PureComponent: f, ...
> ReactDOM
✖ ↳ _SECRET_INTERNALS_DO_NOT_USE_OR_YOU_WILL_BE_FIRED: {...}, createPortal: f, createRoot: f, findDOMNode: f, flushSync: f, ...
:
```

Why there are 2 files for React and ReactDOM?

React is not limited to browsers only there is React native also for mobiles

ReactDOM means web version of React which gives us access to DOM

Let us use React Now, do same H1 thing using React

We create h1 now like this:

```
<script>

  const heading = React.createElement("h1", {}, "Namaste Everyone!");
```

Now we want to render this heading inside div of id = "root"

We use const root = ReactDOM.createRoot(document.getElementById('root')) to tell react that this is my root

Now we use root.render(heading)

```
<script>
  const heading = React.createElement("h1", {}, "Namaste Everyone!");

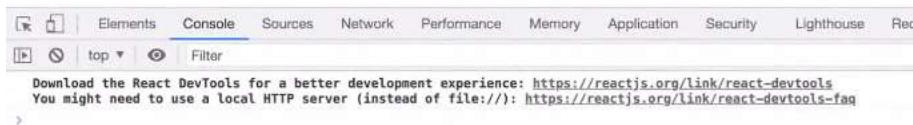
  const root = ReactDOM.createRoot(document.getElementById("root"));

  root.render(heading);
</script>
```

Output is



Namaste Everyone!



If we do `console.log(heading)`  
We see an object of type `h1`  
`render()` injects element in to DOM

```
↳ $typeof: Symbol(react.element), type: 'h1', key: null, ref: null, props: {...}, ...]
  ↳ $typeof: Symbol(react.element)
  ↳ key: null
  ↳ props: {children: 'Namaste Everyone!'}
  ↳ ref: null
  ↳ type: "h1"
  ↳ _owner: null
  ↳ _store: {validated: false}
  ↳ _self: null
  ↳ _source: null
  ↳ [[Prototypal]]: Object
```

Can we have multiple roots?  
Generally in our react app, we have only 1 root and 1 render method

What if we have something on top of div with Id = root?

Let say we make div of id header  
And div of Id = root  
And div of Id = footer  
What will be output?

```
<h1>Header</h1>
<div id="root"><h1>Footer</h1></div>
</body>
<script>
```

Everything will run as it is just that React will be rendered inside root only. There can be header or footer also. So we can use react anywhere in our project like search bar, footer etc etc



## Header

Namaste Everyone!

## Footer

What is {} in React.createElement?

```
const heading = React.createElement("h1", {}, "Namaste Everyone!");
```

Let say we have one more heading with id = title inside div with id = root

```
<body>
  <div id="root">
    <h1 id="title">Hello world</h1>
  </div>
</body>
<script
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
></script>
<script
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
></script>
<script>
  const heading = React.createElement("h1", [
    {
      id: "title",
      children: "Namaste Everyone!"
    }
  ], "Namaste Everyone!");

```

So to make changes in id = title we can pass these parameters inside {}

```
const heading = React.createElement(
  "h1",
  {
    id: "title",
  },
  "Namaste Everyone!"
);
```

Our DOM now looks like

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div id="root">
      <h1 id="title">Namaste Everyone!</h1> == $8
    </div>
    <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
    <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
  </body>
</html>
```

Now if we fill root with many h1 headings

Now if we render heading "Namaste Everyone" inside root what will happen?

React will overwrite everything and only Namaste Everyone will be there inside root

```
<body>
  <div id="root">
    <h1 id="title">Hello world</h1>
    <h1 id="title">Hello world</h1>
  </div>
</body>
```

Output

Namaste Everyone!

We generally write "Not Rendered Correctly" inside root div

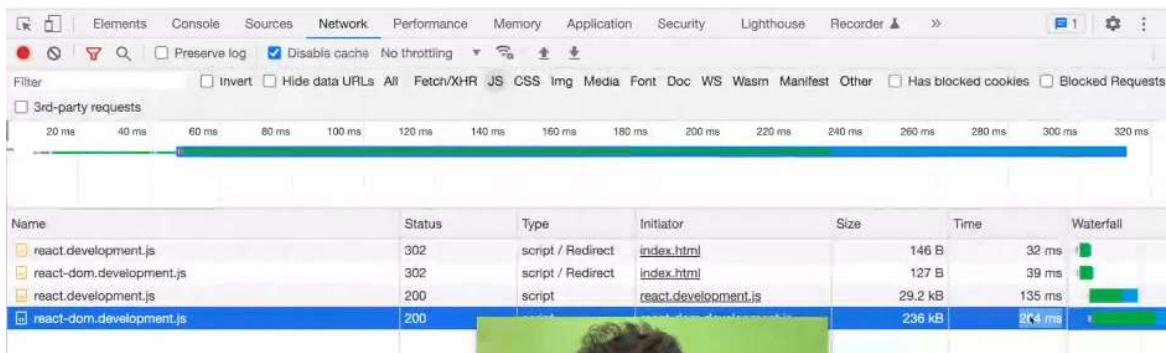
To make sure that if there is some error we see that message and can solve error that why root not rendered correctly

```
<body>
  <div id="root">Not Rendered</div>
</body>
<script>
```

It takes some time for react to render in our browser so we always see "Not Rendered Correctly" for sometime on doing refresh.

It takes sometime for scripts to load

Namaste Everyone!



What if we put script tag inside body below div id = root?  
It will not work.

```

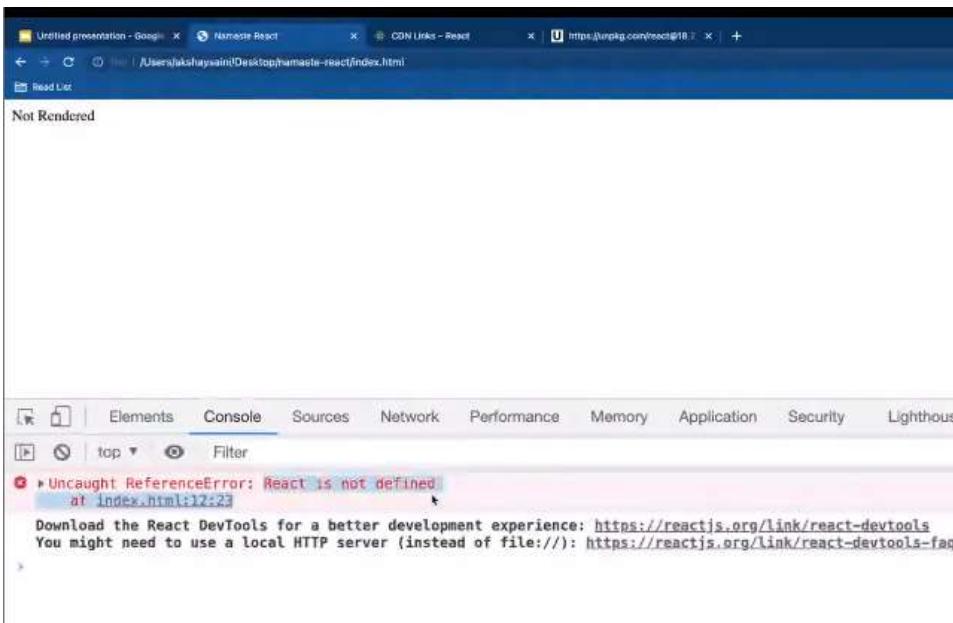
<body>
  <div id="root">Not Rendered</div>
  <script>
    const heading = React.createElement(
      "h1",
      {
        id: "title",
      },
      "Namaste Everyone!"
    );

    console.log(heading);

    const root = ReactDOM.createRoot(document.getElementById("root"));

    //passing a react element inside the root
    root.render(heading);
  </script>
  <script
    crossorigin
    src="https://unpkg.com/react@18/umd/react.development.js"
  ></script>
  <script
    crossorigin
    src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
  ></script>
</body>

```



What is difference between `async`/`defer`?

**Async** allows your script to run as soon as it's loaded, without blocking other elements on the page. **Defer** means your script will only execute after the page has finished loading. In most cases, `async` is the better option — but there are exceptions

`Async` in `script` tag is a way to load scripts asynchronously. That means, if a script is `async`, it will be loaded independently of other scripts on the page, and will not block the page from loading.

If you have a page with several external scripts, loading them all asynchronously can speed up the page load time, because the browser can download and execute them in parallel.

To use `async`, simply add the `async` attribute to your script tag:

```
<script async src="script.js"></script>
```

By using the `defer` attribute in HTML, the browser will load the script only after parsing (loading) the page. This can be helpful if you have a script that is dependent on other scripts, or if you want to improve the loading time of your page by loading scripts after the initial page load.

To use `defer`, simply add the `defer` attribute to your script tag:

```
<script defer src="script.js"></script>
```

If we want to build the below structure in our HTML using React. How to do it?

```
<body>
  <div id="root">Not Rendered</div>

  <div id="container">
    <h1>Heading 1</h1>
    <h2>Heading 2</h2>
  </div>

  <script
    crossorigin
    src="https://unpkg.com/react@18/umd/react.development.js"
  ></script>
  <script
    crossorigin
    src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
  ></script>
  <script>
```

We do it like:

We will pass heading 1 and heading 2 as an Array

We put heading 1 and heading 2 inside container

Now we render the container inside root

```

<script>
  const heading = React.createElement(
    "h1",
    {
      id: "title",
    },
    "Heading 1"
  );

  const heading2 = React.createElement(
    "h2",
    {
      id: "title",
    },
    "Heading 2"
  );

  const container = React.createElement(
    "div",
    {
      id: "container",
    },
    [heading, heading2]
  );

  console.log(container);

  const root = ReactDOM.createRoot(document.getElementById("root"));

  //passing a react element inside the root

  //async defer
  root.render(container);

```

Our DOM looks like:



If we want to build a big index.html

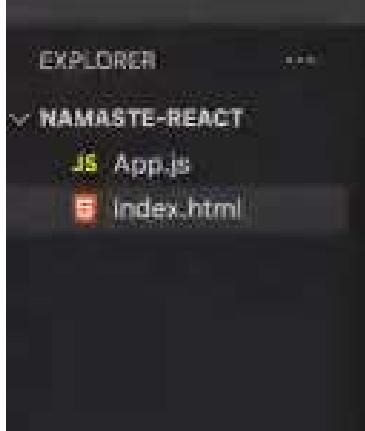
The above method is not development-friendly

React came with a idea to build HTML using JS

We need not to go to HTML anymore, we can do anything from React using APIs like createElement() etc etc.

To make a complex file, its better to split our components and put all JS in app.js

```
<script>
  crossorigin
  src="https://unpkg.com/react@18/umd/react.development.js"
</script>
<script>
  crossorigin
  src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
</script>
<script src="App.js"></script>
</body>
</html>
```



Why do we import CSS inside head in HTML?

**seeks to reduce the number of times the browser must re-flow the document by ensuring that the CSS styles are all parsed in the head, before any body elements are introduced.**

This is based on the best practice for optimizing browser rendering.

What is rel = stylesheet in link tag while linking CSS in HTML?

The REL attribute is used **to define the relationship between the linked file and the HTML document**. REL=StyleSheet specifies a persistent or preferred style while REL="Alternate StyleSheet" defines an alternate style. A persistent style is one that is always applied when style sheets are enabled.

CDN of react.development.js is for development

CDN of react.production.js has same code but it is much more optimised and for production

## Session 2

Revising JS

What is function keyword in JS?

It is present inside JS,

Data Structure used for memory in JS is Heap

Function without name is anonymous function which can be assigned to a variable as well

```
function x() {
    const a = 10;
}
var xyz = 30;

x(); //functional execution context is created

var x = function () {
    console.log("I'm an anonymous function");
}
```

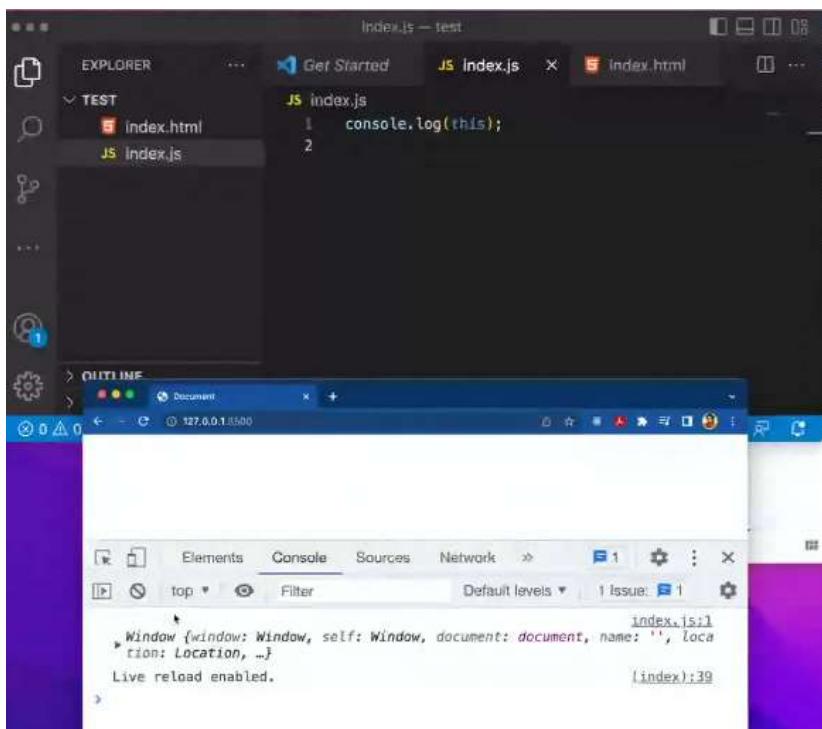
Expression is something that executes. Like `console.log` is a expression but `a = 30` is not a expression.

Arrow function was introduced in ES6 with let, const, promises, spread operator etc etc  
=> is known as fat-arrow

Only difference between normal function and arrow function is 'this' keyword

Arrow function, 'this' refers to window

Output of `console.log(this)` is window object



In normal function 'this' refer to parent object

The screenshot shows a VS Code interface with an open file named `index.js`. The code defines a variable `obj` containing a function `fn` that logs `this`. When the code is run in the browser's developer tools, it shows that `obj` has a prototype chain starting with the global `Window` object.

```
index.js - test
EXPLORER ... Get Started JS index.js × index.html ...
TEST
JS index.html > obj
JS index.js
1 console.log(this);
2
3 const obj = [
4   fn: function () {
5     console.log(this);
6   },
7 ];
8 obj.fn();
9

> OUTLINE
> TIMELINE
0 tabnine starter Spaces: 4 UTF-B-LF () JavaScript Port: 5500 ✓ Prettier D

Elements Console Sources Network > Default levels 1 Issue: 1
top Filter index.js:1
Window {window: Window, self: Window, document: document, name: '', location: Location, ...}
  ▾ {fn: f} index.js:5
    ▾ fn: f {}
    ▾ [[Prototype]]: Object
>


```

But for arrow function it refers to window object

The screenshot shows a VS Code interface with an open file named `index.js`. It contains two functions: `fn` (a normal function) and `fn2` (an arrow function). Both log `this`. In the browser's developer tools, the prototype chain for `fn2` is shown as starting with the global `Window` object, while `fn` starts with the object it was defined on.

```
index.js - test
EXPLORER ... Get Started JS index.js × index.html ...
TEST
JS index.html > ...
JS index.js
1 console.log(this);
2
3 const obj = {
4   fn: function () {
5     console.log(this);
6   },
7   fn2: () => {
8     console.log(this);
9   },
10 };
11 obj.fn();
12 obj.fn2();
13

> OUTLINE
> TIMELINE
0 tabnine starter Spaces: 4 UTF-B-LF () JavaScript Port: 5500 ✓ Prettier D

Elements Console Sources Network > Default levels 1 Issue: 1
top Filter index.js:1
Window {window: Window, self: Window, document: document, name: '', location: Location, ...}
  ▾ {fn: f, fn2: f} index.js:5
    ▾ fn: f {}
    ▾ fn2: f {}
    ▾ [[Prototype]]: Object
  ▾ {window: Window, self: Window, document: document, name: '', location: Location, ...}
    ▾ [[Prototype]]: Object
>


```

Window is the global object given to us by the browser

What if we have normal function and another normal function inside it.

```
EXPLORER Get Started JS index.js index.html
TEST index.html JS index.js
JS index.js > ② x
1 function x() {
2   console.log(this);
3   function y() {
4     console.log(this);
5   }
6   y();
7 }
8 x();
9
```

> OUTLINE  
> TIMELINE  
0 o tabnine starter Spaces: 4 UTF-8 LF {} JavaScript ⚙ Port: 5500 ✓ Prettier

Elements Console Sources Network > 1 Issue: 1

Default levels: 1 Issue: 1

index.js:2  
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

index.js:4  
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

What if we have one more function inside it Z( )

```
EXPLORER Get Started JS index.js index.html
TEST index.html JS index.js
JS index.js > ② x > ③ y > ④ z
1 function x() {
2   console.log(this);
3   function y() {
4     console.log(this);
5     function z() {
6       console.log(this);
7     }
8     z();
9   }
10 y();
11 }
12 x();
```

> OUTLINE  
> TIMELINE  
⚠ 0 o tabnine starter Spaces: 4 UTF-8 LF {} JavaScript ⚙ Port: 5500 ✓ Prettier

Elements Console Sources Network > 1 Issue: 1

Default levels: 1 Issue: 1

index.js:2  
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

index.js:4  
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

index.js:6  
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

If we make a function and an object.

We call that object using the function. Let's see what happens

EXPLORER    ...    Get Started    JS Index.js    index.html

TEST

index.html  
JS index.js

```
JS index.js > ...
1 const person = {
2   name: "Akshay",
3 };
4 const person2 = {
5   name: "Simran",
6 };
7
8 function x() {
9   console.log(this);
10 }
11 x.call(person);
12
```

> OUTLINE  
> TIMELINE

1.0 o tabnine starter Spaces: 4 UTF-8 LF {} JavaScript Port : 5500 ✓ Prettier

Elements Console Sources Network >    1 issue: 1

top Filter Default levels > 1 issue: 1

▶ {name: 'Akshay'}    index.js:9

EXPLORER    ...    Get Started    JS Index.js    index.html

TEST

index.html  
JS index.js

```
JS index.js > ...
1 const person = {
2   name: "Akshay",
3 };
4 const person2 = {
5   name: "Simran",
6 };
7
8 function x() {
9   console.log(this);
10 }
11 x.call(this);
12 x.call(person);
13 x.call(person2);
14
```

> OUTLINE  
> TIMELINE

0 o tabnine starter Spaces: 4 UTF-8 LF {} JavaScript Port : 5500 ✓ Prettier

Elements Console Sources Network >    1 issue: 1

top Filter Default levels > 1 issue: 1

▶ {name: 'Akshay'}    index.js:9  
▶ {name: 'Simran'}    index.js:9  
▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}    index.js:9

What if we put that function inside object

The screenshot shows the VS Code interface. The code editor has two tabs: 'index.html' and 'index.js'. The 'index.js' tab is active, displaying the following code:

```
const person = {
  name: "Akshay",
  print: function () {
    console.log(this);
  }
};
const person2 = {
  name: "Simran",
};
person.print();
```

The status bar at the bottom indicates 'Ln 12, Col 1' and 'JavaScript'. Below the code editor is the 'PROBLEMS' panel, which shows one error: '▶ {name: 'Akshay', print: f}'.

What if we use .call()

call takes window object and then this refers to window object

The screenshot shows the VS Code interface. The code editor has two tabs: 'index.html' and 'index.js'. The 'index.js' tab is active, displaying the following code:

```
> const person = {
  name: "Akshay",
  print: function () {
    console.log(this);
  }
};
> const person2 = {
  name: "Simran",
};
person.print();
person.print.call();
person.print.call(this);
```

The status bar at the bottom indicates 'Ln 13, Col 26' and 'JavaScript'. Below the code editor is the 'PROBLEMS' panel, which shows three errors:

- ▶ {name: 'Akshay', print: f} index.js:4
- ▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...} index.js:4
- ▶ Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

What if we use .call(object)

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows files: index.html, index.js.
- TEST**: Shows index.js with the following code:

```
1 const person = {
2   name: "Akshay",
3   print: function () {
4     console.log(this);
5   }
6 };
7 const person2 = {
8   name: "Simran",
9 };
10 person.print();
11 person.print.call();
12 person.print.call(this);
13 person.print.call(person2);
14 person.print.call(person2);
```

- TIMELINE**: Shows a timeline entry for a tabnine starter at Line 14, Col 28.
- BROWSER**: Shows the browser developer tools with the call stack:
  - > {name: 'Akshay', print: f} index.js:4
  - > Window {window: Window, self: Window, document: document, name: '', location: Location, ...} index.js:4
  - > Window {window: Window, self: Window, document: document, name: '', location: Location, ...} index.js:4
  - > {name: 'Simran'} index.js:4

This refers to window in arrow function that is why we are getting undefined

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows files: index.html, index.js.
- TEST**: Shows index.js with the following code:

```
1 const obj = {
2   firstName: "Akshay",
3   printName: () => {
4     console.log(this.firstName);
5   }
6 };
7
8 obj.printName();
```

- BROWSER**: Shows the browser developer tools with the output of the code execution:
  - Console tab: undefined
  - Call stack: index.js:4

## Interview Tips

Luck is very important and we cannot control it.  
Many Companies don't train their interviewers well  
A person can be a good software engineer but bad interviewer

What we can control is:

## 1. Our preparation

### **Technical preparation**

**Communication skills:** Lot of people fail due to communication skills ,We spend so much time for tech skills but comm skills are equally important.

Learn to speak while you write, speak your thoughts.

Practice to speak even when you are coding alone.

### **Mock Interviews**

## 2. Talk while you are coding so that interviewer can also see what you are doing.

If you cannot explain, interviewer thinks you also don't know or you have crammed things.

## 3. In a company you don't work alone so comm skills should be good. Does not matter how good Software engineer you are, you should be able to communicate your thoughts and ideas to others.

## 4. Spoken english is very important.

## 5. Preparation on the interview day (not technical preparation)

You should not be in panic state before/during interview

Keep your pen and paper handy

Keep you water, next to you.

Keep your laptop charged, keep your charger handy.

Keep power point near to you.

These small things mess your interview.

Keep your camera always open during interview.

Keep your phone on silent.

Have a power backup, mini UPS (it does not cost much).

## 6. Confidence comes from preparation (not just your technical prep) (it includes non tech prep also which are mentioned above).

# Session 3

How to make our app deployable?

How to launch our app

How to build our own create-react-app

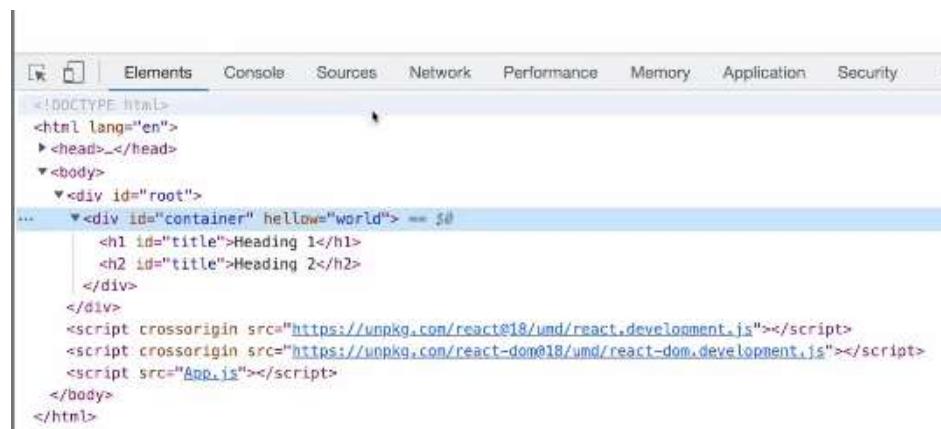
Let say with ID we also pass something which is not an attribute, will it also get passed in container?

```

const heading2 = React.createElement(
  "h2",
  {
    id: "title",
  },
  "Heading 2"
);
|
|
const container = React.createElement(
  "div",
  {
    id: "container",
    hellow: "world",
  },
  [heading, heading2]
);

```

Yes, we can put in anything over { } while using createElement



They are not called as attributes

They are called as props just like properties or attributes

Is this a production ready app?

What it needs to be production ready?

Bundle Everything, remove console, server, optimisation, Caching, minify many things

We need to use something known as Bundlers

Same as web-pack, it is a type of bundler

Vite is also a bundler, parcel is also a bundler

We will be using Parcel, in original create-react-app, web-pack is used as Bundler and it also uses Babel.

Module Bundling, on a high level, is a process of integrating together a group of modules in a single file so that multiple modules can be sent to the browser in a single bundle.

When we write our code in a modular pattern:

- We keep our javascript in separate files and folders based on functionality.
- Add the script tag for each file that we are using in correct order of dependency.

For each script tag , browser will send the request to the server which will have bad effect on the performance of our application. In order to overcome this we usually create a single bundled file which will integrate all the other files and that bundled file is sent to browser.

Module bundling can also include a minification step i.e. all the unnecessary characters like space, comma, comments etc. are removed from the file and its minified version is created

and whenever a request comes from the browser that minified version is sent back. Less data means less browser processing time.

#### **Webpack vs Rollup vs Parcel :**

All these differ in following things, although there is very little difference.

They almost do the same job

Configuration, Entry points, Transformations, Tree Shaking, Dev Server , Hot Module Replacement, Code Splitting

All above optimisations are done by bundlers only.

What is Parcel?

Parcel is a package, means a module of JS files

Some piece of code so we need a package manager also

That is why we do "npm init"

What can we do other than "npm"?

We can also use "yarn"

What is npm?

It does not stands for node package manager

If we go to official doc of npm we see



Nerdiest Precious Modules



New Priority Mail



Nanometers Per Millisecond



Nitrogen Poisonous Monoxide

But now-where is written node package manager.

There is a repo of npm where they take any name for npm.

So there is no official name for npm

**npm** is the world's largest **Software Registry**.

The registry contains over 800,000 **code packages**.

How to install npm?

We write "npm init"

{

"name" : "foo",

"version" : "1.2.3",

"description" : "A package for fooing things",

"main" : "foo.js",

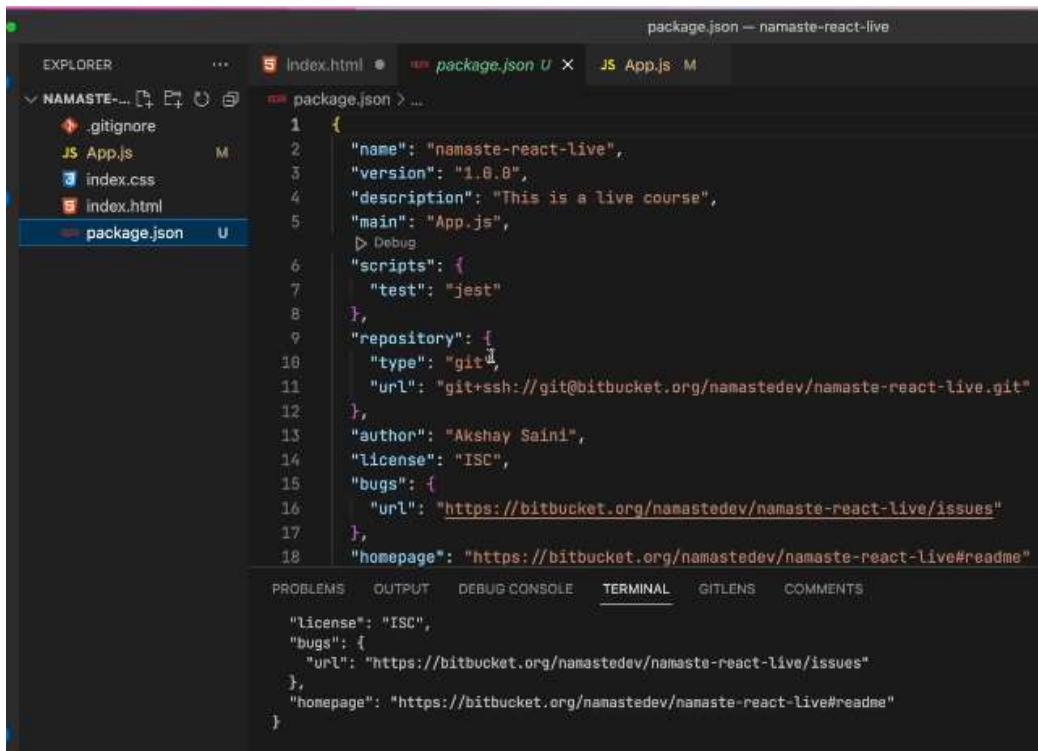
"keywords" : ["foo", "fool", "foolish"],

"test command": jest,

"author" : "John Doe",

```
"licence" : "ISC"  
}
```

Now we get package.json in our file and folder structure



The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows a folder named "NAMASTE..." containing ".gitignore", "App.js", "index.css", "index.html", and "package.json".
- Editor:** The "package.json" file is open, displaying its JSON content. The content includes fields like "name", "version", "description", "main", "scripts", "repository", "author", "license", "bugs", and "homepage".
- Bottom Bar:** Shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, GITLENS, and COMMENTS.
- Terminal:** At the bottom, there is a terminal window showing the command "npm install -D parcel".

Why do we use npm?

Because we use so many packages so we need someone to manage these many packages. Our app does not run only on react, we need certain packages to build it. These packages come inside npm.

What is inside package.json?

All information we filled.

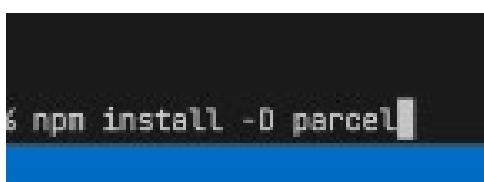
How to ignite our app?

We use parcel (Zero configuration build tool for everything)

npm install to install a package + package name

We do not want parcel in production we want it for development so we use -D

npm install -D parcel where -D means devDependency



```
6 npm install -D parcel
```

Some people also do --save-dev also instead of -D, it is same

Dependency means all packages that my project needs, my project is dependent on it.

Parcel is one of the dependency which our react project need as bundler.

Now we get package.lock.json and in package.json we have devDependencies {  
 Parcel : "version" }



```
package.json — namaste-repo

EXPLORER          ... 5 index.html  package.json U X JS App.js M

✓ NAMASTE-REACT-LIVE
> node_modules:
  .gitignore
  JS App.js M
  A index.css
  S index.html
  package-lock.json U
  package.json U

  package.json > () devDependencies:
    "author": "Akshay Saini",
    "license": "ISC",
    "bugs": {
      "url": "https://bitbucket.org/namastedev/namaste-react-live"
    },
    "homepage": "https://bitbucket.org/namastedev/namaste-react-live",
    "devDependencies": {
      "parcel": "^2.8.2"
    }
  }
```

Package-lock.json is an important file.

<sup>^</sup> is known as caret

```
homepage: "https://bit.ly/2PZBwDy"
"devDependencies": {
  "parcel": "^2.8.2"
}
```

What's the difference between tilde(~) and caret(^) in package.json?

- **~version** “**Approximately equivalent to version**”, will update you to all future patch versions, without incrementing the minor version. `~1.2.3` will use releases from `1.2.3` to `<1.3.0`. Our package will automatically update to new version for major changes.
  - **^version** “**Compatible with version**”, will update you to all future minor/patch versions, without incrementing the major version. `^1.2.3` will use releases from `1.2.3` to `<2.0.0`. Our package will automatically update to new version for minor changes.

If we do not use anything means I just want this version only, do not update

```
"devDependencies": {  
  "parcel": "2.8.2"  
}
```

## What is package-lock.json?

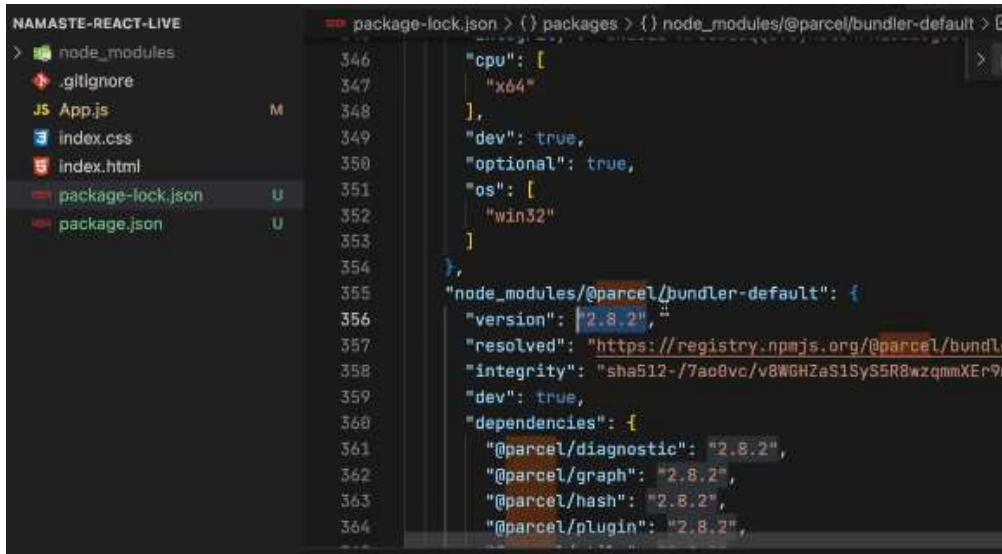
## package.json

It records important metadata about the project.

## package.lock.json

- It allows future devs to install the same dependencies in the project.
- It records the exact version of every installed dependency, including its sub-dependencies and their versions.

We see our package-lock.json and we get exact version of our Parcel.



```
NAMASTE-REACT-LIVE
> node_modules
  346  "cpu": [
  347    "x64"
  348  ],
  349  "dev": true,
  350  "optional": true,
  351  "os": [
  352    "win32"
  353  ],
  354  },
  355  "node_modules/@parcel/bundler-default": {
  356    "version": "2.8.2",
  357    "resolved": "https://registry.npmjs.org/@parcel/bundler-default/-/bundler-default-2.8.2.tgz",
  358    "integrity": "sha512-7ao0vc/v8WGHzaS1SyS5R8wzqmmXEr9mhII82cbLQ4LA2WUtKsYcvZ2gjJuAAN1CHC66xqwYjIJScQOk/QXg=="
  359    "dev": true,
  360    "dependencies": {
  361      "@parcel/diagnostic": "2.8.2",
  362      "@parcel/graph": "2.8.2",
  363      "@parcel/hash": "2.8.2",
  364      "@parcel/plugin": "2.8.2",
  365    }
  366  }
```

It is working on my local but breaking in production?

Let say we use ^ in package.json

So package updates itself automatically and it breaks in production so package-lock.json has exact version, it locks the version.

It takes snapshot of exact version we have in our project.

## Important things about package-lock

Never put package-lock.json in git ignore

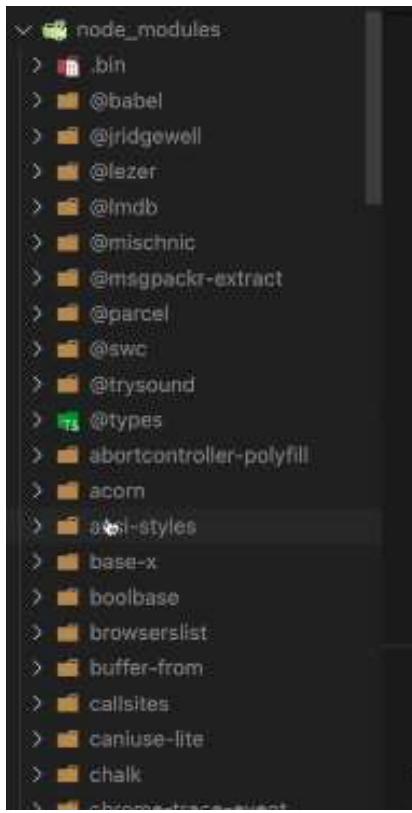
Always put it in your git with your project.

We need to push our package-lock to server and we cannot do it directly so we push it in git and server fetch it from git so we need to push package-lock in git.

It maintains a hash of version of package also, keeps track that hash is same in local and production. It maintains the integrity.

```
"node_modules/@parcel/bundler-default": {
  "version": "2.8.2",
  "resolved": "https://registry.npmjs.org/@parcel/bundler-default/-/bundler-default-2.8.2.tgz",
  "integrity": "sha512-7ao0vc/v8WGHzaS1SyS5R8wzqmmXEr9mhII82cbLQ4LA2WUtKsYcvZ2gjJuAAN1CHC66xqwYjIJScQOk/QXg=="}
```

When we installed parcel, node modules also got created.



Whatever we install, it gets installed in node modules.

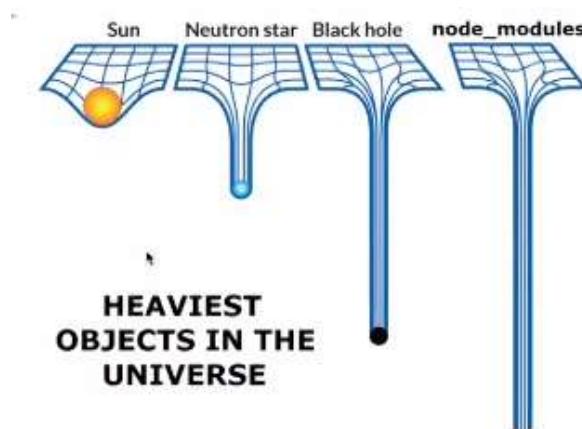
So node modules is like database of npm.

Parcel is there in node modules.

Parcel uses lot of things to optimise our app.

Parcel also has many dependencies to do this optimisation, they are also inside node modules so our node modules becomes huge.

We have something like browserlist in node modules which help our app work nicely in older version of browsers, sameway there are many packages to optimise our app.



Should we add our node\_modules to our git??

It is foolish to put node modules in git repo, it is the heaviest thing in your project. It 1GB large.

Our package-lock.json has sufficient material to make another node modules

We can generate another node modules using package-lock.json so no need to push node\_modules.

Just push package-lock.json

We will generate our node modules in server using package-lock.json and it will make sure our app does not break due to node modules not being pushed inside git.

We are using CDN to get react in our project

This is not the good way.

Currently we are using react 18, what if it gets updated to react 19?

CDN is on different server

What if we create our own server and fetch react from it?

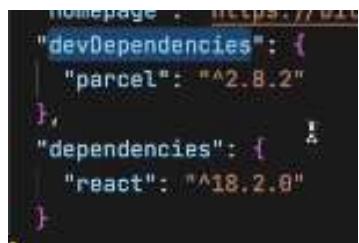
That is why we do not use CDN and create our own react app

How to install react in our project?



```
% npm install react
```

Now we do not use -D as we need react globally and we get react in our dependencies in package.json



```
homepage : https://vit...
```

```
"devDependencies": {  
  "parcel": "^2.8.2"  
},  
"dependencies": {  
  "react": "^18.2.0"  
},
```

This is the right way to install react in our project.

Now we install ReactDOM

"npm i" is same as "npm install"



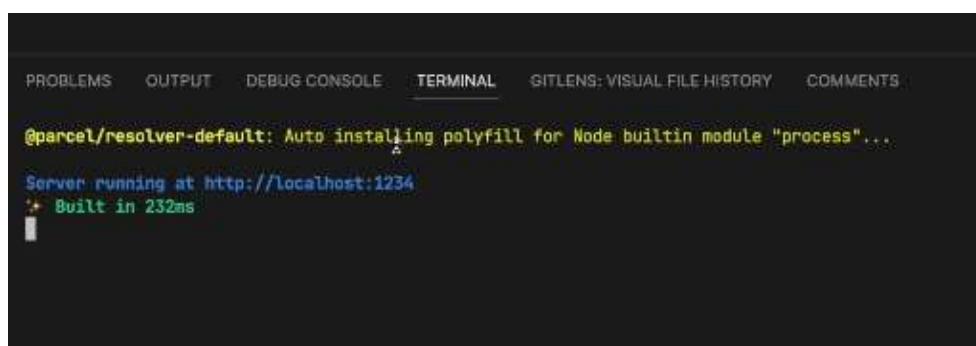
```
$ npm i react-dom
```

npx means execute using npm

Now we use "npx parcel index.html"

Where index.html is our entry-point

It starts a mini-server for us.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS: VISUAL FILE HISTORY COMMENTS  
  
parcel/resolver-default: Auto installing polyfill for Node builtin module "process"...
Server running at http://localhost:1234
+ Built in 232ms
```

Parcel gave a mini-server to us.

Now our app runs on localHost:1234



Now our console gives this error



Because we have use React.createElement() etc in our app.js but we have not imported React in our js file

How to import React??

We use a keyword "import"

We did not get this error before because we were using a CDN earlier.

We want to use react and ReactDOM from our node modules now so we use "import"

```
JS App.js > ...
...
1 | import React from "react";
2 | import ReactDOM from "react-dom";
3 |
4 | const heading = React.createElement(
5 |   "h1",
6 |   {
7 |     id: "title",
8 |   },
9 |   "Heading 1"
10);
11
12 const heading2 = React.createElement(
13   "h2",
14   {
15     id: "title",
16   },
17   "Heading 2"
18);
```

Now our code gives below error

```

✖ @parcel/transformer-js: Browser scripts cannot have imports or exports.

/Users/akshaysaini/Desktop/namaste-react-live/App.js:1:1
> 1 | import React from "react";
|   ^^^^^^
> 2 | import ReactDOM from "react-dom";
|   ^^^^^^
/Users/akshaysaini/Desktop/namaste-react-live/index.html:13:5
> 12 |     <script src="App.js"></script>
|   ^^^^^^
> 13 |   </body>
|   ^^^^^^ The environment was originally created here
> 14 | </html>
|   ^^^^^^
> 15 |

✖ Add the type="module" attribute to the <script> tag.
  Learn more

```

Elements Console Sources Network Performance Memory Application Security

top Filter

✖ Uncaught ReferenceError: React is not defined  
at App.js:1:12

✖ [parcel]: @parcel/transformer-js: Browser scripts cannot have imports or exports.  
/Users/akshaysaini/Desktop/namaste-react-live/App.js:1:1

```

> 1 | import React from "react";
|   ^^^^^^
> 2 | import ReactDOM from "react-dom";
|   ^^^^^^

```

/Users/akshaysaini/Desktop/namaste-react-live/index.html:13:5

```

> 12 |     <script src="App.js"></script>
|   ^^^^^^
> 13 |   </body>
|   ^^^^^^ The environment was originally created here
> 14 | </html>
|   ^^^^^^

```

Add the type="module" attribute to the <script> tag.

Never touch your node modules

Never update your package.json or node modules.

The error is coming because we have use <script src="app.js"> in index.html

And we use import in app.js

Browser does not understands import

So we need to tell browser that this is not a normal JS file, it's a module so what we do is

Specify it's a type = "module"

```

<div id="root">Not Rendered</div>
<script type="module" src="App.js"></script>
</body>]
</html>

```

Earlier it was like

import ReactDOM from "react-dom" but now they have updated it to

```

import React from "react";
import ReactDOM from "react-dom/client";

```

There is something known as **Hot Module Replacement (HMR)**

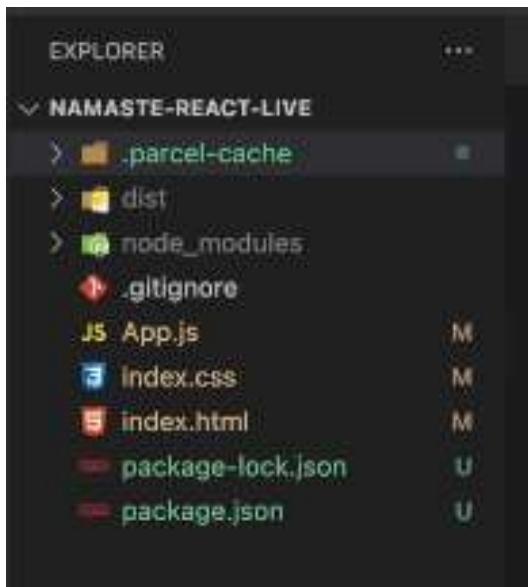
Means keep on updating the page on saving any changes

Parcel does this HMR, and saves all the changes in our HTML,CSS or JS.

How does Parcel does HMR?

There is something known as **file watcher algorithm** and parcel uses this algo which is written in **C++ internally**.

What is parcel\_cache and dist in file and folder structure?



Parcel need some space to do this HMR and other things

So it creates parcel\_cache and it contains all files which does all optimisation things for us in parcel

dist folder keeps the file minified for us

npx parcel index.html creates a development build and host it on our server.

How to tell parcel to make a production build?

We use "build" command and it minifies all our files, get it ready for production and push them inside dist folder.

```
npx parcel build index.html
```

We use "main: App.js" in our package.json which tells entry point of our app as App.js

Which is not needed if we are using parcel so we delete this command

```
package.json -- namaste-react-live

EXPLORER          ... 5 Index.html M  package.json U X JS App.js M ●  Index.css M
✓ NAMASTE-REACT-LIVE
> .parcel-cache
> node_modules
> .gitignore
JS App.js
TS index.css
TS index.html
package-lock.json
package.json

package.json > ...
1  {
2    "name": "namaste-react-live",
3    "version": "1.0.0",
4    "description": "This is a live course",
5    "main": "App.js",
6    "scripts": {
7      "test": "jest"
8    },
9    "repository": {
10      "type": "git",
11      "url": "git+ssh://git@bitbucket.org/namastedev/namaste-react-live"
12    },
13    "author": "Akshay Saini",
14    "license": "ISC",
15    "bugs": {
16      "url": "https://bitbucket.org/namastedev/namaste-react-live"
17    },
18    "homepage": "https://bitbucket.org/namastedev/namaste-react-live"
19    "devDependencies": {}
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    GITLENS: VISUAL FILE HISTORY

```
4 |   "description": "This is a live course",
5 |   "main": "App.js",
6 |   ^^^^^^^^^ Did you mean "App.html"?
7 |   "scripts": {
8 |     "test": "jest"
```

We give entry point while installing parcel only

npx parcel build index.html

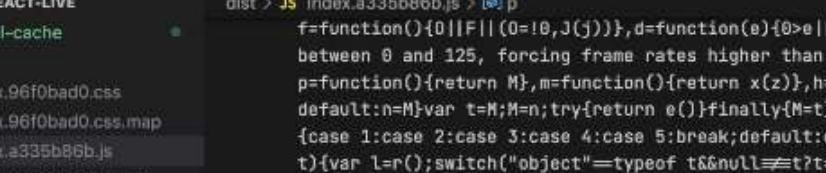
And it creates 3 files which are below:

```
● akshaysaini@Akshays-MacBook-Pro namaste-react-live % npx parcel build index.html
+ Built in 1.51s

dist/index.html          413 B    237ms
dist/index.96f0bad0.css   98 B     19ms
dist/index.a335b86b.js   140.32 KB   1.83s
○ akshaysaini@Akshays-MacBook-Pro namaste-react-live %
```

These 3 files are bundle of our app which will go to production build and will go to dist folder.

These 3 files has our code for HTML, CSS, JS



The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure with files like .parcel-cache, dist, index.html, node\_modules, .gitignore, App.js, index.css, index.html, package-lock.json, and package.json.
- index.html M**, **package.json U**, **JS index.a335b86b.js X**, **JS App.js**: These are the currently open tabs in the editor.
- index.a335b86b.js**: The code editor displays the content of this file, which is a heavily minified JavaScript file. It includes comments explaining its purpose as a performance optimization for frame rates between 0 and 125.

Parcel minified everything for us.  
It bundled everything for us.  
Removed all console logs and cleaned the code

What takes lot of time to load in browser?

No, HTML

No, CSS

No, JS

node modules is in server so it is not the answer.

**Images, media takes most time to load in browser**

Parcel manages dev and production build and is superfast

Parcel does image optimisation also, it minifies the images also if they are in our project.

Parcel also does **Caching while development for us**, What is caching?

We see our build took 515ms initially for dev build.

```
o akshaysaini@Akshays-MacBook-Pro namaste-react-live %
o akshaysaini@Akshays-MacBook-Pro namaste-react-live % npx parcel index.html
  Server running at http://localhost:1234
    Built in 515ms
```

Now if we refresh our page we see time is 5ms, 4ms, 9ms

The time is reducing means parcel is using something known as caching.

Parcel also compresses our files, renames our variables. This process also called compression.

Parcel checks your project's compatibility with older version of browsers.

Sometimes we need our build to test in https also. Parcel gives us that functionality to enable https in our localserver

```
$ npx parcel index.html --https
```

It took lot of build time this time as we are doing this change for first time, if we do it again It takes lesser time due to caching

```
o akshaysaini@Akshays-MacBook-Pro namaste-react-live % npx parcel index.html --https
  Server running at https://localhost:1234
    Built in 18ms
```

If we run 2 project at a time, Port number of server will also change automatically, it is also managed by Parcel.

Should we push parcel-cache in git?

We should not push our parcel-cache in git. And we should put it in our .gitignore

The screenshot shows a file explorer interface with two panes. The left pane, titled 'EXPLORER', lists files and folders under a project named 'NAMASTE-REACT-LIVE'. It includes '.parcel-cache', 'dist', 'node\_modules', '.gitignore', 'App.js', 'index.css', 'index.html', 'package-lock.json', and 'package.json'. The right pane shows the contents of the '.gitignore' file, which contains three lines: '51', '52', and '53 .parcel-cache'. The file is marked as modified ('M').

Because anything which can be auto-generated on server will be put inside git-ignore. We do not push it to git.

So we do not push parcel-cache and dist in git as we can generate them later

Parcel also uses something known as consistent hashing algorithms to cache things up and bundling up.

Parcel is a zero-config bundler.

**If React is Narendra Modi then Parcel is Amit Shah**



If BJP wins, it's not only because of Namo and Amit Shah

There are some other ministers which are more or less important.

React (Modi) need Parcel (Amit Shah) and parcel need many dependencies (other ministers).

**What are benefits of using Parcel?**

```

* HMR - Hot Module Reloading
* File Watcher algorithm - C++
* BUNDLING
* MINIFY
* Cleaning our Code
* Dev and Production Build
* Super Fast build algorithm
* Image Optimization
* Caching while development
* Compression
* Compatible with older version of browser
* HTTPS on dev
* port Number
* Consistent Hashing Algorithm
* Zero Config
*
/

```

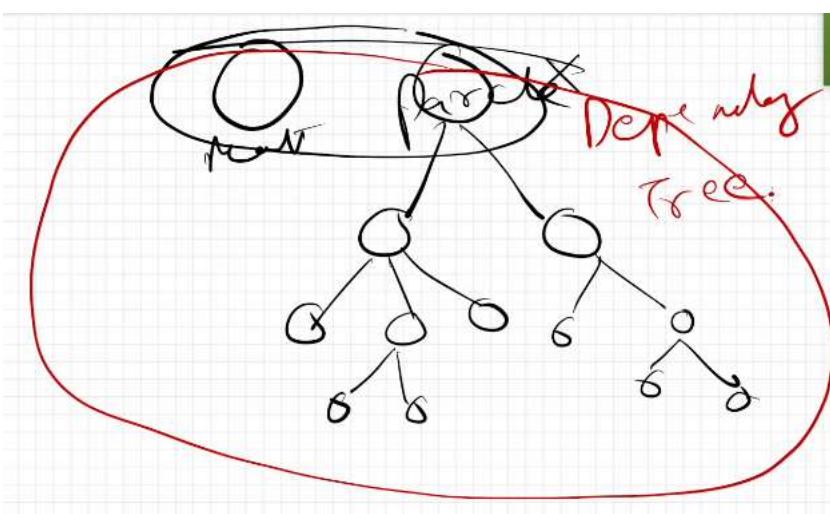
Parcel uses many dependencies in itself and all dependencies can have their dependencies so this is infinite loop and this process is called **Transitive Dependencies** (dependencies ki dependencies)

```

{
  "node_modules/@parcel/bundler-default": {
    "version": "2.8.2",
    "resolved": "https://registry.npmjs.org/@parcel/bundle...",
    "integrity": "sha512-7aoBvc/v8#6HZaS1SyS5R8wzqmmXEr9...",
    "dev": true,
    "dependencies": {
      "@parcel/diagnostic": "2.8.2",
      "@parcel/graph": "2.8.2",
      "@parcel/hash": "2.8.2",
      "@parcel/plugin": "2.8.2",
      "@parcel/utils": "2.8.2",
      "nullthrows": "^1.1.1"
    },
    "engines": {
      "node": "≥ 12.0.0",
      "parcel": "^2.8.2"
    },
    "funding": {
      "type": "opencollective",
      "url": "https://opencollective.com/parcel"
    }
  }
}

```

This structure of Transitive Dependencies is called **Dependency Tree**



## Read the docs (Be Curious)



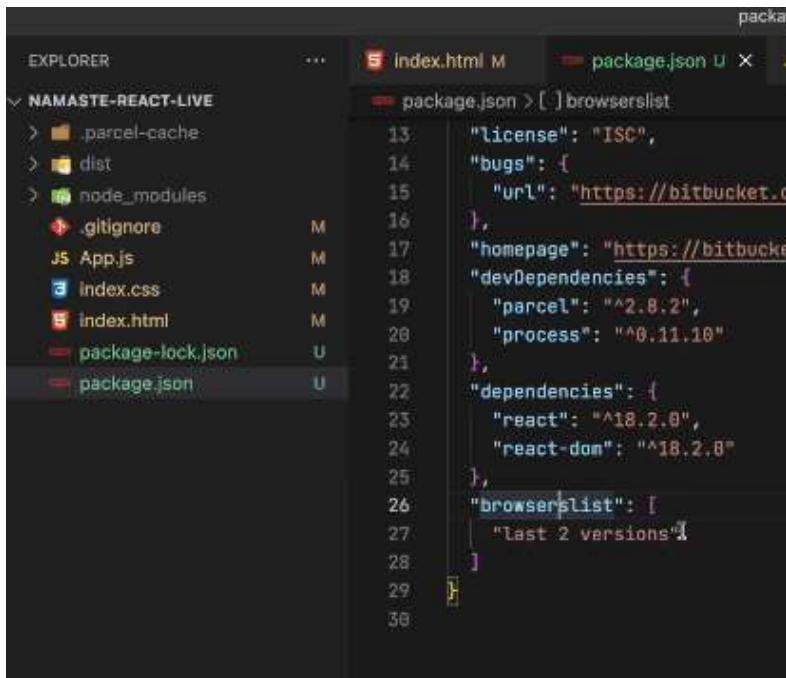
This is our create react app without using create-react-app

```
› NAMASTE-REACT-LIVE
  > 📁 .parcel-cache
  > 📁 dist
  > 📁 node_modules
  ⚡ .gitignore          M
  JS App.js             M
  CSS index.css         M
  HTML index.html       M
  JSON package-lock.json U
  JSON package.json      U
```

How to make your app, compatible with older version of browsers?  
We use **browserList** which is already used by parcel.

```
"browserslist": [
  "defaults and supports es6-module",
  "maintained node versions"
]
```

We put this code in our package.json  
browserList is feeded with a Array which has some configurations.  
If we write "last 2 versions" our app will run in last 2 versions of all browsers available



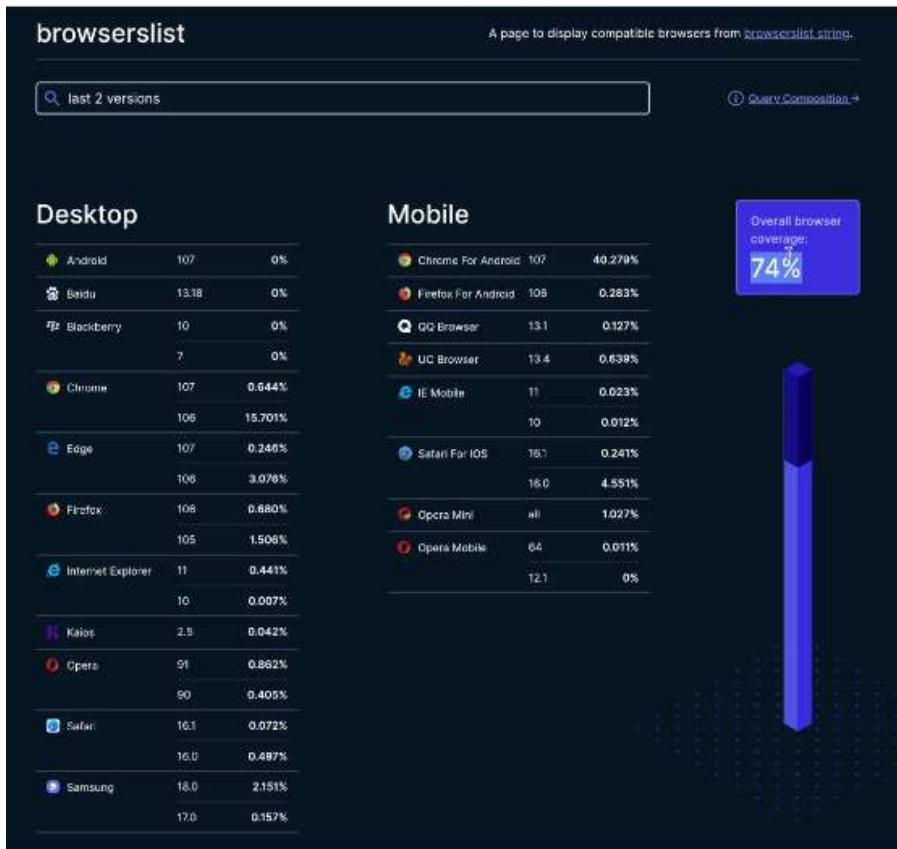
The screenshot shows the VS Code interface with the Explorer sidebar open, displaying a project named 'NAMASTE-REACT-LIVE'. The package.json file is selected and open in the editor. The code in the package.json file includes a 'browserslist' section with the value 'last 2 versions'.

```
package.json > [ ] browserslist
  13:   "license": "ISC",
  14:   "bugs": {
  15:     "url": "https://bitbucket.org/namaste/namaste-react-live/-/issues"
  16:   },
  17:   "homepage": "https://bitbucket.org/namaste/namaste-react-live",
  18:   "devDependencies": {
  19:     "parcel": "^2.8.2",
  20:     "process": "^0.11.10"
  21:   },
  22:   "dependencies": {
  23:     "react": "^18.2.0",
  24:     "react-dom": "^18.2.0"
  25:   },
  26:   "browserslist": [
  27:     "last 2 versions"
  28:   ]
  29:
  30:
  31:
  32:
  33:
  34:
  35:
  36:
```

If we want support for only last 2 chrome versions, we write



We can use browserList.dev website to see



## Full List

You can specify the browser and Node.js versions by queries (case insensitive):

- `defaults` : Browserslist's default browsers (`> 0.5%`, `last 2 versions`, Firefox ESR, not dead).
- By usage statistics:
  - `> 5%` : browsers versions selected by global usage statistics. `>=`, `<` and `<=` work too.
  - `> 5% in US` : uses USA usage statistics. It accepts [two-letter country code](#).
  - `> 5% in alt-US` : uses Asia region usage statistics. List of all region codes can be found at [caniuse-lite/data/regions](#).
  - `> 5% in my stats` : uses [custom usage data](#).
  - `> 5% in browserslist-config-mycompany.stats` : uses [custom usage data](#) from `browserslist-config-mycompany/browserslist-stats.json`.
  - `cover 99.5%` : most popular browsers that provide coverage.
  - `cover 99.5% in US` : same as above, with [two-letter country code](#).
  - `cover 99.5% in my stats` : uses [custom usage data](#).
- Last versions:
  - `last 2 versions` : the last 2 versions for each browser.
  - `last 2 Chrome versions` : the last 2 versions of Chrome browser.
  - `last 2 major versions` or `last 2 iOS major versions` : all minor/patch releases of last 2 major versions.
- `dead` : browsers without official support or updates for 24 months. Right now it is `IE 11`, `IE_Mob 11`, `BlackBerry 10`, `BlackBerry 7`, `Samsung 4`, `OperaMobile 12.1` and all versions of `Baidu`.
- Node.js versions:
  - `node 10` and `node 10.4` : selects latest Node.js `10.x.x` or `10.4.x` release.
  - `last 2 node versions` : select 2 latest Node.js releases.
  - `last 2 node major versions` : select 2 latest major-version Node.js releases.

## Different Type of script tags in HTML

The **HTML <script> type Attribute** is used to specify the MIME (**MIME (Multipurpose Internet Mail Extensions)** is an extension of the original Simple Mail Transport Protocol (**SMTP**) email protocol. It lets users exchange different kinds of data files, including audio, video, images and application programs, over email.) type of script and identify the content of the Tag. It has a Default value which is “text/javascript”.

**Syntax:**

```
<script type="media_type">
```

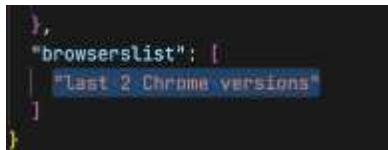
**Attribute Values:** It contains a single value i.e media\_type which specifies the MIME type of script.

**Common “media\_type” values are:**

- text/javascript (this is default)
- text/ecmascript
- application/ecmascript
- application/javascript

# Session 4

Let's talk about JSX, Babel



It does not mean our app will not work with other browsers, it means that It will definitely work on chrome + it will work on other browsers also.

When browsers update themselves, there are some changes

Some older browsers still not support ES6 so we need to convert our code to older browser compatible means we create a polyfill for newer things which our browser does not support currently.

Polyfill = If our browser does not know about map() or promise() then polyfill means we can create our own map() which does exact same work as map() does.

What converts our newer code to older code for older browsers?

Babel does that

If there is something new in the market, there must be some older replacement of it which is compatible in older versions of browsers so babel just convert that newer thing to that old thing so that our browser supports it.

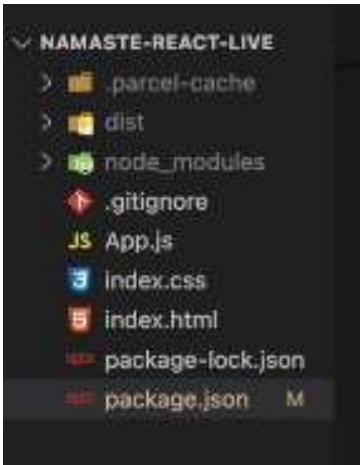
Babel has something like browserList for browser version.

We need not to write polyfill, babel does it for us.

Babel is just a piece of JS code which takes some modern code and spit older code

How did gitignore come from in our folder structure?

We did "git init" to initialise an empty repo, it made gitignore automatically.



### What is Tree Shaking?

Parcel does something known as Tree Shaking which means removing unwanted code.

#### How to know which code is unwanted?

Suppose we are using a library in our project which offers 20 functions. We install that library and whole piece of code comes in our project.

Now, say we just want to use only 2 functions out of that 20

Then those 18 are unwanted functions and that code for these 18 functions is just unwanted.

Parcel detects this un-used piece of code and ignores all of it, this optimises our app and this is called Tree Shaking.

#### Does web-pack does not have these functions?

All bundlers have almost all these things but they are different in small ways. Most of them have the same basic features but there are some differences.

create-react-app uses web-pack + babel

Generally we build a script inside package.json to simplify our work and help us write less code to start our app like currently we write "npx parcel index.html"

So we make changes to this script in our package.json

```
  "scripts": {
    "start": "parcel index.html",
    "test": "jest"
  },
```

Now our command changes to: `npm run start`

```
  $ npm run start
```

We have a build command also: `npx parcel build index.html`

We change it to something like: "**build**": "`parcel build index.html`" now we just write `npm run build`

```

"scripts": [
  "start": "parcel index.html",
  "build": "parcel build index.html",
  "test": "jest"
],

```

In commands **npm** is used when we want to install something inside our project whereas **npx** is used when we want execute something in our project.

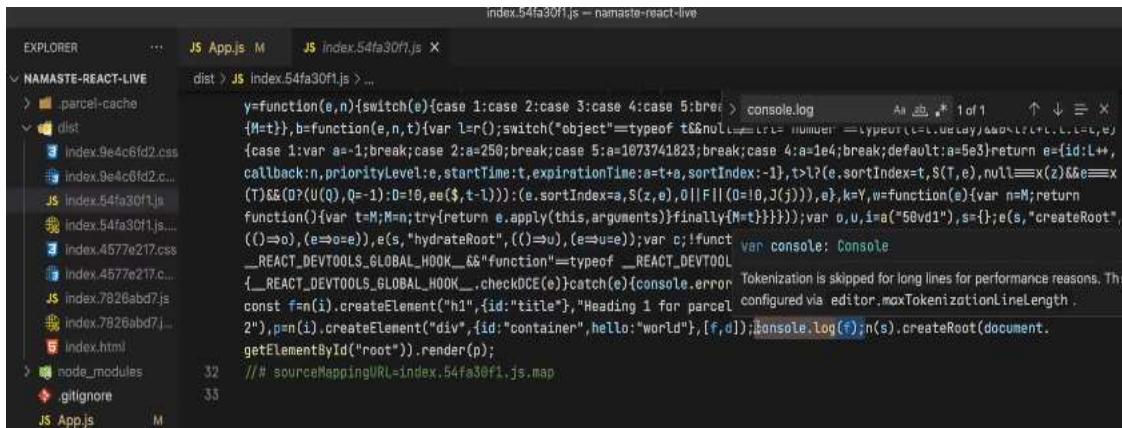
**NPM is a package manager used to install, delete, and update Javascript packages on your machine.** NPM is a package executer, and it is used to execute javascript packages directly, without installing them.

**npx = npm run**

That is why we write **npm run start** which is = **npx parcel index.html**

Instead of **npm run start** we can also write **npm start**. Its same

Parcel even removes console.logs while doing production build.  
But we see in our build.js that it did not



```

index.54fa30f1.js - nameaste-react-live
EXPLORER JS App.js M JS index.54fa30f1.js X
NAMASTE-REACT-LIVE dist > JS index.54fa30f1.js > ...
  > parcel-cache
  > dist
    > Index.9e4c6fd2.css
    > Index.9e4c6fd2.c...
    JS index.54fa30f1.js
    > Index.54fa30f1.js...
    JS index.4577e217.css
    > Index.4577e217.c...
    JS index.7826abd7.js
    > Index.7826abd7.j...
    > index.html
    > node_modules
      > gitignore
JS App.js M
  32 // # sourceMappingURL=index.54fa30f1.js.map
  33

```

Actually what happens is, it does not remove console logs automatically, we need to configure our project in such a way that it removes console.logs

There is a package which does this  
Babel plugin transform remove console  
This removes our console logs

## babel-plugin-transform-remove-console

Edit

### Example

In

```
JavaScript
console.log("foo");
console.error("bar");
```

Copy

Out

```
JavaScript
```

Copy

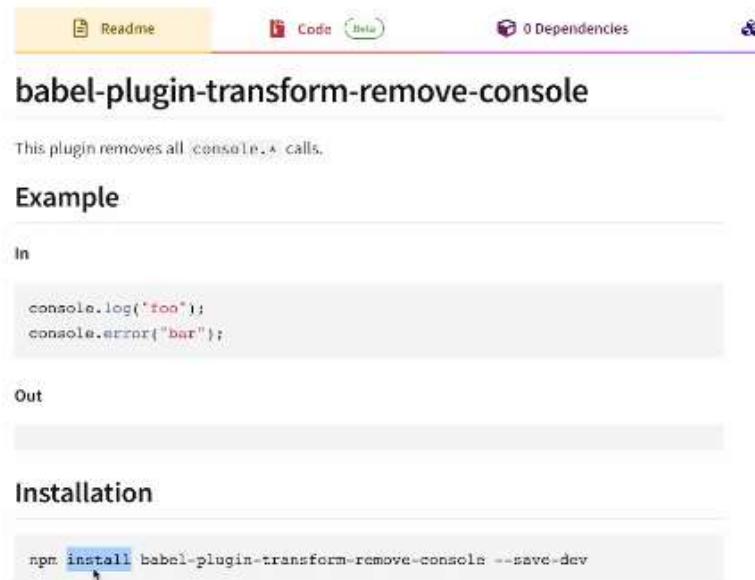
### Installation

Shell:

```
npm install babel-plugin-transform-remove-console --save-dev
```

Copy

Now we will install this plugin in our project, we can also go to npm website



This plugin removes all `console.*` calls.

### Example

In

```
console.log("foo");
console.error("bar");
```

Out

### Installation

```
npm install babel-plugin-transform-remove-console --save-dev
```

Parcel does not remove console logs for us, we need plugin to do that

We do `--save-dev` if we need it as our dev-dependency

Its version information goes inside `package.json` and its code goes inside `node modules`.

It does not work now.

We need to first configure it also

Go to that plugin website and see its usage

## Installation

```
npm install babel-plugin-transform-remove-console --save-dev
```

### Usage

#### Via .babelrc (Recommended)

```
.babelrc
```

```
// without options
{
  "plugins": ["transform-remove-console"]
}
```

```
// with options
{
  "plugins": [ ["transform-remove-console", { "exclude": [ "error", "warn" ] } ] ]
}
```

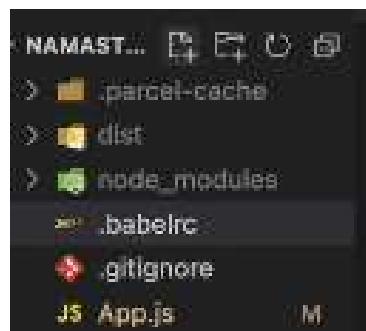
#### Via CLI

```
babel --plugins transform-remove-console script.js
```

#### Via Node API

```
require('@babel/core').transform('code', {
  plugins: ['transform-remove-console']
});
```

.babel.rc is a configuration of babel. We create a babel.rc



We pick usage code from documentation and paste it inside babel.rc.

Exclude error and warn means exclude error and warning message from console of our production app.

```
JS App.js M └─.babelrc U X
└─.babelrc > [ ] plugins > [ ] 0 > {} 1 > [ ] exclude > □ 0

1   {
2     "plugins": [ ["transform-remove-console",
3       { "exclude": [ "error", "warn" ] } ] ]
```

Now we make **npm run build** and we do not get any console in our js file in dist.

## Let's code

Suppose we have multiple siblings in a element,

We are passing an array as our children,

```
const container = React.createElement(  
  "div",  
  {  
    id: "container",  
    hello: "world",  
  },  
  [heading, heading2]           You, 16 seconds ago  
);
```

We get a warning in console for 'key'

We need to give keys as props to our children inside {} of our createElement()

Key can be anything which uniquely identifies the element.

```
const heading = React.createElement(  
  "h1",  
  {  
    id: "title",  
    key: "h1",           You, 16 seconds ago  
  },  
  "Heading 1 for parcel"  
);  
  
const heading2 = React.createElement(  
  "h2",  
  {  
    id: "title",  
    key: "h2",           You, 16 seconds ago  
  },  
  "Heading 2"  
);
```

### Why do we need a key?

Suppose we have the following structure,

When comparing two React DOM elements of the same type, React looks at the attributes of both, keeps the same underlying DOM node, and only updates the changed attributes. For example:

```
<div className="before" title="stuff" />  
  
<div className="after" title="stuff" />
```

By comparing these two elements, React knows to only modify the className on the underlying DOM node.

When updating style, React also knows to update only the properties that changed. For example:

```
<div style={{color: 'red', fontWeight: 'bold'}} />  
  
<div style={{color: 'green', fontWeight: 'bold'}} />
```

When converting between these two elements, React knows to only modify the color style, not the fontWeight.

After handling the DOM node, React then recurses on the children

## Recurse On Children

By default, when recursing on the children of a DOM node, React just iterates over both lists of children at the same time and generates a mutation whenever there's a difference.

For example, when adding an element at the end of the children, converting between these two trees works well:

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

React will match the two `<li>first</li>` trees, match the two `<li>second</li>` trees, and then insert the `<li>third</li>` tree.

If you implement it naively, inserting an element at the beginning has worse performance. For example, converting between these two trees works poorly:

```
<ul>
  <li>Duke</li>
  <li>Villanova</li>
</ul>

<ul>
  <li>Connecticut</li>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

React will mutate every child instead of realizing it can keep the `<li>Duke</li>` and `<li>Villanova</li>` subtrees intact. This inefficiency can be a problem

**Matlab new element ko add krne ke liye react ab poore older DOM ko destroy krega and new DOM banayega with the structure connectuct -> Duke -> Villanova**

**Which can reduce the performance for us and React has to do lot of work in this. So something introduced called as "keys"**

## Keys

**In order to solve this issue, React supports a key attribute. When children have keys, React uses the key to match children in the original tree with children in the subsequent tree. For**

example, adding a key to our inefficient example above can make the tree conversion efficient:

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>

<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

Now React knows that the element with key '2014' is the new one, and the elements with the keys '2015' and '2016' have just moved.

In practice, finding a key is usually not hard. The element you are going to display may already have a unique ID, so the key can just come from your data:

```
<li key={item.id}>{item.name}</li>
```

As a last resort, you can pass an item's index in the array as a key. This can work well if the items are never reordered, but reorders will be slow.

We should never have same keys for 2 children.

### How does React.createElement() works?

createElement is there in code of react which is there in node modules.

React.createElement() gives us objects.

If we console log our heading

```
const heading = React.createElement(
  "h1",
  {
    id: "title",
    key: "h1",
  },
  "Heading 1 for parcel"
);
console.log(heading);

const heading2 = React.createElement(
  "h2",
  {
    id: "title",
    key: "h2",
  },
  "Heading 2"
);
```

We see an object with props as id and key

The screenshot shows the Chrome DevTools Elements tab. It displays the DOM structure of a React application. At the top, there's a navigation bar with tabs like Elements, Console, Sources, Network, Performance, Memory, Application, and Security. Below the navigation bar, the main area shows the DOM tree. A specific element, an `h1`, is selected and expanded. Its properties are listed: `key: "h1"`, `ref: null`, and `props: {id: 'title', children: 'Heading 1 for parcel'}`. Other properties shown include `_owner: null`, `_store: {validated: false}`, `_self: null`, and `_source: null`. A note at the bottom of the expanded element says `Uncaught (in promise) Error: A listener indicated an asynchronous response by returning true, but`. There's also a small note at the top left of the expanded element: `↳ If you see this message, it means the component has been updated.`

```
<h1>
<body>
  <div id="root">Not Rendered</div>

  <div class="header">          You, 3 seconds ago +
    <h1>Namaste React</h1>
    <ul>
      <li>About Us</li>
      <li>Support</li>
      <li>Home</li>
    </ul>
  </div>

  <script type="module" src="App.js"></script>
</body>
```

createElement gives us an object which is then converted into HTML code and put up in the DOM.

If we have to insert the below structure inside our DOM

The screenshot shows the browser's developer tools. It displays the rendered DOM structure. The page contains a `<div id="root">` element with the text "Not Rendered". Inside this, there's a `<div class="header">` element. This header contains an `<h1>Namaste React</h1>` element and a `<ul>` element. The `<ul>` element contains three `<li>` elements with the text "About Us", "Support", and "Home" respectively. Below this structure is a `<script type="module" src="App.js"></script>` tag.

```
<head>
<body>
  <div id="root">Not Rendered</div>

  <div class="header">          You, 3 seconds ago +
    <h1>Namaste React</h1>
    <ul>
      <li>About Us</li>
      <li>Support</li>
      <li>Home</li>
    </ul>
  </div>

  <script type="module" src="App.js"></script>
</body>
```

How to do this using React?

We will create a div first and div should have 2 children h1 and ul

Instead of heading 1, our h1 will come

The screenshot shows a code editor with a single line of code. It uses the `React.createElement` function to create a `div` element with the id "container" and the hello world text. It then creates an `h1` element with the id "title" and key "h1", containing the text "Heading 1 for parcel". Finally, it creates a `ul` element with the id "heading2".

```
const container = React.createElement(
  "div",
  {
    id: "container",
    hello: "world",
  },
  [React.createElement(
    "h1",
    {
      id: "title",
      key: "h1",
    },
    "Heading 1 for parcel"
  ), heading2]
);
```

Instead of heading 2, ul will come and inside ul, li will come with About us, support, home  
So we put all these inside array for ul

```
  heading = () => {
    ,
    React.createElement("ul", {}, [React.createElement(
      "li", "You, 2 seconds ago + Uncommitted changes
      ,
      "About Us"
    ), React.createElement("ul", {}, [React.createElement(
      "li", "I
      ,
      "About Us"
    ), React.createElement("ul", {}, [React.createElement(
      "li", "You
      ,
      "About Us"
    )])])
  }
}
```

This will become a huge mess if we do it this way.

React.createElement() for big structure becomes huge mess  
So there should definitely be a good and clean way to do this.

So generally instead of React.createElement()  
We use something known as JSX (Javascript XML)

While writing React, the basic motive was to write HTML using JS in a efficient way  
So they first made React.createElement() API, But it made the code messy like we saw above.

Instead of writing React.createElement( ) everywhere, we import {createElement} from react and now we do not need to write React.createElement()  
Just writing createElement() does the job for us.

```
import { createElement } from "react";
import ReactDOM from "react-dom/client";

// React.createElement => Object => HTML(DOM)

const heading = createElement(
  "div",
  {
    id: "title",
    key: "h2",
  }
)
```

We can also use "ce" instead of createElement() by doing

```
to
import { createElement as ce } from "react";
import ReactDOM from "react-dom/client";
```

But it did not do any big change, code still looks messy

So **JSX was introduced**

How to create h1 tag using JSX?

```
const heading = <h1>Namaste React</h1>;
```

```
const root = ReactDOM.createRoot(document.getElementById("root"));
```

```
//passing a react element inside the root
```

```
//async defer
```

```
root.render(heading);
```

We can give id and key like:

```
const heading2 = [
```

```
  <h1 id="title" key="h2">..
```

```
  | Namaste React
```

```
  </h1>
```

```
];
```

If we are writing it in multiple line, we need to use **parenthesis ( )**

```
|
```

```
const heading2 = (
```

```
  <h1 id="title" key="h2">
```

```
    Namaste React
```

```
  </h1>
```

```
);
```

## Is JSX, HTML inside JS?

**JSX allows us to write HTML elements in JavaScript** and place them in the DOM without any createElement() and/or appendChild() methods. JSX converts HTML tags into react elements. **You are not required to use JSX, but JSX makes it easier to write React applications.**

JSX is not HTML, it is HTML like syntax, fancy way to write HTML inside JS but it is not HTML inside JS.

Now our final code, looks like:

```

import { createElement as ce } from "react";
import ReactDOM from "react-dom/client";

// React.createElement => Object => HTML(DOM)

const heading = ce(
  "h1",
  {
    id: "title",
    key: "h2",
  },
  "Namaste React"
);

// JSX ???

const heading2 = [
  <h1 id="title" key="h2"> You, 3 minutes ago + Uncommitted changes
  | Namaste React
  </h1>
];

const root = ReactDOM.createRoot(document.getElementById("root"));

//passing a react element inside the root

//async defer
root.render(heading);

```

### There are certain major differences between HTML and JSX

In JSX, we write in camel-case => tab-index in HTML becomes tabIndex in JSX  
 class in HTML becomes className in JSX

In HTML almost all tags have an opening and a closing tag except probably a few like `<br/>`

In JSX, however, any element can be written as a self-closing tag, for example: `<div/>`

Example:

```
const string = <img
src={user.avatarUrl} />;
```

HTML elements have attributes.

e.g., `maxlength` in `<input maxlength="16" />`

JSX elements have props.

e.g., `maxLength` in `<input maxLength="16" />`

In HTML, multiple elements can be returned.

For example:

```
<ul>
<li>unordered list
<ol>
<li>ordered list</li>
<li>ordered list</li>
<li>ordered list</li>
</ol>
</li>
<li>unordered list</li>
<li>unordered list</li>
```

Nested JSX must return one element, which we'll call a parent element that wraps all other levels of nested elements:

```
<div>
<p>pink</p>
<p>yellow</p>
<p>green</p>
</div>
```

Without the wrapper element, JSX won't transpile. In React, we can render JSX directly into HTML DOM

```
</ul>
```

using React rendering API, aka *ReactDOM*. The formula for rendering React elements seems like this:

*ReactDOM.render(componentToRender, targetNode)*

*ReactDOM.render()* must be called after the JSX elements declarations.

How is below code executed by JSX?

```
// JSX ??  
  
const heading2 = (  
  <h1 id="title" key="h2">  
    Namaste React  
  </h1>  
)
```

Our browser does not understand this code



But Babel understands it, Babel is a JS compiler. Its another JS library which takes some piece of code and returns some code.

Babel reads our code line by line and make an **Abstract Syntax tree (AST)**

What does JSX do is, JSX uses `React.createElement()` behind the scenes and it gets converted to object which is then fitted to DOM.

Who is converting JSX to `React.createElement()` ?

Babel, does that

## Advantages of JSX

1. JSX in React.js **makes code easy to read and write**
2. One of the advantages of JSX is that **React creates a virtual DOM (a virtual representation of the page) to track changes and updates**. Instead of rewriting the entire HTML, React modifies the DOM of the page whenever the information is updated. This is one of the main issues React was created to solve.

Write code such that it is easily readable by humans.

Babel comes along with parcel

We have 2 package-lock.json

1 is we have created and other is inside node\_modules which keeps track of all packages and libraries inside node modules (Transitive Dependencies).

JSX is not imported as its not a package. JSX is a syntax

Below code is known as JSX expression

```
const heading2 = () => {
  <h1 id="title" key="h2">
    Namaste React
  </h1>
};
```

## React Element

h1 tag is called as element inside React

From now on, we will not use React.createElement()

We will use JSX

```
import React from "react";
import ReactDOM from "react-dom/client";

// JSX => React.createElement => Object => HTML(DOM)

const heading2 = () => {
  <h1 id="title" key="h2">
    Namaste React
  </h1>
};

const root = ReactDOM.createRoot(document.getElementById("root"));

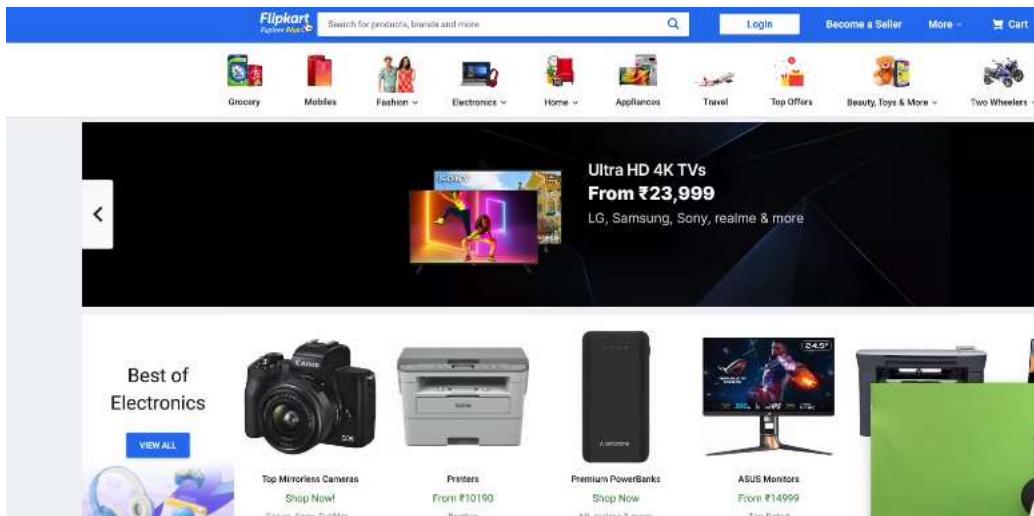
//passing a react element inside the root

//async defer
root.render(heading);
```

## What are components?

Everything is component in react.

Inside flipkart.com



Navbar carousel, products block everything is a component.

There are **2 types** of components in react

1. **Functional component (New way of writing code)**
2. **Class Component (OLD way of writing code)**

We will mostly use Functional Component

## Functional Components

They are nothing but a function and is hoisted same as a normal function in JS.

```
// React Component
// - Functional - NEW - I'll use this most of the time

const HeaderComponent = () => {
```

It is JS function which returns some piece react element or a component or JSX.

```
// React Component
// - Functional - NEW - I'll use this most of the time

const HeaderComponent = () => {
  return <h1>Namaste React functional component</h1>;
};
```

### Note:

**Functional component or for any component, Name starts with Capital Letter. It is not Mandatory but it is a normal convention, good practice to use this.**

**Functional component is a normal function which returns some element or a components or some piece of JSX.**

Suppose we want to build a big HTML structure

```
const HeaderComponent = () => {
  return <div><h1>Namaste React functional component</h1><h2>This is a h2 tag</h2></div>;
};
```

**When we write this complicated structure and our code goes in multiple lines, we have wrap it inside ( )**

```
const HeaderComponent = () => {
  return <div>
    <h1>Namaste React functional component</h1>
    <h2>This is a h2 tag</h2>
  </div>
};
```

Some people skip writing "return" to look cool

```
const HeaderComponent = () =>
  <div>
    <h1>Namaste React functional component</h1>
    <h2>This is a h2 tag</h2>
  </div>
};
```

Above code works already fine

Both component are same

```
const HeaderComponent = () => {
  return (
    <div>
      <h1>Namaste React functional component</h1>
      <h2>This is a h2 tag</h2>
    </div>
  );
};

const HeaderComponent2 = () => (
  <div>
    <h1>Namaste React functional component</h1>
    <h2>This is a h2 tag</h2>
  </div>
);
```

We can also write normal function instead of Arrow function

```
const HeaderComponent = function () {
  return (
    <div>
      <h1>Namaste React functional component</h1>
      <h2>This is a h2 tag</h2>
    </div>
  );
};
```

When we have to render our React element we write like this

```
const heading = (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);
```

For above element, we write like

```
//async defer
root.render(heading);
```

For Component how do we render it?

```
const HeaderComponent = () => {
  return (
    <div>
      <h1>Namaste React functional component</h1>
      <h2>This is a h2 tag</h2>
    </div>
  );
};

const root = ReactDOM.createRoot(document.getElementById("root"));

//passing a react element inside the root

//async defer
root.render(<HeaderComponent />);
```

Only difference between React element and React component is, react component is an Arrow function which return some JSX. whereas element is an simple JSX (not a function).

**A React element is an object representation of a DOM node. A component encapsulates a DOM tree.** Elements are immutable i,e once created cannot be changed. The state in a component is mutable.

## Element

An element is always gets returned by a component.

The element does not have any methods.

A React element is an object

## Component

A component can be functional or a class that optionally takes input and returns an element.

Each component has its life cycle methods.

A component encapsulates a DOM tree.

representation of a DOM node.

Elements are immutable i,e once created cannot be changed.

An element can be created using React.createElement( ) with type property.

We cannot use React Hooks with elements as elements are immutable.

Elements are light, stateless and hence it is faster.

The state in a component is mutable.

A component can be declared in different ways like it can be an element class with render() method or can be defined as a function.

React hooks can be used with only functional components

How to use React element inside React Component?

Let say we want our heading element to be used inside HeaderComponent

```
const heading = [
  <h1 id="title" key="h2">
    | Namaste React
  </h1>
];

const HeaderComponent = () => {
  return (
    <div>
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  );
};
```

We do something like: {header}

```
const HeaderComponent = () => {
  return (
    <div>
      {heading}
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  );
};
```

Output looks like:



## Namaste React

Namaste React functional component

This is a h2 tage

If our heading is a Component then we use it like this inside our component

```
const Title = () => (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);

const HeaderComponent = () => {
  return [
    <div>
      <Title /> You have 1 uncommitted changes
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tage</h2>
    </div>
  ];
}
```

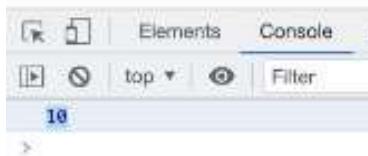
There is another way of writing this:

Just call it like a normal JS function

```
const HeaderComponent = () => {
  return [
    <div>
      {Title()}
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tage</h2>
    </div>
  ];
}
```

Inside JSX we can write anything in JS inside {} and it will render

```
var xyz = 10;
const HeaderComponent = () => {
  return (
    <div var console: Console>
      {console.log(xyz)} You, 8 seconds ago.
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tage</h2>
    </div>
  );
}
```



We can also do calculation there

```
var xyz = 10;
const HeaderComponent = () => {
  return (
    <div>
      {1 + 2}
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  );
};
```

So while writing JS in JSX we need to write it inside {}

That is why we call element like {title} inside Component as title is just a normal JS variable.

Let say we have an API call and we get some data

Suppose our API got hacked and our API gets some malicious code in our code  
And we render that data in component.

This attack is known as **Cross Site Scripting (XSS)**

```
const data = api.getData();

const HeaderComponent = () => {
  return (
    <div>
      {data}
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  );
};
```

Still JSX takes care of this, It makes sure our app is safe.

It **sanitization**

JSX expressions {} automatically take care of encoding HTML before rendering, which means even if u don't sanitise your input your webpage is XSS safe.

All I can find on Google around sanitizing HTML are packages that sanitize Markdown. So my intermediate solution is to convert my API-returned HTML to Markdown to-markdown, and then convert that back to HTML using marked, which includes a sanitizer.

We need to use 2 libraries to first mark-down code and then convert it back to HTML using marked which automatically sanitises it and keep us away from attacks.

```
import toMarkdown from 'to-markdown'
import marked from 'marked'

var sampleApiResponse = '<h1>I am a heading</h1><p>I am a
paragraph</p>';

var MyComponent = React.createClass({
  render: function(){
    var markup = this.props.htmlToRender
    var sanitize = function(htmlString){
      return marked(toMarkdown(html), {sanitize: true})
    }

    return (
      <div dangerouslySetInnerHTML={sanitize(markup)}></div>
    );
  }
}

ReactDOM.render(
  <MyComponent htmlToRender={sampleApiResponse} />,
  document.getElementById('react-wrapper')
)
```

## Component Composition

In React, we can make components more generic by accepting props, which are to React components what parameters are to functions. Component composition is **the name for passing components as props to other components, thus creating new components with other components**

```
const Title = () => (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);

const HeaderComponent = () => {
  return [
    <div>
      <Title /> You, 1 second ago + Uncommitted
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  ];
}
```

## React Reconciliation

When you use React, at a single point in time you can think of the **render() function**

as creating a tree of React elements. On the next state or props update, that render() function will return a different tree of React elements. React then needs to figure out how to efficiently update the UI to match the most recent tree.

There are some generic solutions to this algorithmic problem of generating the minimum number of operations to transform one tree into another. However, the algorithms have a complexity in the order of  $O(n^3)$  where  $n$  is the number of elements in the tree.

If we used this in React, displaying 1000 elements would require in the order of one billion comparisons. This is far too expensive. Instead, **React implements a heuristic  $O(n)$  algorithm based on two assumptions:**

1. Two elements of different types will produce different trees.
2. The developer can hint at which child elements may be stable across different renders with a **key prop**.

In practice, these assumptions are valid for almost all practical use cases.

## Session 5

Let us make an food website, food-villa

We saw we can build functional component, Its just a normal function which returns a piece of JSX.

React.createElement gives an object

How to write comment inside JSX?

Just like writing JS we use { } we also use { } for comments and comment like normal JS comment.

```
// Composing Components
const HeaderComponent = () => {
  return (
    <div>
      // This is a comment
      /* 
      * 
      * Multi Line comment */
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  )
}
```

Is JSX mandatory?

No, its not mandatory for React.

Not even Typescript, nor ES6, React is a library which does not force us to use JSX, Typescript etc etc

We can use React as an utility like if we have a huge website we can use React just only in Footer of our website.

Our whole React runs under "root" so if we want to use react in footer we just need to make our "footer" as our "root"

```
const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<HeaderComponent />);
```

We have seen 2 ways to **call a functional component in react** using <Title/> and {Title()}

But there is **another way, we can also call it like**

```
// Composing Components
const HeaderComponent = () => {
  return (
    <div>
      <Title></Title> You, 1 second ago + Undone
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tag</h2>
    </div>
  );
}
```

Generally we use **<Title/> syntax** which is also called self-closing tag.

**Before writing code, always do some planning, do some pen scratching**

Make a blueprint of how our app will look: Header, search bar, footer, body, cards for food items, information inside cards...



**Structure the Layout**

We make an functional component, AppLayout  
Which will return some JSX with header, body (Search bar, cards) , footer

```
const AppLayout = () => {
  return (
    /**
     * Header
     * - Logo
     * - Nav Items(Right Side)
     * - Cart
     *   You, 17 seconds ago
     * Body
     * - Search bar
     * - RestaurantList
     *   - RestaurantCard
     *     - Image
     *     - Name
     *     - Rating
     *     - Cusines
     * Footer
     * - links
     * - Copyright
    */
  );
}
```

Now we make Header Component first of all, all inside App.js intially

```
// Composing Components
const HeaderComponent = () => {
  return (
    <div>
      <Title />
      <div className="nav-items">
        <ul>
          <li>Home</li>
          <li>About</li>
          <li>Contact</li>
          <li>Cart</li>
        </ul>
      </div>
    </div>
  );
};
```

Now, Let us put some CSS here, add className to div for styling

```
// Composing Components
const HeaderComponent = () => {
  return (
    <div className="header">
      <Title />
      <div className="nav-items">
        <ul>
          <li>Home</li>
          <li>About</li>
          <li>Contact</li>
          <li>Cart</li>
        </ul>
      </div>
    </div>
  );
};

:
```

## CSS

JS App.js M index.css M X

```
index.css > .header
You, 32 seconds ago | 1 author (You)
1 .header {
2   display: flex;
3   justify-content: space-between;
4   margin: 10px;
5 }
6
7 .nav-items > ul {
8   list-style-type: none;
9   display: flex;
10
11
12 .nav-items > ul > li {
13   padding: 10px;
14 }
```

## Result



Let us use some Logo

How to put image?

Just use normal "img" tag with alt and className

We can also give anchor tag to it so that it goes to home page on clicking

```
const Title = () => {
  <a href="/"> You, 1 second ago - Uncommitted changes
    
  </a>
};
```

Let us Build our **Body**

**Any piece of JSX component can have maximum one parent**

```
const AppLayout = () => {
  return (
    <Header/>
    <Body/>
    <Footer/>
  );
};

const jsx = <h1>JSX</h1><h1>Second JSX</h1>
```

So we can either write it inside a div

```
const jsx = <div><h1>JSX</h1><h1>Second JSX</h1></div>
```

So we need to wrap everything inside a div

But this method adds up an extra div in our DOM below root

**We do not want to so there is something like `React.Fragment` which is a component exported by React**

```
// F;
// React.Fragment
const jsx = (
  <React.Fragment>
    <h1>JSX</h1>
    <h1>Second JSX</h1>
  </React.Fragment>
);

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render.jsx;
```

JSX can only have one parent

Suppose if we want to have 2 parents we can use a div

If we do not want a div in our DOM we can use `React.Fragment`

**React.Fragment is just like an empty tag**

Instead of writing `<React.Fragment><React.Fragment/>` we can also write `<></>` which is an short-hand for React Fragment.

```
const jsx = [
  <>
    <h1>JSX</h1>
    <h1>Second JSX</h1>
  </>
];
```

But if we want to give style, we cannot give style to a `<></>` so we will make it a div again and give styling like

#### (using Inline Style) in JSX

We can use style which takes an object i.e give style in key-value pair

But object is itself JS so we need to wrap it inside { }

```
<div style={{ background: "red", }}></div>
```

```
const styleObj = {
  backgroundColor: "red",
};

// React
const jsx = (
  <div style={styleObj}>
    <h1>JSX</h1>
    <h1>Second JSX</h1>
  </div>
);
```

```
const styleObj = {
  border: "1px solid red",
};

// Inline styling in React
const jsx = (
  <div
    style={styleObj}
  >
    <h1>JSX</h1>
    <h1>Second JSX</h1>
  </div>
);
```

Other way of styling is give it a className and do External CSS in index.css

So our Body looks like this currently

```
const Body = () => {
  return <h4>Body</h4>;
};

const Footer = () => {
  return <h4>Footer</h4>;
};

const AppLayout = () => {
  return (
    <React.Fragment>
      <Header />
      <Body />
      <Footer />
    </React.Fragment>
  );
};
```

Where will our data come from?

Right now, we will use some hard coded data after that we will also use an API

Let us make a Restro Card, we need many Restro cards

How should our resto card look like?

Our card should have an image, name of restro, Food labels/tags, Star ratings,

```
const RestrauntCard = () => {
  return (
    <div className="card">
      
      <h2>Burger King</h2>
      <h3>Burgers, American</h3>
      <h4>4.2 stars</h4>
      <small>You, 1 second ago - Uncommitted changes</small>
    </div>
  );
};
```

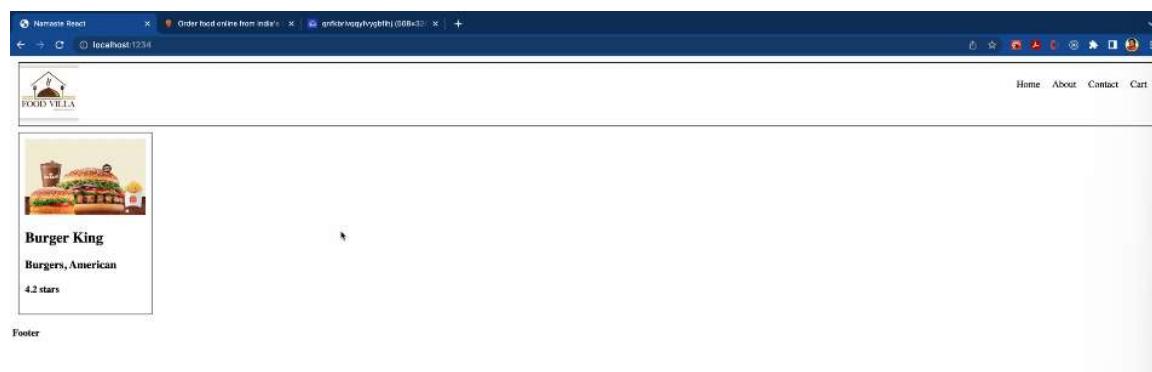
Let us call this inside our body, Let us make our body as a div

```
const Body = () => {
  return (
    <div>
      <RestrauntCard />
    </div>
  );
};
```



Let us give it some styling to our resto card and make it look little good

```
.card{  
    width: 200px;  
}  
  
.card > img {  
    width: 100%;|
```



We see our image, heading etc are Hard-coded in our resto card  
How to make it Dynamic?

Let us make an JS object and call it inside our functional component RestroCard

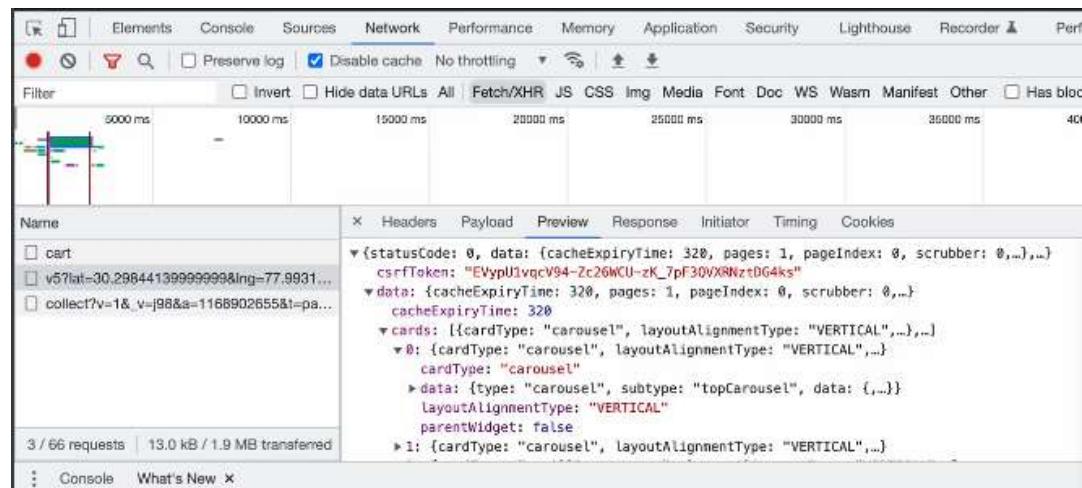
```
const burgerKing = {  
  name: "Burger King",  
  image: "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/qnfkbrlvqgy#vygb1cunis: ["Burger", "American"],  
  rating: "4.2"  
};  
  
const RestrauntCard = () => {  
  return (  
    <div className="card">  
      <img src={burgerKing.image} />  
      <h2>{burgerKing.name}</h2>  
      <h3>{burgerKing.cusines}</h3>  
      <h4>{burgerKing.rating} stars</h4>  
    </div>  
  );  
};
```

When we pass cuisines in our JSX, it takes it as an Array and **BurgerAmerican** comes like this, we need it to look like **Burger, American** so we use something called **.join(",")**

```
const RestrauntCard = () => {
  return (
    <div className="card">
      <img src={burgerKing.image} />
      <h2>{burgerKing.name}</h2>
      <h3>{burgerKing.cuisines.join(",")}</h3>
      <h4>{burgerKing.rating} stars</h4>
    </div>
  );
};
```

We need Dynamic Data Such that number of Restro cards changes dynamically according to data we get, In reality does not come as object, it comes as Array of objects

We check **API of swiggy** website using **Network Tab** of browser and Go to **Fetch/XHR**



The screenshot shows the Network tab in the developer tools with the Fetch/XHR tab selected. A request for 'cart' is expanded, showing its Headers, Payload, Preview, Response, Initiator, Timing, and Cookies. The Response section displays a complex JSON object. The JSON structure includes fields like statusCode, data (with cacheExpiryTime, pages, pageIndex, scrubber), csrfToken, and cards. Each card object has properties like cardType ('carousel'), layoutAlignmentType ('VERTICAL'), data (type 'carousel', subtype 'topCarousel'), and parentWidget (false). The JSON is displayed with collapsible arrows for nested objects.

This is how JSON of Swiggy looks like:

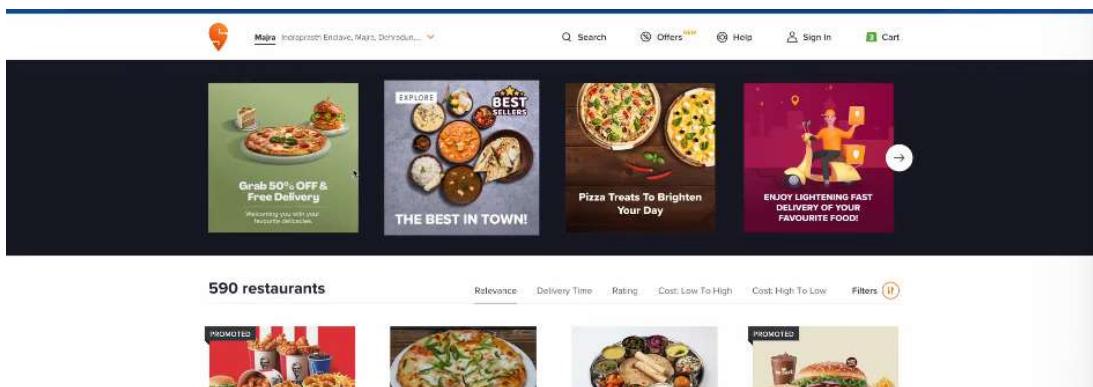
```

// 20230107123212
// https://www.swiggy.com/dapi/restaurants/list/v5?lat=30.2984413999999

{
  "statusCode": 0,
  "data": {
    "cacheExpiryTime": 320,
    "pages": 1,
    "pageIndex": 0,
    "scrubber": 0,
    "filters": [],
    "sorts": [],
    "configs": [
      "ribbons": [
        "PREORDER": {
          "type": "PREORDER",
          "text": "Preorder",
          "textColor": "#fffff",
          "imageId": "sfefefefegeg",
          "topBackgroundColor": "#d53d4c",
          "bottomBackgroundColor": "#af2330",
          "priority": 3
        },
        "EXCLUSIVE": [
          {
            "type": "EXCLUSIVE",
            "text": "Exclusive",
            "textColor": "#fffff",
            "imageId": "sfefefefegeg",
            "topBackgroundColor": "#fa4a5b",
            "bottomBackgroundColor": "#d12a3b",
            "priority": 2
          }
        ],
        "NEW": [
          {
            "type": "NEW",
            "text": "Newly Added",
            "textColor": "#fffff",
            "imageId": "sfefefefegeg",
            "topBackgroundColor": "#d53d4c",
            "bottomBackgroundColor": "#af2330",
            "priority": 4
          }
        ],
        "UPCOMING": [
          ...
        ]
      ]
    ]
  }
}

```

Let say There are some offers going on in Swiggy for Dehredun which are listed in a Carousel in a Website but there are no offers in Delhi so when someone opens the website in Delhi he/she does not see that carousel whereas if someone opens the website in Dehradoon, he/she sees that Carousel. So This is called **Config-Driven UI**  
 This is a good practice followed in the industry



When you build a real-world Applications, we want our same website to work in all parts of country and we cannot make different website for different countries, so we control the UI based on data we get from backend, so backend controls what will be seen in Delhi, what will be seen in Kolkata so this is called Config-Driven UI.

This is good practice to follow in Machine coding rounds and in the industry.

#### How to design Config-driven UI?

Let say our UI has Carousel, Food Cards, Offer cards So we make a big object.

```
//Config Driven UI

const config = [
  {
    type: "carousel",
    cards: [
      {
        offerName: "50% off"
      },
      {
        offerName: "No Delivery Charge"
      }
    ]
  }
]
```

Now we add List of resto below Carousel with type: "Restro"

Now we add our resto inside it.

```
  },
  {
    type: "restaurants",
    cards: [
      {
        name: "Burger King",
        image:
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_328,c_fill/gnfkbrlvqqyfvygt",
        cusines: ["Burger", "American"],
        rating: "4.2",
      },
      {
        name: "KFC",
        image:
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_328,c_fill/gnfkbrlvqqyfvygk",
        cusines: ["Burger", "American"],
        rating: "4.2",
      }
    ]
  }
]
```

Let us Take Card Data from Swiggy Website and use it

## Optional Chaining (?.)

The **optional chaining (?.)** operator accesses an object's property or calls a function. If the object accessed or function called using this operator is undefined or null, the expression short circuits and evaluates to undefined instead of throwing an error.

Now we call our First element of Array of objects we have in our JSON, like this

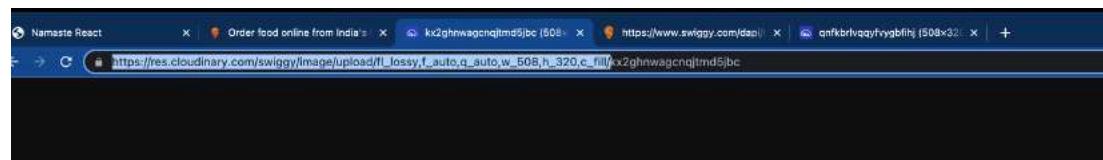
```
//Opt
const RestrauntCard = () => {
  return (
    <div className="card">
      <img src={burgerKing.image} />
      <h2>{restrauntList[0].data?.name}</h2>
      <h3>{burgerKing.cusines.join(", ")}</h3>
      <h4>{burgerKing.rating} stars</h4>
    </div>
  );
};
```

In name we use `(?)` so that if there is no such data then it does not throw error to us.

```
const RestrauntCard = () => {
  return (
    <div className="card">
      <img src={burgerKing.image} />
      <h2>{restrauntList[0].data?.name}</h2>
      <h3>{restrauntList[0].data?.cuisines.join(", ")}</h3>
      <h4>{restrauntList[0].data?.lastMileTravelString} minutes</h4>
    </div>
  );
};
```

We see in our Images of Cards, URL is like `res.cloudinary....`

What is `res.cloudinary` things?



**Blue part** is same for all images, **Non blue part** is ID which we give to get any particular image which is already present in our JSON.

```
totalRatingsString, totalRow Ratings,
cloudinaryImageId: "kx2ghnwagcnqjtm5jbc",
```

Now our Restro Card looks like:

```
const RestrauntCard = () => {
  return (
    <div className="card">
      <img
        src={
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/" +
          restrauntList[1].data?.cloudinaryImageId
        }
      />
      <h2>{restrauntList[0].data?.name}</h2>
      <h3>{restrauntList[0].data?.cuisines.join(", ")}</h3>
      <h4>{restrauntList[0].data?.lastMileTravelString} minutes</h4>
    </div>
  );
};
```

We need to pass Dynamic Data, Each card should be dynamic

We can also do looping

```
//Config Driven UI

const restrautList = [
>  {...},
>  {...},
>  {...},
>  {...},
>  {...},
>  {...},
>  {...},
>  {...},
>  {...},
>  {...},
];

```

We want our card as

```
const Body = () => {
  return (
    <div className="restaurant-list">
      <RestrauntCard restaurant = {restrautList[0]} />
      <RestrauntCard restaurant = {restrautList[1]} />
      <RestrauntCard restaurant = {restrautList[2]} />
      <RestrauntCard restaurant = {restrautList[3]} />
      <RestrauntCard />
      <RestrauntCard />
      <RestrauntCard />
    </div>
  );
}
```

Now how to implement this in our RestroCard Functional component, We use something known as **props**

Props is short-hand for "**Properties**"

```
//props - properties
const Body = () => {
  return (
    <div className="restaurant-list">
      <RestrauntCard restaurant={restrautList[0]} />
      <RestrauntCard restaurant={restrautList[1]} />
      <RestrauntCard restaurant={restrautList[2]} />
      <RestrauntCard restaurant={restrautList[3]} />
      <RestrauntCard restaurant={restrautList[4]} />
      <RestrauntCard restaurant={restrautList[5]} />
    </div>
  );
};
```

Props is way of passing data inside our component

If we want to pass data from parent component to Child component we use props

In above image, body is Parent, RestrauntCard is child

**We now take this data inside RestrauntCard we take it as normal Arguments and Parameters**

**As its just a normal function**

## What are Arguments and Parameters?

When we defines function we call it Parameters,

When we call function we call it Argument

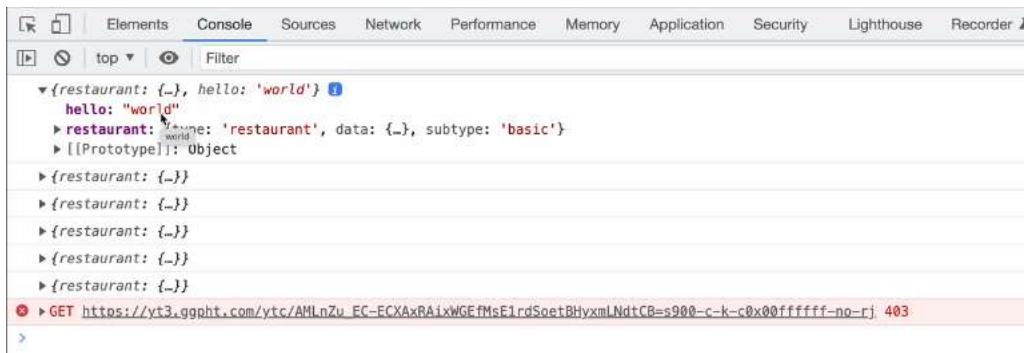
**You pass Arguments, You Receive Parameters**

```
function fn (parm1, parm2){  
}  
  
fn("xyz", "tyx")
```

We can pass as many props

```
//props - properties  
const Body = () => {  
  return (  
    <div className="restaurant-list">  
      <RestrauntCard restaurant={restrauntList[0]} hello="world" />  
      <RestrauntCard restaurant={restrauntList[1]} />  
      <RestrauntCard restaurant={restrauntList[2]} />  
      <RestrauntCard restaurant={restrauntList[3]} />  
      <RestrauntCard restaurant={restrauntList[4]} />  
      <RestrauntCard restaurant={restrauntList[5]} />  
    </div>  
  );  
};
```

We console log and we see



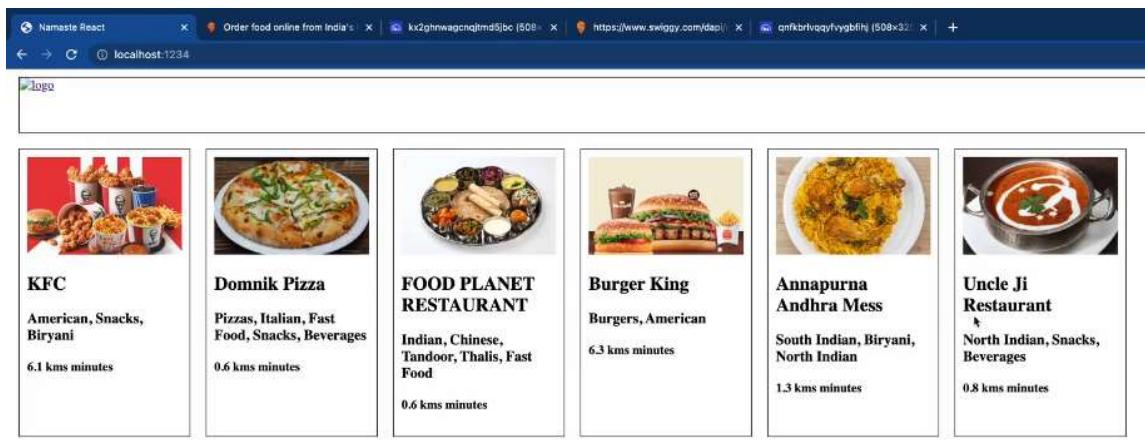
We receive our props as

```

const RestrauntCard = (props) => {
  console.log(props);
  return (
    <div className="card">
      <img
        src={
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/" +
          props.restaurant.data?.cloudinaryImageId
        }
      />
      <h2>{props.restaurant.data?.name}</h2>
      <h3>{props.restaurant.data?.cuisines.join(", ")!}</h3>
      <h4>{props.restaurant.data?.lastMileTravelString} minutes</h4>
    </div>
  );
}

```

Our output looks like:



Footer

Instead of writing props, We can also do **Array Destructuring of props** like this

```

const RestrauntCard = ({restaurant}) => {
  console.log(props);
  return (
    <div className="card">
      <img
        src={
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/" +
          restaurant.data?.cloudinaryImageId
        }
      />
      <h2>{restaurant.data?.name}</h2>
      <h3>{restaurant.data?.cuisines.join(", ")!}</h3>
      <h4>{restaurant.data?.lastMileTravelString} minutes</h4>
    </div>
  );
}

```

We can also do it like

```

const RestrauntCard = ({ restaurant }) => [
  const { name, cuisines, cloudinaryImageId, lastMileTravelString } =
    | restaurant.data;           You, now + Uncommitted changes
  return (
    <div className="card">
      <img
        src={[
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/" +
          cloudinaryImageId
        ]}
      />
      <h2>{name}</h2>
      <h3>{cuisines.join(", ")}</h3>
      <h4>{lastMileTravelString} minutes</h4>
    </div>
  );
];

```

Let say we want to **de-structure** like this

```

const RestrauntCard = ({ name, cuisines, cloudinaryImageId, lastMileTravelString }) => [
  return (
    <div className="card">
      <img
        src={[
          "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/" +
          cloudinaryImageId
        ]}
      />
      <h2>{name}</h2>
      <h3>{cuisines.join(", ")}</h3>
      <h4>{lastMileTravelString} minutes</h4>
    </div>
  );
];

```

What we need to pass in the body as props??

```

//props - properties
const Body = () => {
  return [
    <div className="restaurant-list">
      <RestrauntCard
        name={restrauntList[0].data.name}
        cuisines={restrauntList[0].data.cuisines}
      />
      <RestrauntCard
        name={restrauntList[1].data.name}
        cuisines={restrauntList[1].data.cuisines}
      />
      <RestrauntCard
        name={restrauntList[2].data.name}
        cuisines={restrauntList[2].data.cuisines}
      />
    </div>
  ];
};

```

We pass name, cuisines and **de-structure**

But **instead of writing name, cuisines etc individually and pass them as props, we can also do something like this.**

As we know our `restraunt.data` has everything we need so we can also do something like this  
We can use **(Spread Operator)**

```
//props - properties
const Body = () => {
  return (
    <div className="restaurant-list">
      <RestrauntCard {...restrautList[0].data} />
      <RestrauntCard {...restrautList[1].data} />
      <RestrauntCard {...restrautList[2].data} />
      {/* <RestrauntCard restaurant={restrautList[3]} />
      <RestrauntCard restaurant={restrautList[4]} />
      <RestrauntCard restaurant={restrautList[5]} /> */}
    </div>
  );
};
```

Now we know we are passing [0], [1], [2]..... to each card, let say there are 50 cards then what will we do??

```
//props - properties
const Body = () => {
  return (
    <div className="restaurant-list">
      <RestrauntCard {...restrautList[0].data} />
      <RestrauntCard {...restrautList[1].data} />
      <RestrauntCard {...restrautList[2].data} />
      <RestrauntCard {...restrautList[3].data} />
      <RestrauntCard {...restrautList[4].data} />
      <RestrauntCard {...restrautList[5].data} />
    </div>
  );
};
```

We need to do **looping using MAP()**

**We can also use foreach loop but we prefer to use Map() but why??**

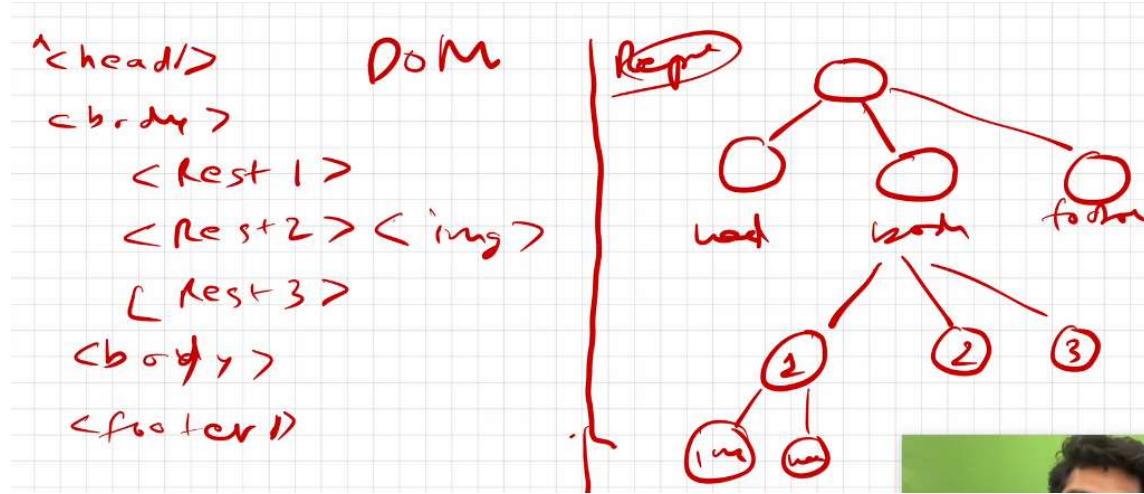
The **forEach()** method does not returns a new array based on the given array. The **map()** method returns an entirely new array. The **forEach()** method returns “undefined”. The **map()** method returns the newly created array according to the provided callback function

```
//props - properties
const Body = () => {
  return (
    <div className="restaurant-list">
      {restrautList.map((restaurant) => [
        return <RestrauntCard {...restaurant.data} />;
      ])
    </div>
  );
};
```

**What is Virtual DOM??**

Virtual DOM is not a concept of just React  
Its an Software engineering concept but its also there in React.

Its an Concept which says, we keep a representation of our DOM in our code with us, this is called Virtual DOM.



Why do we need Virtual DOM in react??

You need Virtual DOM for Reconciliation for React

Reconciliation is an Algorithm which react uses to diff one tree from other and it determines what need to change in UI and what not to change in UI.

Reconciliation is **the process by which React updates the UI to reflect changes in the component state**. The reconciliation algorithm is the set of rules that React uses to determine how to update the UI in the most efficient way possible. React uses a virtual DOM (Document Object Model) to update the UI

The **virtual DOM is a lightweight in-memory representation of the real DOM**, which allows React to make changes to the UI without manipulating the actual DOM. This makes updates faster, as changing the virtual DOM is less expensive than changing the real DOM.

The reconciliation algorithm works by **comparing the current virtual DOM tree to the updated virtual DOM tree, and making the minimum number of changes necessary to bring the virtual DOM in line with the updated state**.

The **algorithm uses two main techniques** to optimize updates:

⌚ **Tree diffing**: React **compares the current virtual DOM tree with the updated virtual DOM tree, and identifies the minimum number of changes necessary to bring the virtual DOM in line with the updated state**.

⌚ **Batching**: React **batches multiple changes into a single update, reducing the number of updates to the virtual DOM and, in turn, the real DOM**.

The reconciliation algorithm is a critical part of React's performance and helps make React one of the fastest and most efficient JavaScript libraries for building user interfaces. After the reconciler compares the current and updated virtual DOM, it identifies the

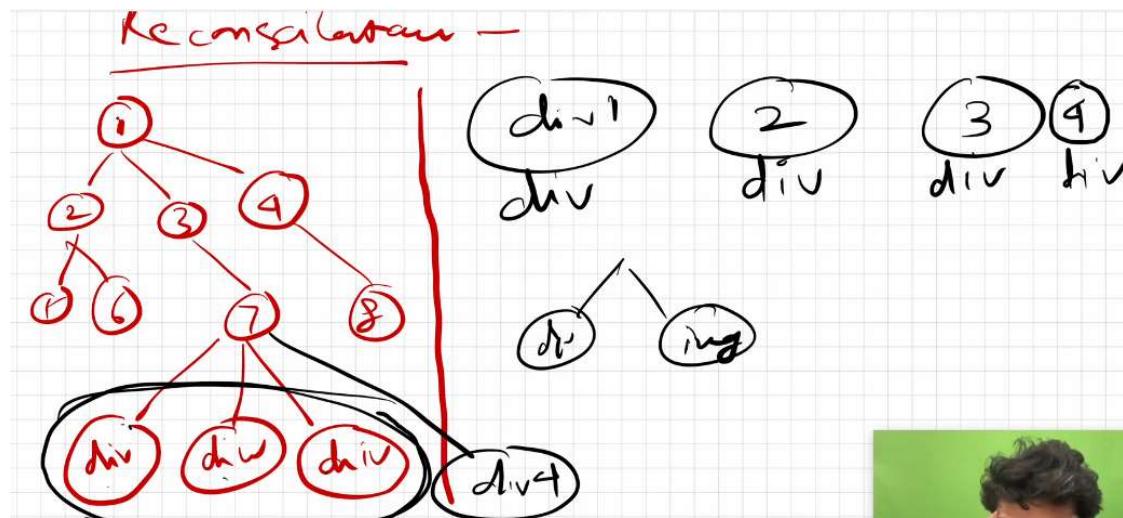
differences and makes the necessary changes to the virtual DOM to bring it in line with the updated state.

This updating of the virtual DOM is done for several reasons:

1. To **keep the virtual DOM as an accurate representation of the current state of the components, so that it can be used in future updates.**
2. To **optimize updates to the real DOM.** The virtual DOM provides a way for React to make changes to the UI without manipulating the real DOM directly. By making changes to the virtual DOM, React can determine the most efficient way to update the real DOM and make only the minimum number of changes necessary.
3. To **ensure that the UI remains in sync** with the state of the components. The reconciliation process ensures that the virtual DOM accurately reflects the current state of the components so that the UI remains up-to-date and in line with the state of the application.

In summary, **the reconciler updates the virtual DOM after the reconciliation process to keep it accurate, optimize updates to the real DOM, and ensure that the UI remains in sync with the state of the components.**

In simple Language, if something is changed in some part of DOM tree it should not re-render whole DOM tree but should re-render only the updated part of DOM so we make use of virtual DOM and this makes React faster.

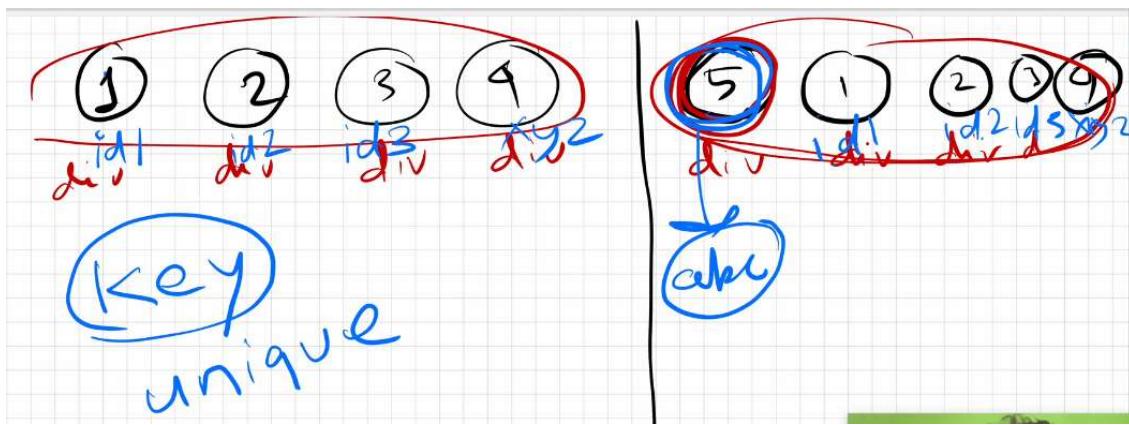


Suppose we have 4 div in our code, we add one more div now we have 5 div in our code. Our react sees 5 div and re-renders everything but If we use something like "key" with each sibling div which is **unique to everybody**.

If we have siblings with same tag like many div's or many img's then we need to pass a "key", if all components are different like one div, one img, one h1 then we need not to pass the key as react can automatically differentiate using tag names.

Now react will get to know 4 divs were already present only the 5th div is newly added and it re-renders only the new div and saves the performance.

Finding this new component is done by diffing Algorithm which is used by React.



Virtual DOM is just an representation of actual DOM, its not actual DOM.  
This whole Reconciliation is happening inside the browser.

## React Fiber

It is a **reimplementation of older versions of the React reconciler**.

**Introduced from React 16**, Fiber Reconciler is the new reconciliation algorithm in React. The term Fiber refers to React's data structure (or) architecture, and originates from 'fiber' - a representation of a node of the DOM tree.

Using React Fiber we can jump from one render to another render and it gives flexibility to the developer to break the work into smaller chunks and render things based on priority or start, stop, resume, abort rendering of certain component based on priority  
It helps improve speed of rendering components at start-up.

### How React used to work before React Fiber??

- React **creates a tree of nodes when the UI renders for the very first time**, with **each node representing a React element**. React **creates a virtual tree (called virtualDOM)**, a clone of the rendered DOM tree.
- React then **traverses the tree, updating the DOM** on which classes or elements needed to be updated, whenever a change is required to be rendered. This is called Reconciliation.
- Essentially, after any UI update, React **compares every node from two trees**, and passes on the cumulative changes to the renderer.

But, **before Fiber, reconciliation and rendering to the DOM weren't separated, and React couldn't pause its traversal to jump to other renders in between. This often resulted in lagging inputs and choppy frame rates.**

- In other words, with **reconciliation forced to be without interruption (or "synchronous")**, **render changes couldn't be inserted in the middle**. This **prevented higher-priority changes from being made until the stack was completely cleared**.

### How React Fiber works??

Fiber **brings in different levels of priority for updates in React. It breaks the computation of**

**the component tree into nodes, or 'units' of work that it can commit at any time.** This allows React to pause, resume or restart computation for various components.  
Fiber allows the reconciliation and **rendering to the DOM to be split into two separate phases:**

## Phase 1: Reconciliation

- In the first phase, React creates a list of all changes to be rendered in the UI (an 'effect list', comprising of new and updated components).
- Once the list is fully computed, React will schedule these changes to be executed in the next phase.
- Note that React doesn't make any actual changes in this phase.

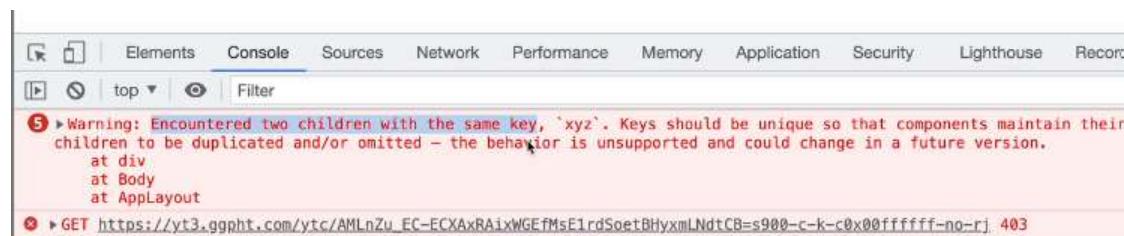
## Phase 2: Commit

- In phase two, also called 'commit' phase, React tells the DOM to render the effect list that was created in the previous phase.
- While the Reconciliation phase can be interrupted, the Commit phase cannot.

So via Fiber, React is able to traverse the component tree through a singly linked list tree traversal algorithm. This algorithm can run in an "asynchronous" manner - allowing React to pause and resume work at particular nodes.

So now let us use key in our MAP() inside Body

Let say we do not give a unique key, then react will itself complain that



So let us give a unique key to all the childs

```
//props - properties
const Body = () => {
  return (
    <div className="restaurant-list">
      {restrauntList.map((restaurant) => (
        <RestrauntCard {...restaurant.data} key={restaurant.data.id}>;
      ))}
    </div>
  );
}
```

## Why do not we use indexes are keys in react??

why don't we just use indexes as keys when looping through an array. Although, many developers have done that in their code it is not necessarily ideal. React recommends that you do not use indexes as keys, since **it could impact performance negatively and could lead to some unstable component behaviour.**

- It is preferred to not use indexes as a key unless you know for sure that the list is a static list (no additions/re-ordering/removal to the list).

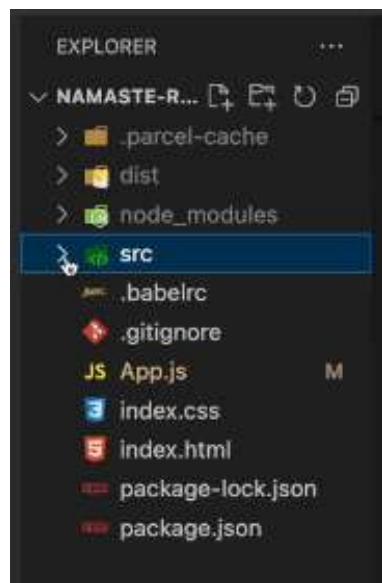
- Never use unstable keys like `Math.random()` to generate a key.
- React will run into performance degradation and unexpected behaviour if unstable keys are used.

**What if we do not have anything unique??**

**Then, we can use index as key instead of not giving any key.**

## Session 6

We have created a folder "src"



We do not need to make any extra folder in our app.

We need to wrap our code into a proper structure that makes our code modular and manageable if our code is very large.

We keep our App.js inside our src

### Is there a recommended way to structure React projects?

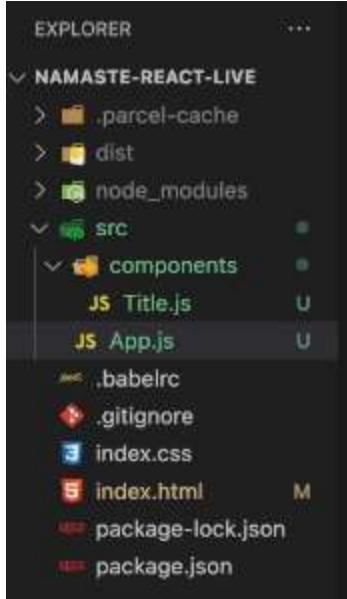
React doesn't have opinions on how you put files into folders. That said there are a few common approaches popular in the ecosystem you may want to consider.

#### **Grouping by features or routes || Grouping by file type**

We can group our files based on some features or routes and make components folder inside src and put all our components inside it.

We need to export the file also

There are 2 ways of exporting a file



Title.js — namaste-react-live

```
JS App.js U JS Title.js U X index.html M
src > components > JS Title.js > default
1 const Title = () => (
2   <a href="/">
3     
7   </a>
8 );
9
10
11 export default Title;
12
```

Why to use "default" ???

Because there is another way of exporting.

Header.js — namaste-react-live

```
EXPLORER ... JS App.js U JS Header.js U X index.html M
src > components > JS Header.js > ...
1 const Title = () => (
2   <a href="/">
3     
7   </a>
8 );
9
10
11 const Header = () => {
12   return (
13     <div className="header">
14       <Title />
15
16       <div className="nav-items">
17         <ul>
18           <li>Home</li>
19           <li>About</li>
20           <li>Contact</li>
21         </ul>
22       </div>
23     </div>
24   );
25 }
```

We have Header and Title both inside one file but if we want to export both of them, can we use "export"??

No, we cannot, we can only export one filer using "export"

So here comes another way

Export directly like

```
Header.js — namaste-react-live
JS App.js U JS Header.js U X index.html M
src > components > JS Header.js > [o] Title
1  export const Title = () => (
2    <a href="/">
3       JS contants.js > ...
1  export const IMG_CDN_URL =
2  | "https://res.cloudinary.com/swiggy/image/upload/fl_llossy,f_auto,q_auto,w_508,h_320,c_fill/";
3
```



Now we import it

```

import Body from './components/Body';
import { IMG_CDN_URL } from "./contants";

```

And we use it like:

```

const Body = ({ name, cuisines, lastMileTravelString }) => {
  return (
    <div className="card">
      <img src={IMG_CDN_URL + cloudinaryImageId} />
      <h2>{name}</h2>
      <h3>{cuisines.join(", ")!}</h3>
      <h4>{lastMileTravelString} minutes</h4>
    </div>
  );
};

export default Body;

```

Let us put whole Card Data in our constant.js and use it in body component and use it to render dynamic card components

```

const Body = ({ restaurantList }) => {
  const Body = () => {
    return (
      <div className="restaurant-list">
        {restaurantList.map((restaurant) => {
          return <RestrauntCard {...restaurant.data} key={restaurant.data.id} />;
        })}
      </div>
    );
  };

  export default Body;
}

```

Let us make an component for RestroCard also

The screenshot shows a code editor with the following file structure:

- EXPLORER**: Shows files and folders: .parcel-cache, dist, node\_modules, src (with components, App.js, contants.js, Footer.js, Header.js, RestrauntCard.js), .babelrc, .gitignore, index.css, index.html, package-lock.json, package.json.
- JS App.js**: Opened tab, contains imports for React and components.
- JS RestrauntCard.js**: Active tab, contains the component definition.
- JS contants.js**: Tab, contains the constant object.
- JS Footer.js**: Tab.
- JS Header.js**: Tab.
- JS RestrauntCard.js**: Content of the active tab:

```

src > components > JS RestrauntCard.js > [x] default
  1   import { IMG_CDN_URL } from "./contants";
  2
  3   const RestrauntCard = ({
  4     name,
  5     cuisines,
  6     cloudinaryImageId,
  7     lastMileTravelString,
  8   }) => {
  9     return (
 10       <div className="card">
 11         <img src={IMG_CDN_URL + cloudinaryImageId} />
 12         <h2>{name}</h2>
 13         <h3>{cuisines.join(", ")!}</h3>
 14         <h4>{lastMileTravelString} minutes</h4>
 15       );
 16     };
 17   };
 18
 19   export default RestrauntCard;
 20

```

Let us Build a Search Functionality inside body component (Just above our Cards)

```

const Body = () => {
  return [
    <>
      <div className="search-container">
        <input
          type="text"
          className="search-input"
          placeholder="Search"
          value=""
        />
        <button className="search-btn">Search</button>
      </div>
      <div className="restaurant-list">
        {restaurantList.map((restaurant) => {
          return (
            <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
          );
        });
      </div>
    </>
  ];
}

```

But if we write something in our search bar, its not working.  
Why??

If we write samething inside our HTML, it is working fine.

```

<body>
  <input type="text" className="search-input" placeholder="Search" value="" />
</body>

```

But it does not work find in react, because now react is controlling it.  
If we make an variable and pass it as value attribute to our seach input, it will becomes value of search input but we cannot update It now

```

const searchTxt = "KFC";

return (
  <>
    <div className="search-container">
      <input
        type="text"
        className="search-input"
        placeholder="Search"
        value={searchTxt}
      />
    </div>
  </>
);

```

## React uses **one-way Data Binding**

Data Binding is the process of connecting the view element or user interface, with the data which populates it.

The connection between the data to be displayed in the view and the component's logic is called data binding in ReactJS.

**One-way Data Binding:** ReactJS uses one-way data binding. In one-way data binding one of the following conditions can be followed:

- **Component to View:** Any change in component data would get reflected in the view.

- **View to Component:** Any change in View would get reflected in the component's data.

We need to modify the variable while we are writing something to see the changes getting reflected in our app so we use **onChange** in our input tag

**onChange** takes an function as input, **onChange** will fire whenever our input is changing

```
<div className="search-container">
  <input
    type="text"
    className="search-input"
    placeholder="Search"
    value={searchTxt}
    onChange={(e) => console.log(e.target.value)}>
</div>
```

Whenever we are writing something, we see it in console but we cannot see it getting changed inside search bar because KFC is hard-coded so we need to update the value of **searchTxt** also during **onChange**

We do something like this now

```
<input
  type="text"
  className="search-input"
  placeholder="Search"
  value={searchTxt}
  onChange={(e) => {
    searchTxt = e.target.value;
  }}>
```

But it also won't work

So to make this work, **local variables like `searchTxt` are not used in react, if we need to maintain a variable which keeps on changing, we need to make it like a React variable which is also known as "state".**

**Everytime we want to make a local variable we use "state" in react so we use something known as **useState** hook**

## useState() Hook

They are like normal functions. That is why we call it like **useState()**

Where does we get it from ??

We import it from "react" as named import

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

**Every hook has specific function for it.** `useState` is used to create Variable/State

## How to use useState??

useState() function returns an Array which has variable name as first argument and function as second argument.

We can also give deafult value to a useState variable like useState("KFC")

```
// searchText is a local state variable
const [searchText] = useState("KFC"); // To create state variable

return (
  <div className="search-container">
    <input
      type="text"
      className="search-input"
      placeholder="Search"
      value={searchText}
      onChange={(e) => {
        searchText = e.target.value;
      }}
    />
  </div>
)
```

How to create Variable in JS?

```
const searchTxt = "KFC";
```

How to create Variable in React?

```
// searchText is a local state variable
const [searchText] = useState("KFC"); // To create state variable
```

But we cannot directly modify our variable in React then how to modify this variable?

We can only modify it using a function which is again given by useState()

```
// searchText is a local state variable
const [searchText, setSearchText] = useState("KFC"); // To create state variable
```

Now we do like

```
value={searchText}
onChange={(e) => {
  setSearchText(e.target.value);
}}
```

Now our code looks like

```
// searchText is a local state variable
const [searchInput, setSearchInput] = useState("KFC"); // To create state variable

return (
  <>
    <div className="search-container">
      <input
        type="text"
        className="search-input"
        placeholder="Search"
        value={searchInput}
        onChange={(e) => {
          // e.target.value ⇒ whatever you write in input
          setSearchInput(e.target.value);
        }}
      />
    </div>
)
```

We can also destructure useState like

```
const searchvar = useState(); // returns = [variable name, function to update the variable]

const [searchText, setSearchText] = searchvar;
```

Now if we want to update our state we use "setSearchText" , we use onChange  
Which takes an arrow or normal function and it gives us "event" which contains our value  
we are typing. So using it we can update our state

```
const [searchText, setSearchText] = useState("hellow");

return (
  <>
    <div className="search-container">
      <input
        type="text"
        className="search-input"
        placeholder="Search"
        value={searchText}
        onChange={(e) => {
          setSearchText(e.target.value);
        }}
      />
    </div>
)
```

Now if we want to print "whatever we write in our state" simultaneously  
We write like

```

const [searchText, setSearchText] = useState("hellow");

return (
  <>
    <div className="search-container">
      <input
        type="text"
        className="search-input"
        placeholder="Search"
        value={searchText}
        onChange={(e) => {
          setSearchText(e.target.value);
        }}
      />
      <h1>{searchText}</h1>
    </div>
  </>
)

```

Now we are reading the text in h1 and writing it in search bar at the same time, this is called **2 way binding**.

useState returns us an Array.

## When we have local variables, why do we need state variables??

Suppose we make a local variable and somebody updated my local variable, react will not know when I have updated the local variable and It will not change it in UI.  
React has no idea about whatever happening to our variable so to let react know about it, we need to make a state then only react will register our Variable.

To make our variable to be in sync with the UI, we use state variables

Now after making our variable a state, whenever we change the state, it gets re-rendered because now react knows about it.

Let us make searchClicked = false, which becomes "true" when we click search button  
So we make a new state for it

```

const [searchClicked, setSearchClicked] = useState(false);

```

Now we want our searchClicked to become "true" on clicking "search"

```

<button
  className="search-btn"
  onClick={() => {
    setSearchClicked(true);
  }}
>
  Search
</button>

```

## BTS (Behind the State)

Whenever our state changes, our whole body component destroys and new component is created very fast.

Reconciliation is happening everytime our state is changing.

Only the h1 will be re-rendered.

### How to create a Toggle Button "false" to true on click, "true" to "false" on click

```
<button
  className="search-btn"
  onClick={() => {
    if (searchClicked === "true") {
      setSearchClicked("true");
    } else {
      setSearchClicked("false");
    }
  }}> abc false </button>
```

Now let us focus on **Search Functionality**

We need to filter the data and update the resto list on clicking "Search" Button.

How to update Restro list on click??

To maintain the list of Restro we need to make another state.

```
const [restaurants, setRestaurants] = useState(resturantList);
```

So on clicking "Search" we will filter the data and change the state of Restro list.

How to write filter algorithm??

We need to filter resto list based on whatever we write inside input of search bar

So we make an **function filterData** and pass searchText and restuarants as arguments.

```
<button
  className="search-btn"
  onClick={() => {
    //need to filter the data
    filterData(searchText, restaurants);
    // update the state - restaurants
  }}>
  Search
</button>
```

Now we update the restoList using the data

```
<button
  className="search-btn"
  onClick={() => [
    //need to filter the data
    const data = filterData(searchText, restaurants);
    // update the state - restaurants
    setRestaurants(data);
  ]}>
  Search
</button>
```

Now let us write our filter algorithm

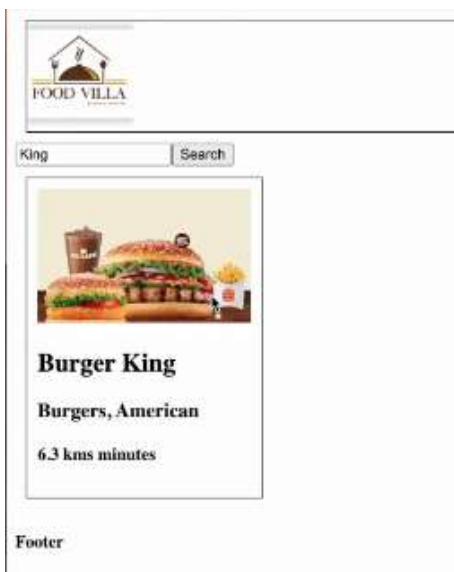
We know our List of restro is Array of Objects so let us apply filter() method of Array

```
function filterData(searchText, restaurants) {
  return restaurants.filter((restaurant) =>
    restaurant.data.name.includes(searchText)
  );
}
```

We can also write it like:

```
function filterData(searchText, restaurants) {
  const filterData = restaurants.filter((restaurant) =>
    restaurant.data.name.includes(searchText)
  );

  return filterData;
}
```



But now if we empty the search bar and click Search, it won't show anything because now state = " " and there is value after filtering which matches " "

## Session 7 (QNA)

## Session 8

useState hook helps us in making local variables.

Whenever there is something that changes in UI, we use local state to handle that and show that in UI.

We make a state so that react can register that yes, a change has been made and sync is made between UI and our state so that it gets reflected on the UI.

Why React is fast?

What makes React very fast?

It has virtual DOM, Reconciliation, Diff Algorithm which helps in faster DOM manipulation which makes it fast.

Whenever we make a state variable and use useState() to handle it, whenever we change the state whole component gets re-rendered but only that part will be updated where value of our state lies, this is due to re-conciliation.

Let us do a console.log and check if its prints or not. Console.log will render once in the beginning when whole component loads but now when we change state let see if it prints or not.

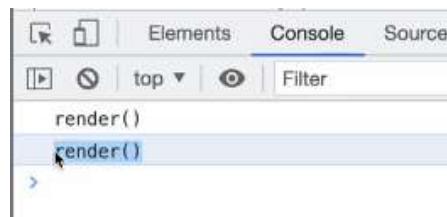
```
const Header = () => {
  const [title, setTitle] = useState("Food Villa");

  console.log("render()");

  return [
    <div className="header">
      <Title />
      You, 6 days ago • chapter-05
      <h1>{title}</h1>
      <button onClick={() => setTitle("New Food App")}>Change title</button>

      <div className="nav-items">
        <ul>
          <li>Home</li>
          <li>About</li>
          <li>Contact</li>
          <li>Cart</li>
        </ul>
      </div>
    </div>
  ];
};
```

It does prints console.log again so it re-renders whole component very fast on changing the state and it triggers the reconciliation process and if only needed part is updated in the DOM tree.



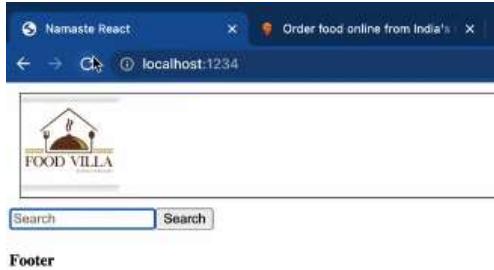
**Let us work on our Search bar issue**

**(But now if we empty the search bar and click Search, it won't show anything because now state = " " and there is value after filtering which matches " " )**

Suppose we do not want to change anything in UI we can use normal variable but If we want

our changes to get reflected in UI, we need to use State

The **issue with the search bar** is, for the first time it gives us correct result for whatever we search but on searching again it does not show anything



What happenend is when we entered something in SearchText and pressed "Search" button, it triggered filterData function and our Restro list got updated according to the SearchText which is shown in UI and now if we again change the searchText and again press "Search" then it will trigger filterData but now our resto list contain only previous result and our searchText will be compared from only that previous result and if searchText does not have anything matching the resto list, we get empty [ ] in resto list.

### How to Resolve this issue then?

Let us use an Outside API now instead of using local data.

```
// https://www.swiggy.com/dopi/restaurants/list/v5?lat=12.9351929&lng=77.6244806999999&page_type=DESKTOP_WEB_LISTING
```

What is [www.swiggy.com/dopi](https://www.swiggy.com/dopi) in above URL??

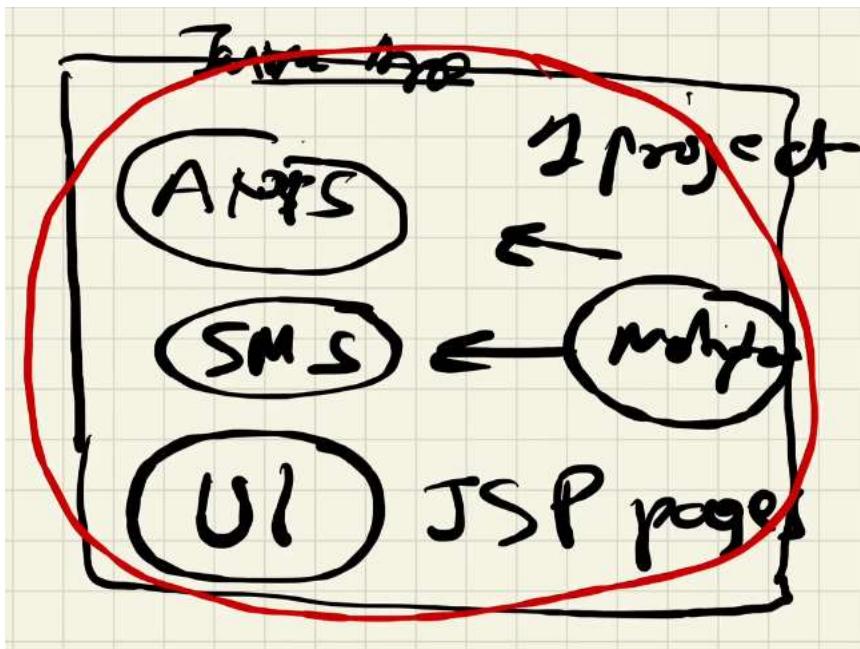
### Micro Service Architecture

When we build a big app

Earlier there used to be a single big application say Java application and this same application use to have everything API, UI, SMS sending, Notification.

If we have to change just one button, we would deploy whole application again. All developers used to work in one project. This architecture is called **Monolith**

One Advantage of this architecture is that you do not need to maintain many apps together.

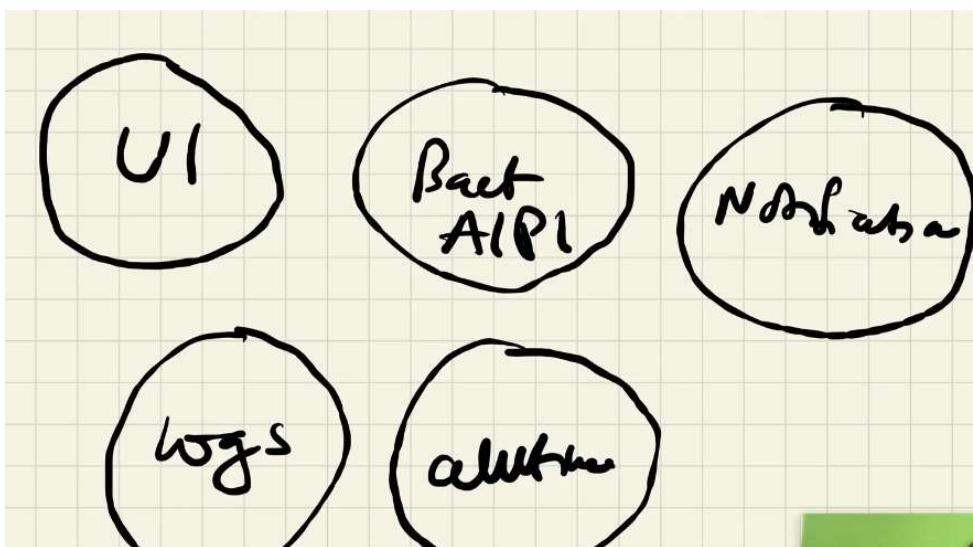


### Micro Service Archihtecture

Instead of same project we now have many small small projects like UI project, Auth project, API project, Notification Project etc etc. Everything hosted on different ports.

Even Database has many replicas and many databases are used.

We can use different Tech stacks in each project. React in Frontend, SQL in Database etc etc. One advantage of this is that its easy to maintain, each project is handled by a different team and they are responsible for it. UI team not need to think of backend and backend need not to think of UI. This is called **Separation of Concern**.



A **monolithic application** is built as a single unified unit while a **microservices** architecture is a collection of smaller, independently deployable services.

A **monolithic architecture** is a singular, large computing network with one code base that couples all of the business concerns together. To make a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-

consuming.

A **microservices architecture**, also simply known as microservices, is an architectural method that relies on a series of independently deployable services. These services have their own business logic and database with a specific goal. Updating, testing, deployment, and scaling occur within each service. Microservices decouple major business, domain-specific concerns into separate, independent code bases. Microservices don't reduce complexity, but they make any complexity visible and more manageable by separating tasks into smaller processes that function independently of each other and contribute to the overall whole.

### How these all small projects are connected to each other?

Suppose port 3000 is mapped to /

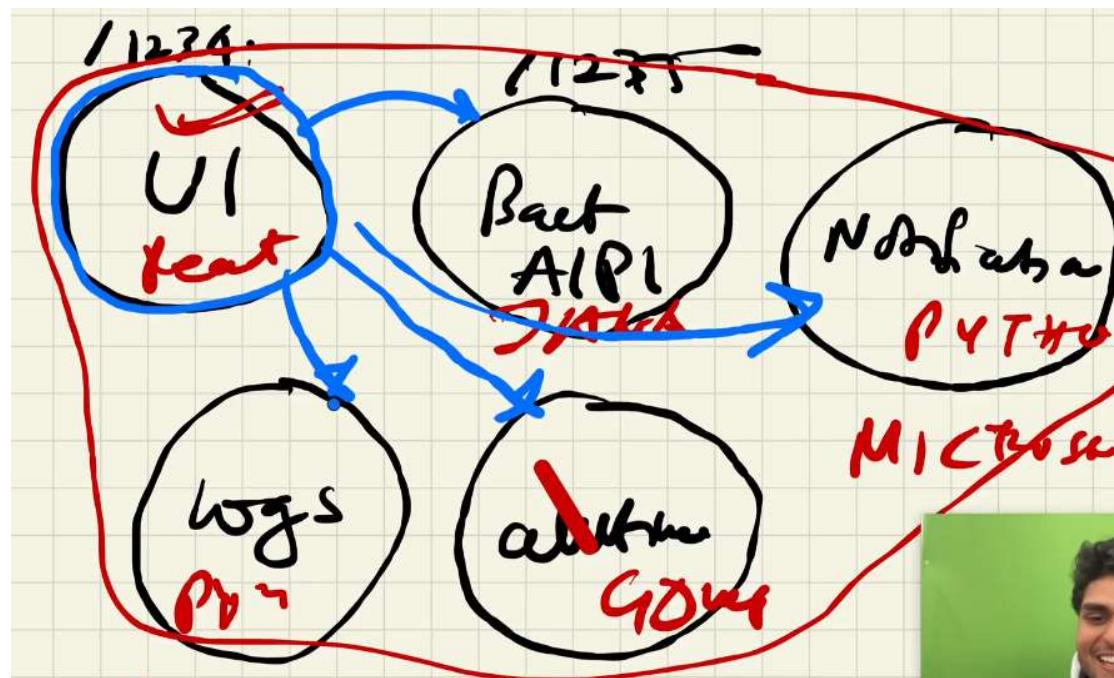
4000 is mapped to /dapi

5000 is mapped to /notification

All these projects are deployed on different port with same domain name and sometime domain name is also different

// [https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.6244806999999&page\\_type=DESKTOP\\_WEB\\_LISTING](https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.6244806999999&page_type=DESKTOP_WEB_LISTING)

Now we as a UI service, Will connect to other projects like API, Database, notification, auth



### Using Fetch()

It is used to Call the API.

Where should we make our API Call?

If we call our API inside component like

```

const Body = () => {
  fetch("/")
  const [restaurants, setRestaurants] = useState(restau
  const [searchText, setSearchText] = useState("");
  console.log(restaurants);

  return (
    <>
      <div className="search-container">...
      </div>
      <div className="restaurant-list">
        {restaurants.map((restaurant) => {
          return (
            <RestaurantCard {...restaurant.data} key={r

```

It will re-render everytime state changes and API will be called again and again so it is not good practice to call an API inside a component.

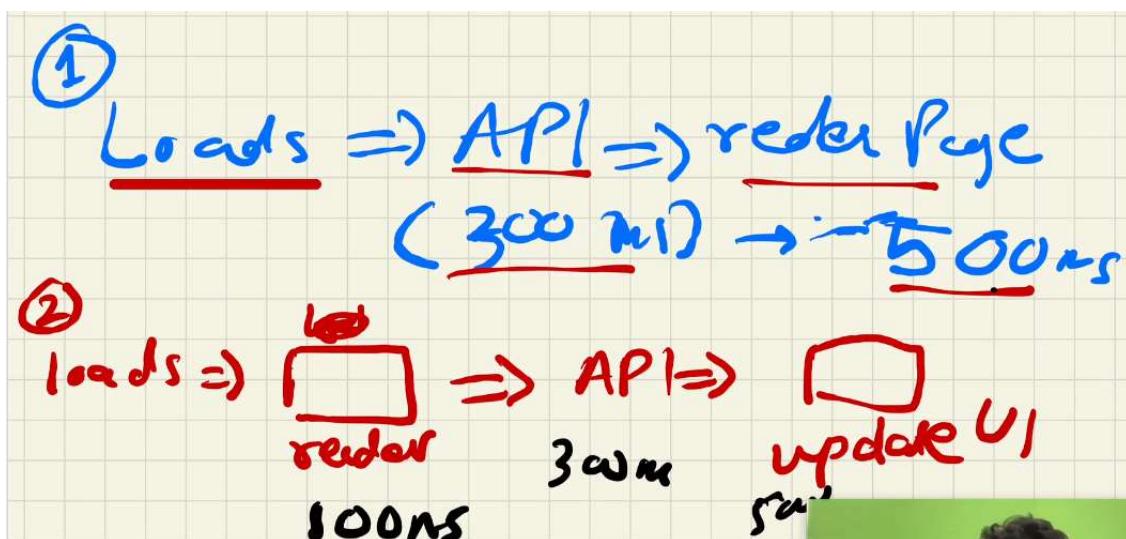
There is a good way to do this in React

There can be **2 ways to do this.**

Suppose user loads the website then

**Approach 1:** Loads the page -> make API call suppose it takes 300 seconds -> render the page after 300 seconds after API response

**Approach 2:** As soon as page loads -> show something to user, render intial page -> call the API and render it in the UI.



In Approach 1, Page will be visible in 300sec whereas In Approach 2 we have something on the page till our API call is happening and Due to Reconciliation Render the API response on the page later.

So to make Approach 2 possible React gives us a functionality called **useEffect() Hook**

So we will render our page quickly and then make an API call.

Let see **how useEffect() Hook works**

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

useEffect is a function which takes an Callback Function and an Array of Dependency. Callback Function will only be called only when our useEffect wants to call that function and React makes sure that it calls it at some specific time which is mentioned inside Array of Dependency [ ].

If we write useEffect like this, then it will call after every re-render which happens on changing of a state.

```
useEffect(() => {
  console.log("render")
});
```

If we do not want to call it on every render, we pass a Dependency array [ ]

Now, it is not dependent on anything so it will be called just once.

```
useEffect(() => {
  console.log("render")
}, []);
```

Let say we only want to call useEffect when searchText changes means when state of searchText changes, useEffect will be called

```
useEffect(() => {
  console.log("render");
}, [searchText]);
```

### Will it be called before render or after render??

If we write something like

```
const Body = () => {
  const [restaurants, setRestaurants] = useState(restaurantList);
  const [searchText, setSearchText] = useState("");

  useEffect(() => {
    console.log("useEffect");      You, 15 seconds ago • Uncommitted
  }, []);                         T

  console.log("render");
}
```

Will it call "render" first or "useEffect"?

```
render  
useEffect
```

It happens because useEffect has a callback function which does not depend on anything means it will be rendered just once **after whole component renders**

So Whenever we have a callback function inside an useEffect with empty dependency array then it will be called once only after initial render

```
// empty dependency array ⇒ once after render  
// dep arry [searchText] ⇒ once after initial render + everytime after rederm (my searchText changes)
```

So **where should we make our API call??**

We should make our API call inside useEffect with an Empty Dependency Array []

```
useEffect(() => [  
    // API call  
], []);
```

**Let us make an Fetch API request to swiggy API**

We will make Fetch API call in a separate function.

We know call happens Asynchronously so we can either use Promise or async/await

So we use async/await.

```
async function getRestaurants() {  
  const data = await fetch("https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.6244");  
  const json = await data.json();  
  console.log(json);  
}
```

```
useEffect(() => [  
    // API call  
    getRestaurants();  
, []];  
  
async function getRestaurants() {  
  const data = await fetch("https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.6244");  
  const json = await data.json();  
  console.log(json);  
}
```

We get this error due to security Parameters of Browser.

The screenshot shows the Chrome DevTools Console tab. At the top, there are tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, Lighthouse, and F. The Console tab is selected. Below the tabs, there's a toolbar with icons for back, forward, search, and filter. A message says "CONSOLE WAS CLEARED". The console output is as follows:

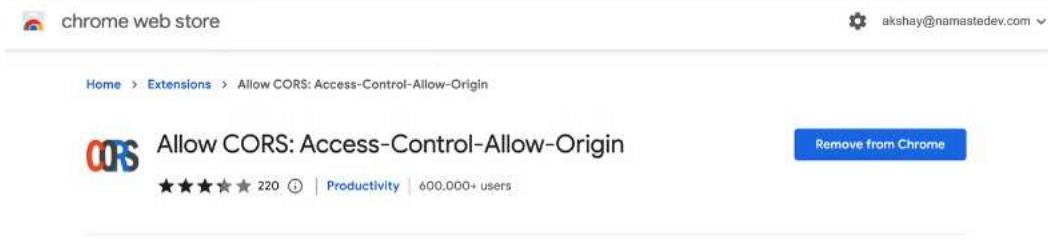
```

render
✖ Access to fetch at 'https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.6244806999999&page_type=DESKTOP_WEB' from origin 'http://localhost:1234' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource. If your server or service needs, set the request's mode to 'no-cors' to fetch the resource with CORS disabled.
✖ > GET https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=77.6244806999999&page_type=DESKTOP_WEB
200
✖ > Uncaught (in promise) TypeError: Failed to fetch
    at getRestaurants (VM2073 6399f84aa000606:31:28)
    at eval (VM2073 6399f84aa000606:28:9)
    at commitHookEffectListMount (react-dom.development.js:23150:26)
    at commitPassiveMountOnFiber (react-dom.development.js:24926:13)
    at commitPassiveMountEffects_complete (react-dom.development.js:24891:9)
    at commitPassiveMountEffects_begin (react-dom.development.js:24878:7)
    at commitPassiveMountEffects (react-dom.development.js:24866:3)
    at flushPassiveEffectsImpl (react-dom.development.js:27039:3)
    at flushPassiveEffects (react-dom.development.js:26984:14)
    at commitRootImpl (react-dom.development.js:26935:5)
    at commitRoot (react-dom.development.js:26682:5)
    at performSyncWorkOnRoot (react-dom.development.js:26117:3)
    at flushSyncCallbacks (react-dom.development.js:12042:22)
    at flushSync (react-dom.development.js:26201:7)
    at scheduleRefresh (react-dom.development.js:27795:5)

```

To solve the issue we can install this pluggin

It will let us by-pass the error

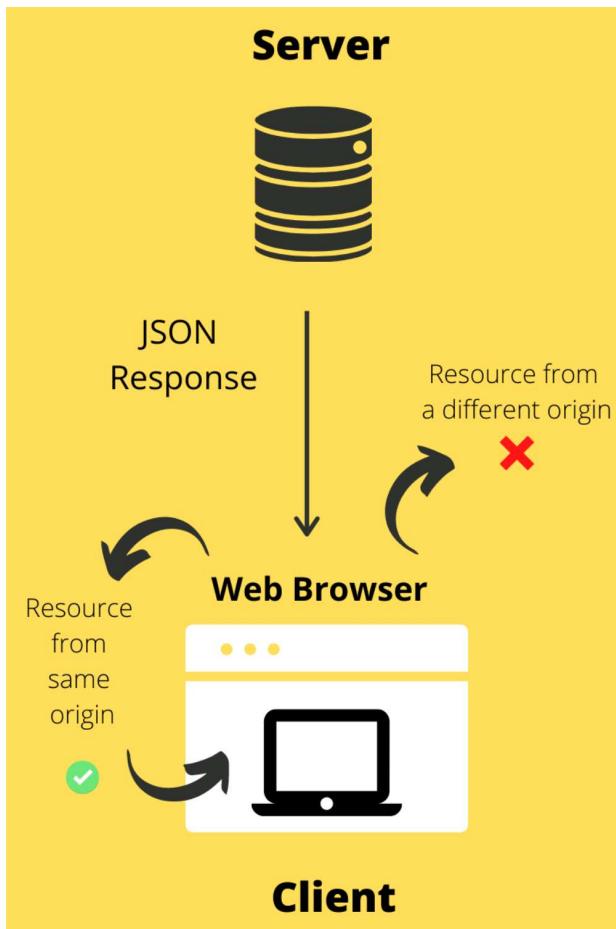


## CORS

CORS stands for cross-origin resource sharing. Just like HTTPS, it's a protocol that defines some rules for sharing resources from a different origin. We know that modern web apps consist of two key components: a client and a server. The client requests some data from the server, and the server sends back data as a response.

## The Same-Origin Policy

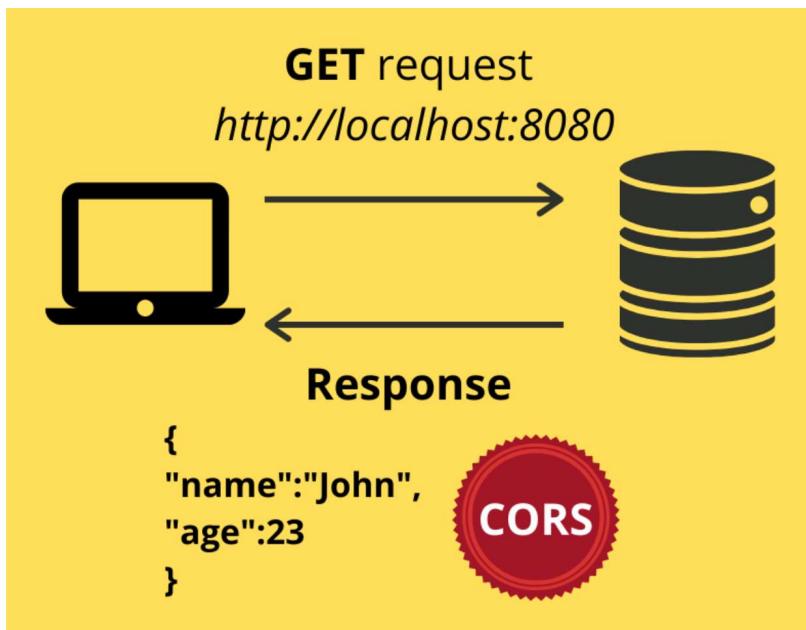
Since the client and server are separate applications, they're usually hosted on different domains. Therefore, your own client that's requesting data from your own server might have different origins. In another scenario, you might use some third-party services for authentication, analytics, etc. The bottom line is that at some point you are going to interact with an application with a different origin than yours. This means you're going to request resources from the application by making an HTTP request.



When you request a resource from an application of a different origin, the web browser uses an SOP (same-origin policy) protocol to block all your requests to that origin. Back in the day, this is what made the Internet secure! For instance, a malicious cracker or hacker from xyz.com wouldn't be able to access your information on abcbank.com. However, this underlying security rule governing browsers does not allow you to request a resource from a different origin.

## Enter CORS

CORS enables you to access a resource from a different origin. It is used to override your browser's default behavior due to SOP. So now when your client requests a resource, the response will additionally contain a stamp that tells your browser to allow resource sharing across different origins.



Once your browser identifies this stamp, responses for requests from different origins are allowed to pass through.

## Back to Code

After enabling the pluggin, now we get the data

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
v5?lat=12.9351929&lng=77.6...				<pre>{   "statusCode": 0,   "data": {     "cacheExpiryTime": 320,     "pages": 1,     "pageIndex": 0,     "scrubber": 0,     "csrfToken": "vd03uNOKM4zN-YIqREXtAJ4BrG3R0jtF66vDuCd4"   } }</pre>		10 ms	

Let us Insert the data to our UI.

We want to put this data in our UI

**JSON.data.cards[2].data.data.cards**

```

  * data:
    cacheExpiryTime: 320
  * cards: Array(3)
    > 0: {cardType: 'carousel', layoutAlignmentType: 'VERTICAL', data: {}, parentWidget: false}
    > 1: {cardType: 'carousel', layoutAlignmentType: 'VERTICAL', data: {}, parentWidget: false}
    > 2:
      cardType: "seeAllRestaurants"
    * data:
      * data:
        * cards: Array(15)
          > 0: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 1: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 2: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 3: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 4: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 5: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 6: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 7: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 8: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 9: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 10: {type: 'restaurant', data: {}, subtype: 'basic'}
          > 11: {type: 'restaurant', data: {}, subtype: 'basic'}

```

```

  setRestaurants(json.data.cards[2].data.data.cards);
}

```

The above way of wrong way to do this thing as it can break if something does not exists in above path, we should use something known as **optional chaining**.

```

// Optional Chaining
setRestaurants(json?.data?.cards[2]?.data?.data?.cards);

// App.js
useEffect(() => {
  // API call
  getRestaurants();
}, []);

async function getRestaurants() {
  const data = await fetch(
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lng=-77.6161284"
  );
  const json = await data.json();
  console.log(json);
  // Optional Chaining
  setRestaurants(json?.data?.cards[2]?.data?.data?.cards);      You, 17
}

```

## Our output



Search | Search

				
<b>Kannur Food Point</b> Kerala, Chinese 3 kms minutes	<b>Meghana Foods</b> Biryani, Andhra, South Indian, North Indian, Chinese, Seafood 1.2 kms minutes	<b>Roti Wala</b> Home Food, North Indian, Thalis 1 kms minutes	<b>Gramin</b> North Indian, Healthy Food, Beverages, Tandoor, Indian, Jain 1.3 kms minutes	<b>Hotel Empire</b> North Indian, Kebabs, Biryani 1.2 kms minutes
				
<b>Muthashy's</b>	<b>Biryani Pot</b>	<b>Asha tiffins</b>	<b>Soul Rasa</b>	<b>McDonald's</b>

When the page renders for the first time what will happen is, Old resto list will be displayed first.



Search | Search

					
<b>KFC</b> American, Snacks, Biryani 6.1 kms minutes	<b>Dominik Pizza</b> Pizzas, Italian, Fast Food, Snacks, Beverages 0.6 kms minutes	<b>FOOD PLANET RESTAURANT</b> Indian, Chinese, Tandoor, Thalis, Fast Food 0.6 kms minutes	<b>Burger King</b> Burgers, American 6.3 kms minutes	<b>Annapurna Andhra Mess</b> South Indian, Biryani, North Indian 1.3 kms minutes	<b>Uncle Ji Restaurant</b> North Indian, Snacks, Beverages 0.8 kms minutes

Footer

Now our useEffect fetches the API Data and put it in the state, state will change and our component will re-render and new list from API response is updated to



Search | Search

					
<b>Gramin</b> North Indian, Healthy Food, Beverages, Tandoor, Indian, Jain 1.3 kms minutes	<b>Meghana Foods</b> Biryani, Andhra, South Indian, North Indian, Chinese, Seafood 1.2 kms minutes	<b>Kannur Food Point</b> Kerala, Chinese 3 kms minutes	<b>Donne Biriyan Mane</b> Biryani, Kebabs 0.8 kms minutes	<b>Roti Wala</b> Home Food, North Indian, Thalis 1 kms minutes	<b>Hotel Empire</b> North Indian, Kebabs, Biryani 1.2 kms minutes
					
<b>Muthashy's</b>	<b>Biryani Pot</b>	<b>Asha tiffins</b>	<b>Soul Rasa</b>	<b>McDonald's</b>	<b>Uncle Ji Restaurant</b>

## Flow of program is:

1. Initial Re-render happens in the component in which Old resto list which we used while initialising our useState hook will render
2. Now flow moves inside useEffect and getRestaurants function will be called
3. We are inside getRestaurants function and now API call is made which gives us some data and we put that data in our state variable of resto list
4. Now component again re-renders because state has been changed and only resto list will be updated due to reconciliation concept in react
5. This is how our resto list gets updated and changes to API response data.

Now while refreshing our screen we are encountering some error sometimes

← → 1 of 3 errors on the page

TypeError: Cannot read properties of undefined (reading 'map')

Body  
src/components/Body.js:62:21

```
59 |   </button>
60 | </div>
61 | <div className="restaurant-list">
> 62 |   {restaurants.map(restaurant) =>
63 |     return (
64 |       <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
65 |     );

```

View compiled

► 14 stack frames were collapsed.

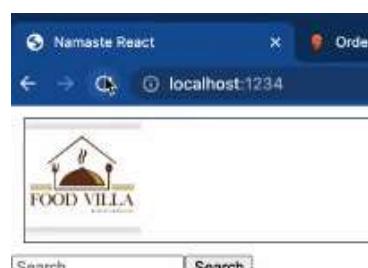
This screen is visible only in development. It will not appear if the app crashes in production.  
Open your browser's developer console to further inspect this error. Click the 'X' or hit ESC to dismiss this message.

We need to solve the errors

Let us first get rid of old resto list data and empty our initialisation of state variable

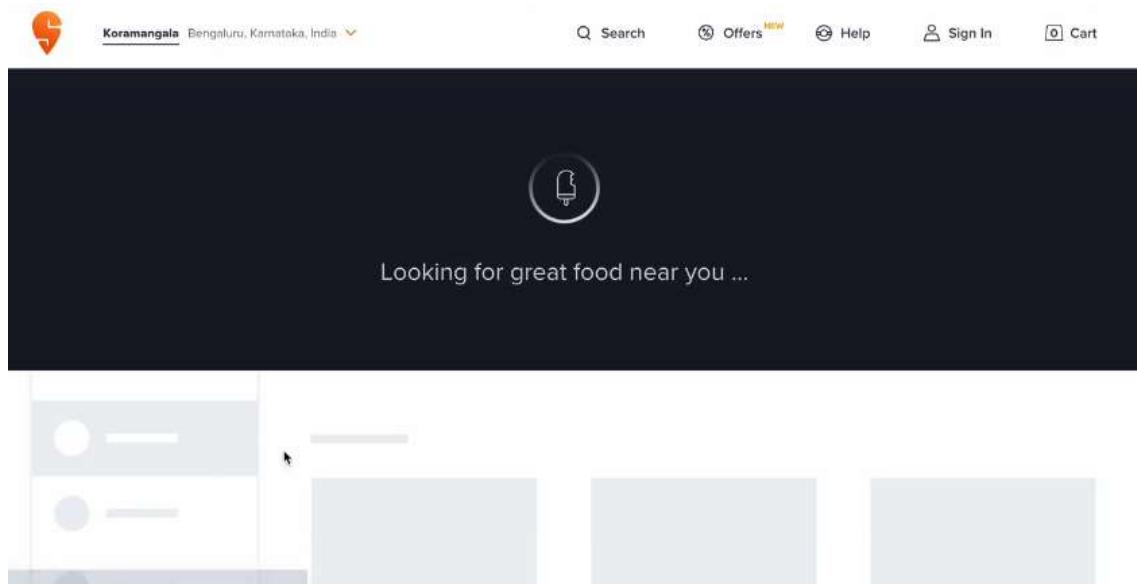
```
const [restaurants, setRestaurants] = useState([]);
```

But now when we render the UI, it shows this for sometime, which is looking ugly



To avoid this, we should show a loader

What swiggy does to handle this, let see  
This is the basic skeleton we load.



Earlier people used to show spinning loader and suddenly data comes up  
this is a bad UX, user does not want these fluctuations.

So Psychologist figured out show empty boxes till data is rendered means let user see  
empty boxes which get filled up as soon as data comes up.

This is known as **Shimmer UI**

## Shimmer UI: A Better Way to Show Loading States

If you have ever used a web or mobile app that takes some time to load data from a server, you might have seen a loading spinner or a progress bar that indicates that something is happening. While these are common ways to show loading states, they are not very engaging or informative for the user. They don't tell the user what kind of content is being loaded, how long it will take, or what to expect next.

A better way to show loading states is to use a shimmer UI. **A shimmer UI is a version of the UI that doesn't contain actual content, but instead mimics the layout and shapes of the content that will eventually appear.** It uses a gradient animation that creates a shimmering effect over the placeholders, giving the user a sense of motion and progress.

There are different ways to implement a shimmer UI depending on the platform and framework you are using. **For example, if you are using React, you can use the Shimmer package which provides a component that creates a shimmer effect over any component. You can also use the ShimmerLoading component that automatically switches between a shimmer UI and a content UI based on a boolean flag.**

All big companies like Facebook, LinkedIn etc uses Shimmer UI



Shimmer Effect

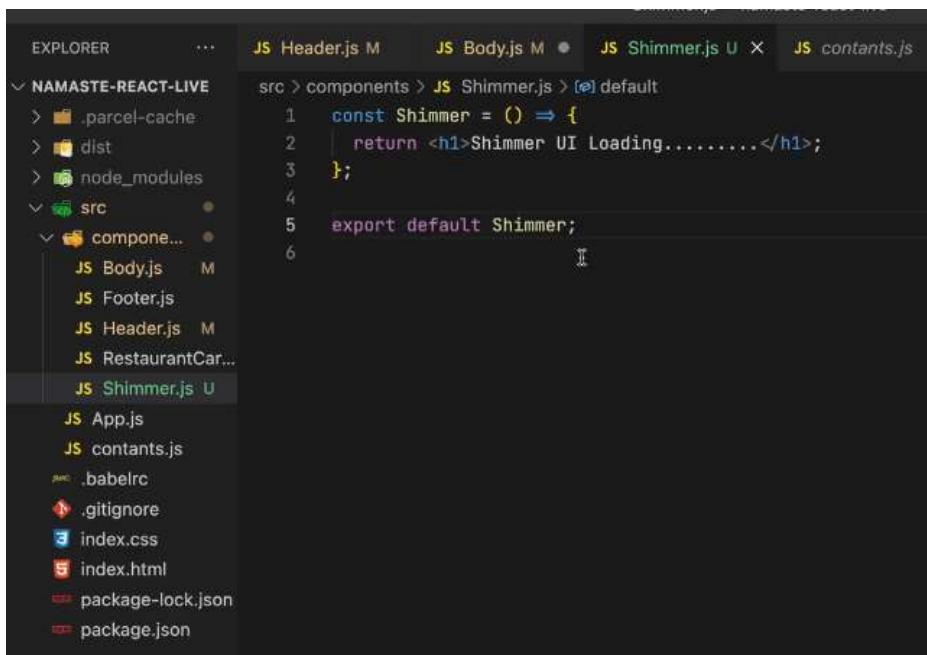
## Conditional Rendering

When we render the component, we need to either render Shimmer UI or render normal UI with API data.

```
//Conditional Rendering
//if restrauant is empty => shimmer Ui
// if restrauant has data => actual data UI
```

### How to code Shimmer UI??

Let us create a Shimmer UI component for our food cards



```
EXPLORER      ... JS Header.js M   JS Body.js M ● JS Shimmer.js U X JS contants.js
└ NAMASTE-REACT-LIVE
  > .parcel-cache
  > dist
  > node_modules
  < src
    > components
      > Shimmer.js U
      JS Body.js M
      JS Footer.js
      JS Header.js M
      JS RestaurantCar...
      JS Shimmer.js U
      JS App.js
      JS contants.js
    .babelrc
    .gitignore
    index.css
    index.html
    package-lock.json
    package.json
```

```
src > components > JS Shimmer.js > [e] default
1  const Shimmer = () => {
2    return <h1>Shimmer UI Loading.....</h1>;
3  };
4
5  export default Shimmer;
```

Now let us see where to put this Shimmer UI.

We can write some conditions before rendering the Body component using ternary operator like:

It's a JS so write inside {}

```
Header.js M JS Body.js 1, M X JS Shimmer.js U JS c  
c > components > JS Body.js > ...  
You, 1 second ago | 1 author (You)  
1 import { restaurantList } from "../contents";  
2 import RestaurantCard from "./RestaurantCard";  
3 import { useState, useEffect } from "react";  
4 import Shimmer from "./Shimmer"; You, 1 sec  
5
```

We load Shimmer like this

```
return restaurants.length === 0 ? (  
  <Shimmer />  
) : (  
  <>  
    <div className="search-container">...  
    </div>  
    <div className="restaurant-list">  
      {restaurants.map((restaurant) => (  
        <RestaurantCard {...restaurant.data} key=
```

### Let us now Solve the Search bar issue

At any given point in time either I want my Original Restro list or I Want Filtered Data list of resto based on my search query.

Whenever I am filtering I will use Original data and In the UI I will show Filtered Restro  
So We make a state for filtered resto

```
const [filteredRestaurants, setFilteredRestaurants] = useState([]);
```

We make a state for allRestros

```
const [allRestaurants, setAllRestaurants] = useState([]);
```

Now on making API call we populate allRestaurants state

```
async function getRestaurants() {  
  const data = await fetch(  
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.935192  
  );  
  const json = await data.json();  
  console.log(json);  
  // Optional Chaining  
  setAllRestaurants([json?.data?.cards[2]?.data?.data?.cards]);  
}
```

Now we render Filtered Data in UI

For initial Render it will show error because filteredData is empty and we are using it in our UI and using map function in it so for the initial render we set the filtered data also as API

data so that for first time when there is no search query all API data is showed in UI.

```
async function getRestaurants() {
  const data = await fetch(
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lon=77.6164308"
  );
  const json = await data.json();
  console.log(json);
  // Optional Chaining
  setAllRestaurants(json?.data?.cards[2]?.data?.data?.cards);
  setFilteredRestaurants(json?.data?.cards[2]?.data?.data?.data?.cards);
}
```

```
return restaurants.length === 0 ? (
  <Shimmer />
) : (
  <>
    <div className="search-container">...
    </div>
    <div className="restaurant-list">
      {filteredRestaurants.map((restaurant) => {
        return (
          <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
        );
      })}
    </div>
  </>
);
};
```

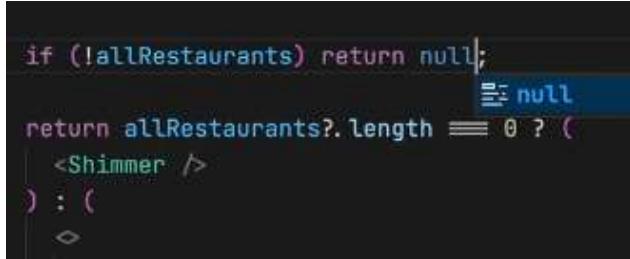
Now we modify our FilterData function and filter from allRestaurants state variable

```
return filteredRestaurants.length === 0 ? (
  <Shimmer />
) : (
  <>
    <div className="search-container">
      <input
        type="text"
        className="search-input"
        placeholder="Search"
        value={searchText}
        onChange={(e) => {
          setSearchText(e.target.value);
        }}
      />
      <button
        className="search-btn"
        onClick={() => {
          //need to filter the data
          const data = filterData(searchText, allRestaurants);
          // update the state - restaurants
          setRestaurants(data);
        }}
      >
```



```
<div className="restaurant-list">
  {filteredRestaurants.map((restaurant) => {
    return (
      <RestaurantCard {...restaurant.data} key={restaurant.data.id} />
    );
  })
</div>
```

To handle error of allRestro.length == undefined we can do optional chaining or we can do something like



```
if (!allRestaurants) return null;
return allRestaurants?.length === 0 ? (
  <Shimmer />
) : (
  <>
```

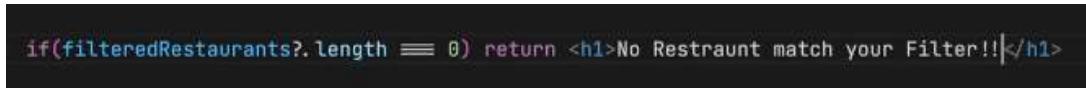
The below code says "If I don't have my restro, don't render my component"  
We can also pass any JSX instead of "null"



```
// not render component
if (!allRestaurants) return null;
```

We show shimmer if our Allrestro.length === 0 but what do to if filterRestro.length === 0 ??

We should show something like, "no result found", How to do it??



```
if(filteredRestaurants?.length === 0) return <h1>No Restraunt match your Filter!!</h1>
```

We do above thing, outside component return statement.

**Now let us handle our lowercase and uppercase in our search query so that if user writes lowercase in query it automatically matches if our query is there in uppercase**

### How to do it?

Convert both to lowercase or upperCase



```
> "Roti" === "roti"
< false
> "Roti".toLowerCase() === "roti".toLowerCase()
< true
```

```

function filterData(searchText, restaurants) {
    // 8 restraint list -> filtered rest with "King"
    const filterData = restaurants.filter((restaurant) =>
        restaurant?.data?.name?.toLowerCase()?.includes(searchText.toLowerCase())
    );

    return filterData;
}

```

### Let us make Login, Logout Button in our Header

When I am logged IN, I should show Logged out button

When I am not logged IN, I should show Log In Button.

```

    <Title />
    <div className="nav-items">
      <ul>
        <li>Home</li>
        <li>About</li>
        <li>Contact</li>
        <li>Cart</li>
      </ul>
    </div>
    <button>Login</button>
    <button>Logout</button>
  </div>
};

export default Header;

```

Let us say we make an AuthUser function in which we call a API which gives us either Success/Failure so when Success we show Log Out button, when failure we show Log In button.

```

const authenticateUser = () => [
  // API call to check authentication
  return true;      You, 1 second ago *
];

```

**We can write JS inside {} but we cannot write below code inside {}**

```
</div>
{
  a=10;
  console.log(); You, no
}
<button>Login</button>
<button>Logout</button>
</div>
;

import default Header;
```

### We can only write JS expressions inside {}

The above code has a = 10 as statement 1, console.log(a) as statement 2

But to write above code in valid way and make it as Expression we can do something like this

```
{
  //JS Expression & Statement
  [(a = 10), console.log(a)]
}
```

Now we will use Ternary Operator to load Log In, Log out button

We can write if-else but it's a statement not a expression, Ternary operator is an expression so we use it inside {}

Let us create a state for that

```
const [isLoggedIn, setIsLoggedIn] = useState(false);
```

```
const Header = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(true);

  return [
    <div className="header">
      <Title />
      <div className="nav-items">
        <ul>
          <li>Home</li>
          <li>About</li>
          <li>Contact</li>
          <li>Cart</li>
        </ul>
        <div> You, 6 days ago • chapter-05 </div>
      </div>
      {isLoggedIn ? <button>Logout</button> : <button>Login</button>}
    </div>
  ];
};
```

Let us making it toggle button

```
> div>
  {isLoggedIn ? (
    <button onClick={() => setIsLoggedIn(false)}>Logout</button>
  ) : (
    <button onClick={() => setIsLoggedIn(true)}>Login</button>
  )}
)
```

## Where is Diffing algorithm Written in React??

Its written in react core  
But executed by ReactDOM, Its same in React-native also

# Session 9

useEffect is called after our component is rendered  
Two parameters it takes, callback function and dependency array  
Callback function is called after our render  
If we have an empty array [ ], callback will be called only after first render that is only one time  
If we put [searchText] inside array then useEffect will be called on change of state of searchText

## What if we don't have dependency array [ ] inside useEffect??

It means useEffect is not dependent on anything so it will show its default behaviour so everytime render happens, useEffect will be called after each render of component.  
Each time a state changes, our component will re-render and our useEffect will be called

```
useEffect(() => {
  console.log("useEffect");
});      You, 1 second ago
```

**Never Create a component inside a component in react  
Keep it in the top level, otherwise rendering issue will come**

### Few Important things about hooks

1. Never write a useState hook inside if-else condition. If-else creates its own variable which creates inconsistency for react.

```
if(){}
const [searchText, setSearchText] = useState("");
```

2. Never write it inside a for-loop because each time for loop runs, new variable will be created which creates confusion and inconsistency for react.

```
for(){}
```

```
forQ{  
  const [searchText, setSearchText] = useState("");  
}  
  You, 1 second ago • Uncommitted changes
```

- useState is given to us by react so that we can create local variables inside functional component . So never write useState outside component.  
Always create state variables inside our functional component.

```
const [searchText, setSearchText] = useState("");  
  
const Body = () => {  
  const [allRestaurants, setAllRestaurants] = useState([]);  
  const [filteredRestaurants, setFilteredRestaurants] = useState([]);  
  const [searchText, setSearchText] = useState("");  
  
  useEffect(() => {  
    getRestaurants();  
  }, [ ]);  
  
  async function getRestaurants() {  
    const data = await fetch(  
      "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351  
    );  
    const json = await data.json();  
    console.log(json);  
    // Optional Chaining  
    setAllRestaurants(json?.data?.cards[2]?.data?.data?.cards);  
  }  
};
```

### Can we use more than one useEffect??

Yes, we can create as many useEffect according to our use cases, say some useEffect work on changing searchText, some work on resto list so yes we can do that.

### How to keep our images locally?

We can create a folder inside src, called as asset and make an folder img inside asset and now put your images there.



### How to import local images to our code?

We can import it like normal component as given below

```
import React from 'react';  
import Logo from "../assets/img/foodVilla.png";
```

Now we can use it as

```
<a href="/">  
  <img className="logo" alt="logo" src={Logo} />  
</a>
```

We can use any type of image, we can also load it via CDN, swiggy also uses CDN to host its image. Because CDN is faster, it optimises our image and makes them render faster,



### What Is a CDN for Images?

A CDN for images specializes in **optimizing, converting, and delivering images worldwide.** They are like an API network that modifies images to **distribute them faster from the servers to your visitor's screen.**

## Image URL from an image CDN



### How to implement shimmer?

We can make one shimmer for ourself or we can use a npm package for it.

**Be conscious about what packages you are importing in your code**

**There are 1000 of packages, you should be cautious of what packages you are importing in your code, you should not use package for something you can create easily**

### Let us make our own shimmer for our food cards

What is shimmer? Its an dummy card

We need to create similar kind of view, similar kind of card as food card and apply some css to it

A screenshot of a code editor showing the Shimmer.js component code. The code defines a functional component named Shimmer. It returns a single element: a div with the class "shimmer-card". The code is written in JSX and includes imports for React and styled-components.

CSS (same as our card just that add some background colour)

```

.shimmer-card {
  width: 200px;
  height: 200px;
  background: lightgrey;
  margin: 20px;
}

```

Now we need to load these shimmers equal to number of our food cards or equal to our whole screen.

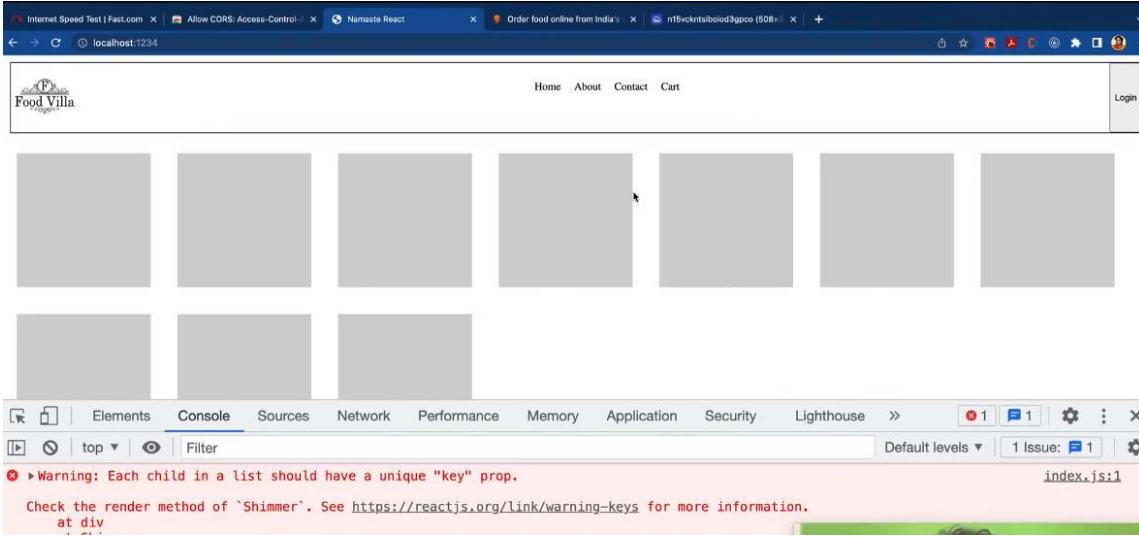
`Array(10)` will create an array of 10 element, `.fill` will fill each element with " " and `.map` will map with each element of array and runs shimmer card for each element.

```

const Shimmer = () => {
  return (
    <div className="restaurant-list">
      {Array(10)
        .fill(" ")
        .map((e, index) => (
          <div key={index} className="shimmer-card"></div>
        ))}
    </div>
  );
}

export default Shimmer;

```



### When should be import a big package?

When, things get complex like in Fintech company has big big forms and each form has its different regex for validation etc etc. In that case we can instead of making validation, error checking for each form, we can use something like **Formik**.

**Always use Formik for forms while making forms in react**

We cannot make our babel so we import babel so use packages when needed only.

<https://formik.org> ::

## Formik: Build forms in React, without the tears

Formik is the world's most popular open source form library for React and React Native. Get Started · GitHub. Declarative. Formik takes care of the repetitive ...

### Tutorial

Formik is a small group of React components and hooks for ...

### Overview

Tutorial - Validation - Basic Example - React Native - ...

### Basic Example

This example demonstrates how to use Formik in its most basic way ...

### Validation

Formik is designed to manage forms with complex validation ...

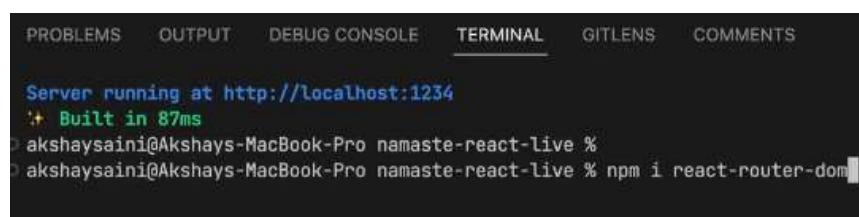
[More results from formik.org »](#)

# React Router Dom

We will be making different routes/ different pages for our navbar elements

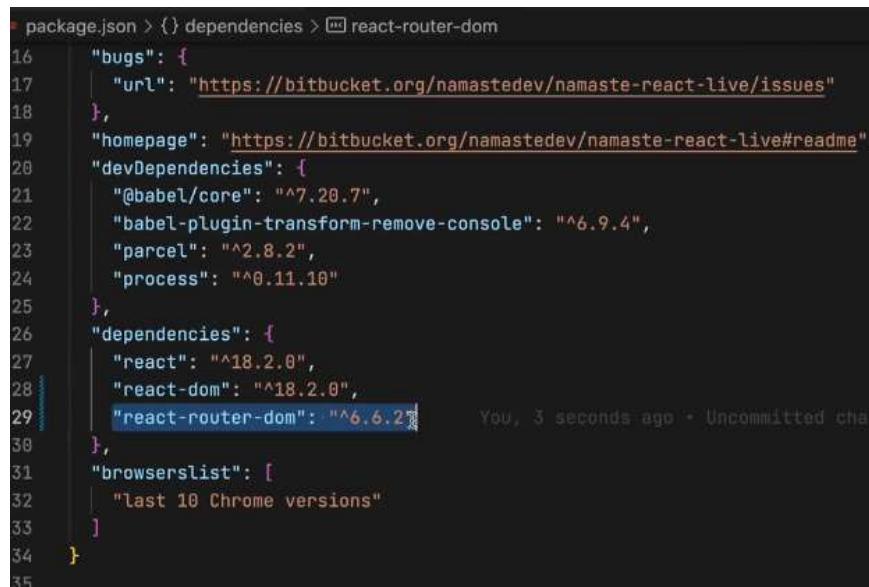
We will use **V6 version (latest)** of react router.

### How to install?



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS COMMENTS

Server running at http://localhost:1234
+ Built in 87ms
akshaysaini@Akshays-MacBook-Pro namaste-react-live %
akshaysaini@Akshays-MacBook-Pro namaste-react-live % npm i react-router-dom
```



```
package.json > {} dependencies > react-router-dom
16   "bugs": {
17     "url": "https://bitbucket.org/namastedev/namaste-react-live/issues"
18   },
19   "homepage": "https://bitbucket.org/namastedev/namaste-react-live#readme"
20   "devDependencies": {
21     "@babel/core": "^7.20.7",
22     "babel-plugin-transform-remove-console": "^6.9.4",
23     "parcel": "^2.8.2",
24     "process": "^0.11.10"
25   },
26   "dependencies": {
27     "react": "18.2.0",
28     "react-dom": "18.2.0",
29     "react-router-dom": "6.6.2"      You, 3 seconds ago + Uncommitted changes
30   },
31   "browserslist": [
32     "Last 10 Chrome versions"
33   ]
34 }
35
```

Let us use it now, we will make about US page, contact page and see how router works

The screenshot shows the VS Code interface with the Explorer, JS App.js, JS Body.js, and JS About.js tabs. The About.js file is open, displaying its code:

```
const About = () => {
  return (
    <div>
      <h1>About Us Page</h1>
      <p> This is the Namaste React Live Course Chapter 07 - Finding the Path </p>
    </div>
  )
}

export default About;
```

Now we need to make a route such that we put /about in our URL, it takes us to our about page.

Let us go to **App.js** and **import react router dom**

1. Import **createBrowserRouter** from react-router-dom

```
import { createBrowserRouter } from "react-router-dom";
```

2. Create **appRouter** now (**Always create this, below our component because our code runs in a sequence so it will not be able to find the components we are using inside appRouter**)

Inside it, we tell if  
we click /about -> go to about page  
For / -> go to home page

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
  },
  {
    path: "/about",
    element: <About />,
  },
]);
```

This will not work currently, we need to provide this appRouter to our app

3. Use **RouterProvider**

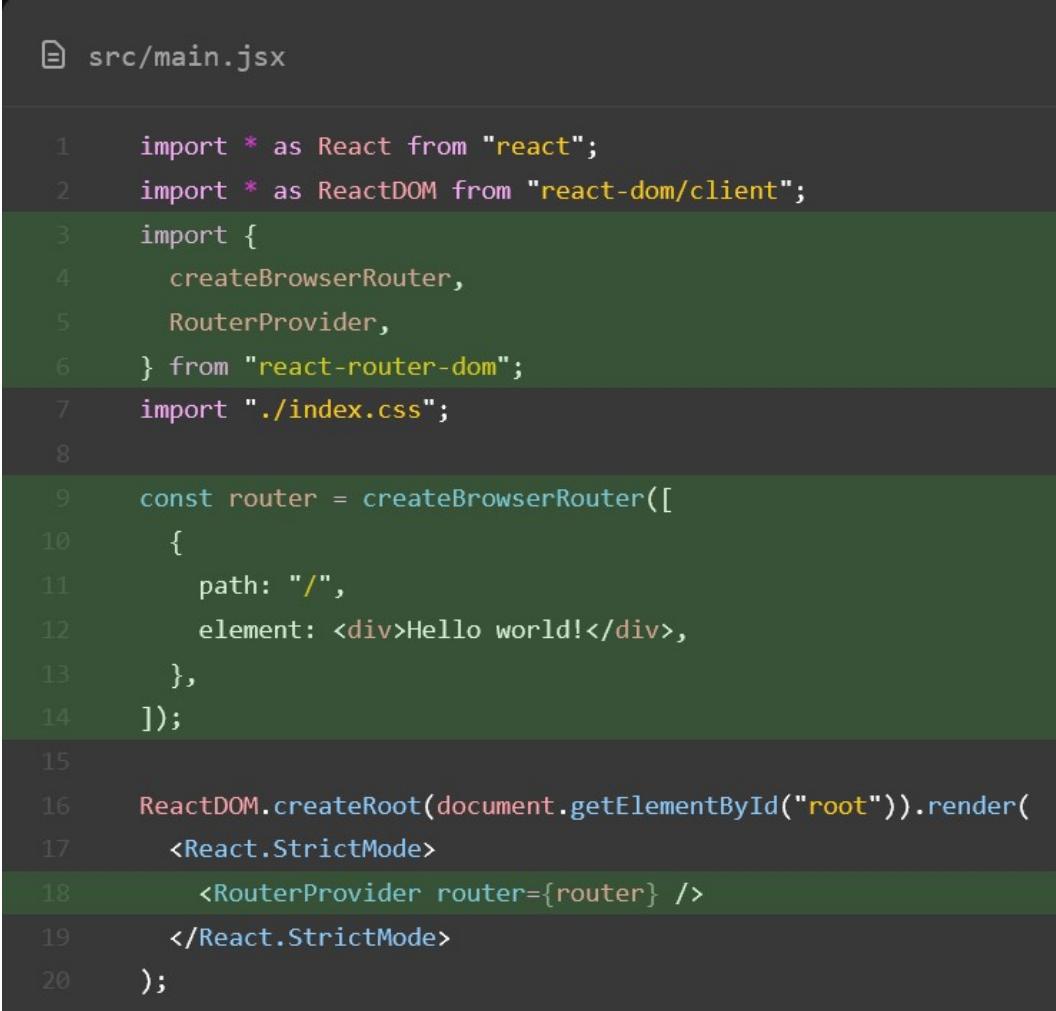
```
import { createBrowserRouter, RouterProvider } from "react-router-dom";
```

Now render this inside app.js instead of AppLayout

```
const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(<RouterProvider router={appRouter} />);    You, 1 sec
```

Now whatever appRouter renders, will get rendered



The screenshot shows a code editor with a dark theme. The file is named 'src/main.jsx'. The code imports React, ReactDOM, and createBrowserRouter from react-router-dom. It defines a router object with a single path '/' pointing to a 'Hello world!' component. This router is then passed to ReactDOM.createRoot to render it into the 'root' element.

```
src/main.jsx

1 import * as React from "react";
2 import * as ReactDOM from "react-dom/client";
3 import {
4   createBrowserRouter,
5   RouterProvider,
6 } from "react-router-dom";
7 import "./index.css";
8
9 const router = createBrowserRouter([
10   {
11     path: "/",
12     element: <div>Hello world!</div>,
13   },
14 ]);
15
16 ReactDOM.createRoot(document.getElementById("root")).render(
17   <React.StrictMode>
18     <RouterProvider router={router} />
19   </React.StrictMode>
20 );
```

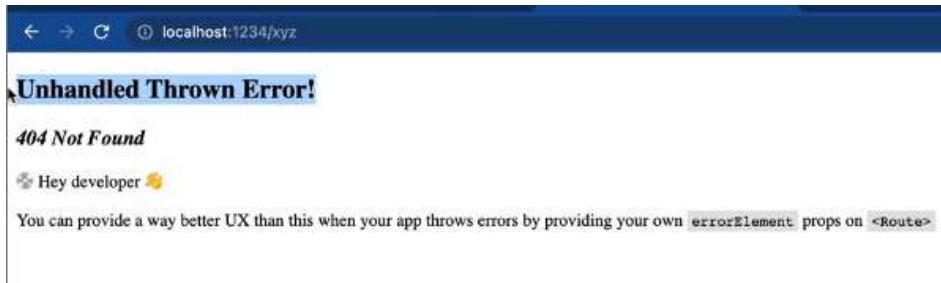
Now we go to our **/about** we see our **about us page**



What If we write /anythingElse, what will happen?

It will throw an error page which is given by reactr-router-dom itself, Our console is still not

red so its great UX.



## How to make our own Error Page?

We will make another component for it.

A screenshot of a code editor showing a file named 'Error.js'. The code defines a functional component 'Error' that returns a div containing an h1 and an h2. The code editor interface includes tabs for 'App.js', 'Body.js', 'About.js', and 'Error.js', with 'Error.js' currently active. The left sidebar shows the project structure with files like 'index.css', 'index.html', and 'package.json'.

Now we pass this information to our appRouter in app.js

```
import { Error } from "react-components/Error";
import { createBrowserRouter, RouterProvider } from "react-router-dom";

const AppLayout = () => {
  return (
    <>
      <Header />
      <Body />
      <Footer />
    </>
  );
};

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,          You, 1 second ago + Uncommitted changes
    errorElement: <Error />,
  },
  {
    path: "/about",
    element: <About />,
  },
]);

```

We want to show more information about this error, we can use [useRouterError hook from react-router-dom](#)

```
import { useRouteError } from "react-router-dom";

const err = useRouteError();
console.log(err);
```

```
ErrorResponse {status: 404, statusText: 'Not Found', internal: true, data: 'Error: No route matches URL "/hjhj"', error: Error: Error: No route mat
ches URL "/hjhj"
  at getInternalRouterError (http://localhost:1234/index.7271...)} Error.js:5
  data: "Error: No route matches URL \"/hjhj\""
> error: Error: No route matches URL "/hjhj" at getInternalRouterError (http://localhost:1234/index.7271efb6.js:33124:57) at createRouter (ht
internal: true
status: 404
statusText: "Not Found"
> [[Prototype]]: Object
```

This hooks tell us what type of error we are having.

```

> components > JS Error.js > [x] Error
  import { useRouteError } from "react-router-dom";

  const Error = () => {
    const err = useRouteError();
    console.log(err);
    return (
      <div>
        <h1>Oops!!</h1>
        <h2>Something went wrong!!</h2>
        <h2>[err.status + " : " + err.statusText]</h2>
      </div>
    );
  };

  export default Error;

```

Now say we want to click "About in Navbar" and it takes us to "/about"

How to do it?

Home About Contact Cart

Can we use <a> Tag?

Yes, but there is a **problem with <a> tag** and that is, it  **reloads the whole page** but we are making **single page application** in which only specific parts renders.

## What is single page applications?

A single page application is a website or web application that **dynamically rewrites a current web page with new data from the web server, instead of the default method of a web browser loading entire new pages.**

A single page application is a single page (hence the name) **where a lot of information stays the same and only a few pieces need to be updated at a time.**

For example, when you browse through your email, you'll notice that not much changes during navigation — the sidebar and header remain untouched as you go through your inbox.

The SPA only sends what you need with each click, and **your browser renders that information**. This is different than a traditional page load where the server rerenders a full page with every click you make and sends it to your browser.

This piece-by-piece, client-side method makes load times much faster for users. It also lessens the amount of information a server has to send and makes the whole process a lot more cost-efficient — a win-win scenario for users and businesses.

There are **2 types of routing:**

### 1. Client-side routing

**It is what we making in our react app, loading different component based on our**

route.

## 2. Server-side routing

It's a way where all our routes come from our server

We will **focus on client-side routing for now**

We could have used  [also but it reloads whole page](#)

```
<li>Home</li>
| <a href="/about">
| | <li>About</li>
| </a>| You, I se
```

So **react-router-dom** gives us **{Link}**

```
import { Link } from "react-router-dom";
```

Its same as  [in syntax but instead of "href" we use "to"](#)

```
<Link to="/about">
| <li>About</li>
</Link>
```

```

const Header = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  return [
    <div className="header">
      <Title />
      <div className="nav-items">
        <ul>
          <Link to="/">
            <li>Home</li>
          </Link> You, 1 second ago + Uncommitted changes
          <Link to="/about">
            <li>About</li>
          </Link>
          <li>Contact</li>
          <li>Cart</li>
        </ul>
      </div>
      {isLoggedIn ? (
        <button onClick={() => setIsLoggedIn(false)}>Logout</button>
      ) : (
        <button onClick={() => setIsLoggedIn(true)}>Login</button>
      )}
    </div>
  ];
};

export default Header;

```

We can also put it inside our "li" its just like <a> tag

```

<li>
  <Link to="/">Home</Link>
</li>

```

Although we are not using <a>, we are using <Link/> but if we check our DOM we see At the end of the day, Link component also uses <a> behind the scenes and react-router-dom just keep tracks of the Link.

```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body> == $0
    <div id="root">
      <div class="header"> flex
        <a href="/">...</a>
        <div class="nav-items">
          <ul> flex
            <li>...</li>
            <a href="/about">...</a>
            <li>Contact</li>
            <li>Cart</li>
          </ul>
        </div>
      </div>
    </div>
```

## Nested Routing

We see when we go to About US page, Our header footer is not there so we want to keep our Header and Footer intact irrespective of Our About us, contact page

---

### About Us Page

This is the Namaste React Live Course Chapter 07 - Finding the Path ↗

We need to change our Router Config

We go to app.js

We need to make <About/> as children of <AppLayout/>

We use something known as **children**

```
const AppLayout = () => {
  return (
    <>
      <Header />
      <About /> // if path is /about
      <Body /> // if path is /
      <Footer />
    </>
  );
};
```

```

const AppLayout = () => {
  return (
    <>
      <Header />
      <About /> // if path is /about
      <Body /> // if path is /
      <Footer />
    </>
  );
};

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/about",
        element: <About />,
      },
    ],
  },
]);

```

Let us create a **Contact Page** also

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: Shows the project structure under "NAMASTE-REACT-LIVE". It includes ".parcel-cache", "dist", "node\_modules", and a "src" folder containing "assets" and "components". Inside "components", there are files: "About.js", "Body.js", "Contact.js" (which is currently selected), "Error.js", "Footer.js", "Header.js", "RestaurantCard.js", and "Shimmer.js".
- Code Editor Area:**
  - File tab: "JS Contact.js U" (with a red error icon).
  - Content pane:

```

const Contact = () => {
  return <h1>Contact Us Page</h1>;
};

export default Contact;

```

```
const AppLayout = () => {
  return (
    <>
      <Header />
      <About /> // if path is /about
      <Body /> // if path is /
      <Contact /> // /contact
      <Footer />
    </>
  );
};
```

Now we create one more children

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/contact",
        element: <Contact />,
      },
    ],
  },
]);
```

But this will not work as our AppLayout currently looks like

```
const AppLayout = () => {
  return (
    <>
      <Header />
      <About />
      <Body />
      <Contact />
      <Footer />
    </>
  );
};
```

So, it will render everybody we don't want it

So we **do not render anything between Header and Footer**

```
const AppLayout = () => {
  return (
    <>
      <Header />
      {/* { Outlet } */}
      <Footer />
    </>
  );
};
```

We use **Outlet** from react-router-dom Which will be **filled by children configuration according to the route**

We also pass children for our body component

```
const AppLayout = () => {
  return (
    <>
      <Header />
      {/* { Outlet } */}
      <Outlet />
      <Footer />
    </>
  );
};

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/",
        element: <Body />,
      },
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/contact",
        element: <Contact />,
      },
    ],
  },
]);
```

Let us make our Contact also a Link in Header.js

```
<Link to="/contact">
| <li>Contact</li> abc C
</Link> abc C
```

## Dynamic Routing/ Segments

We see on swiggy website, on clicking any resto our route dynamically changes

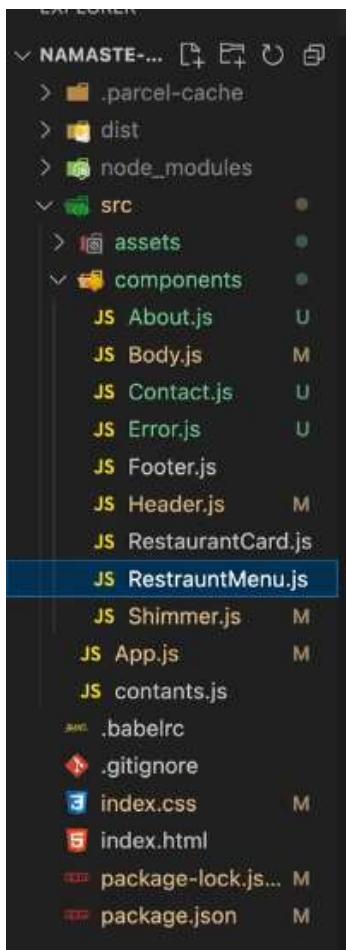


## How to create this Dynamic Routing

What we want is, if we click Burger King, It should take us to Burger king page and our route should also change accordingly



Let us **make a Component for resto details**



```
const RestaurantMenu = () => {
  return (
    <div>
      <h1>Restraunt id: 123</h1>
      <h2>Namaste</h2>
    </div>
  );
};

export default RestaurantMenu;
```

Change it in **appRouter** path also

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/",
        element: <Body />,
      },
      {
        path: "/about",
        element: <About />,
      },
      {
        path: "/contact",
        element: <Contact />,
      },
      {
        path: "/restaurant/:id",
        element: <RestaurantMenu />,
      },
    ],
  },
]);

```

:id is Dynamic, we can give anything like /1234 and it loads RestroMenu

We can **useParams Hook** from react router dom

```
import { useParams } from "react-router-dom";

const RestaurantMenu = () => {
  const params = useParams();
  console.log(params);

  return [
    <div>
      <h1>Restraunt id: 123</h1>
      <h2>Namaste</h2>
    </div>
  ];
};

export default RestaurantMenu;
```

① localhost:1234/restaurant/1234



We can use this id also inside our component like

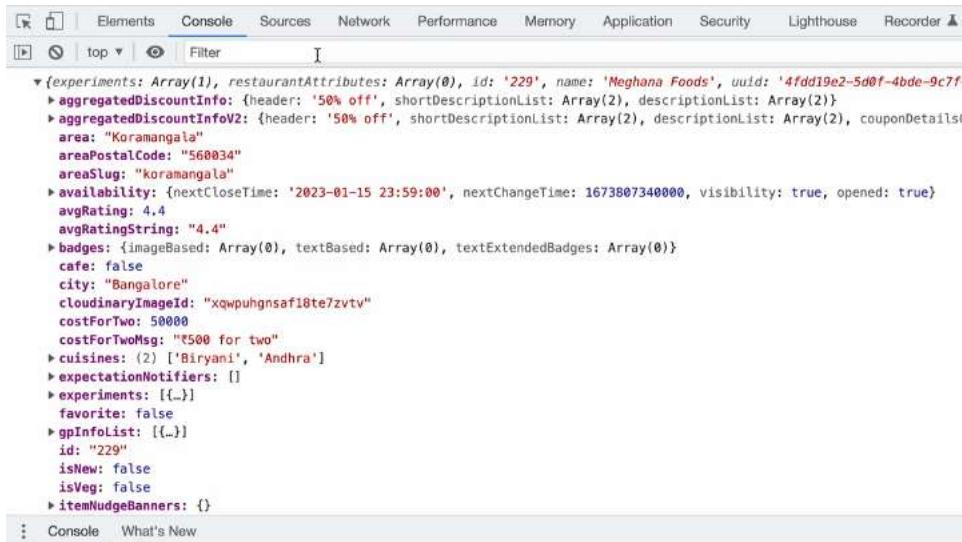
```
import { useParams } from "react-router-dom";

const RestaurantMenu = () => {
  const params = useParams();
  const { id } = params;
  console.log(params);

  return (
    <div>
      <h1>Restraunt id: {id}</h1>
      <h2>Namaste</h2>
    </div>
  );
};

export default RestaurantMenu;
```

Let us make an API call for the Restro details page using API details of any random Restro from swiggy



```
async function getRestaurantInfo() {
  const data = await fetch(
    "https://www.swiggy.com/dapi/menu/v4/full?lat=12.9351929&lng=77.62448069999999&menuId=229"
  );
  const json = await data.json();
  console.log(json.data);
  setRestauraunt(json.data);
}

return (
  <div>
    <h1>Restraunt id: {resId}</h1>
    <h2>{restaurant.name}</h2>
    <img src={IMG_CDN_URL + restaurant.cloudinaryImageId} />
    <h3>{restaurant.area}</h3>
    <h3>{restaurant.city}</h3>
    <h3>{restaurant.avgRating} stars</h3>
    <h3>{restaurant.costForTwoMsg}</h3>
  </div>
);
}

export default RestaurantMenu;
```

Let us Render the menu also inside our page.

We see that the data is not an Array of object now, Data is in object form that is key-value pair and in key we have ID, in value we have data we want to show

To render object we use `Object.value` which will give us Array of objects now with the list of menu items.

Now we can loop onto it

```

return (
  <div>
    <div>
      <h1>Restraunt id: {resId}</h1>
      <h2>{restaurant.name}</h2>
      <img src={IMG_CDN_URL + restaurant.cloudinaryImageId} />
      <h3>{restaurant.area}</h3>
      <h3>{restaurant.city}</h3>
      <h3>{restaurant.avgRating} stars</h3>
      <h3>{restaurant.costForTwoMsg}</h3>
    </div>
    <div>
      <h1>Menu</h1>
      <ul>
        {Object.values(restaurant.menu.items).map((item) => (
          <li key={item.id}>{item.name}</li>
        ))}
      </ul>
    </div>
  </div>
);

export default RestaurantMenu;

```

## Menu

- Plain Rice
  - Paneer 65
  - Chilly Paneer
  - Golden Baby Corn
  - Chilly Gobi
  - Gobi 65
  - Mushroom 65
  - Chilly Mushroom
  - Chilly Babycorn
  - Gobi Manchurian
  - Paneer Manchurian
  - Baby Corn Manchurian
  - Mushroom Manchurian
  - Chilly Chicken (Boneless)
  - Chilly Chicken (Andhra Style)
  - Chicken 65
  - Meghana Chicken 555
  - Lemon Chicken
  - Chicken Kabab
  - Chicken Lollypop
- 

We can put CSS onto it to make it look better.

## Restraunt id: 54321

### Meghana Foods



Koramangala

Bangalore

4.4 stars

₹500 for two

### Menu

- Plain Rice
- Paneer 65
- Chilly Paneer
- Golden Baby Corn
- Chilly Gobi
- Gobi 65
- Mushroom 65
- Chilly Mushroom
- Chilly Babycorn
- Gobi Manchurian
- Paneer Manchurian
- Baby Corn Manchurian
- Mushroom Manchurian
- Chilly Chicken (Boneless)
- Chilly Chicken (Andhra Style)
- Chicken 65
- Meghana Chicken 555
- Lemon Chicken
- Chicken Kabab
- Chicken Lollypop
- Chicken Pakoda
- Pepper Chicken
- Chicken Fry
- Nati Chicken Fry
- Chicken Maharaja
- Chicken Rayalaseema
- Mutton Fry
- Today's Andhra Spl
- Mutton Kheema Fry
- Mutton Pepper Fry
- Chilly Fish
- Apollo Fish
- Spl Fish Fry
- Single Fish Fry
- Chilli Prawns
- Prawns Fry
- Chicken Boneless Curry
- Butter Chicken Curry
- Chicken Curry
- Mutton Curry
- Fish Curry
- Prawns Curry
- Egg Masala Curry

Till the API is called let us show a **shimmer** and using Early rendering by writing this above our component

```
if (!restaurant) {
  return <Shimmer> />;
}
```

```
import { IMG_CDN_URL } from "../contants";
import Shimmer from "./Shimmer";

const RestaurantMenu = () => {
  // how to read a dynamic URL params
  const { resId } = useParams();
  // Use proper names
  const [restaurant, setRestaurant] = useState(null);

  useEffect(() => {
    getRestaurantInfo();
  }, []);

  async function getRestaurantInfo() {
    const data = await fetch(
      `https://www.swiggy.com/dapi/menu/v4/full?lat=12.9351929&lng=77.62448069999999&menuId=${resId}`
    );
    const json = await data.json();
    console.log(json.data);
    setRestaurant(json.data);
  }
}
```

```
return (
  <div class="menu">
    <div>
      <h1>Restraunt id: {resId}</h1>
      <h2>{restaurant.name}</h2>
      <img src={IMG_CDN_URL + restaurant.cloudinaryImageId} />
      <h3>{restaurant.area}</h3>
      <h3>{restaurant.city}</h3>
      <h3>{restaurant.avgRating} stars</h3>
      <h3>{restaurant.costForTwoMsg}</h3>
    </div>
    <div>
      <h1>Menu</h1>
      <ul>
        {Object.values(restaurant?.menu?.items).map((item) => (
          <li key={item.id}>{item.name}</li>
        ))}
      </ul>
    </div>
  </div>
);
```

Or we can do something like

```
return (!restaurant) ? <Shimmer /> :(
  <div class="menu">
    <div>
      <h1>Restraunt id: {resId}</h1>
      <h2>{restaurant.name}</h2>
      <img src={IMG_CDN_URL + restaurant.cloudinaryImageId} />
      <h3>{restaurant.area}</h3>
      <h3>{restaurant.city}</h3>
      <h3>{restaurant.avgRating} stars</h3>
      <h3>{restaurant.costForTwoMsg}</h3>
    </div>
    <div>
      <h1>Menu</h1>
      <ul>
        {Object.values(restaurant?.menu?.items).map((item) => (
          <li key={item.id}>{item.name}</li>
        ))}
      </ul>
    </div>
  </div>
);
```

Now just by changing our Route ID, we can load a new Restro from API

**Restraunt id: 642768****Menu****Gramin****Koramangala****Bangalore****4.4 stars****₹350 for two**

- Daal Ghee Tadka
- Gulka Milk Shake (300ml)
- Jaleera (300ml)
- Corn Soup
- Daal Makhanai
- Kesar Lassi (300ml)
- Lassi Salt (300ml)
- Mix Veg Soup
- Double Tadka Daal
- Lassi Sugarfree Khus (300ml)
- Tomato Soup
- Lassi Sugarfree Mango (300ml)
- Rajma Masala
- Lassi Sugarfree Rose(300ml)
- Babycorn Paneer Allahabadi
- Lassi Sweet (300ml)
- Boondi Raita
- Masala Butter Milk (Spicy Masala) (300ml)
- Bhindi Rajasthani (starter)
- Nimbu Ki Shikanji (300ml)
- Fried Chillies (6 Pcs)
- Rose Lassi (300ml)
- Harabbara Kebab (6 Pcs)
- Garlic Chutney
- Thandai (300ml)
- Paneer Koliwada
- Green Salad
- Vanilla Lassi (300ml)
- Tawa Corn Peas
- Gulab Jamun (2 Pcs)
- Khichi Papad
- Lauki Ka Halwa
- Papad Roasted
- Tawa Karela
- Pineapple Raita
- Bharwan Paneer Tikka (6 Pcs)
- Lasuni Paneer Dhuadar (6 Pcs)
- Plain Curd
- Stuffed Tandoori Aloo
- Veg Raita
- Veg Sheekh Kabab
- Corn Pulao
- Mumbai Pulao

**Now Let us make the functionality that if we click on any Food card, it should take us to details of that restro with its ID being in the Route automatically.**



Search

Search

**Donde Biryani Mane**

Biryani, Kebabs

0.8 kms minutes

**Meghana Foods**

Biryani, Andhra, South Indian, North Indian, Chinese, Seafood

1.2 kms minutes

**Kannur Food Point**

Kerala, Chinese

3 kms minutes

**Leon's - Burgers & Wings (Leon Grill)**

American, Snacks, Turkish, Portuguese, Continental

1.2 kms minutes

**Roti Wala**

North Indian, Home Food, Thalis, Chinese, Punjabi, South Indian, Ice Cream

4 kms minutes

**Hotel Empire**

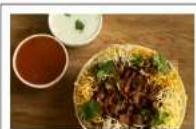
North Indian, Kebabs, Biryani

1.2 kms minutes

**Nagarjuna**

Andhra, South Indian, Biryani, Beverages, Desserts

0.8 kms minutes

**Biryani Pot**

North Indian, Biryani

1.8 kms minutes

**Paradise Biryani**

Biryani, Kebabs, North Indian, Hyderabadi

4.4 kms minutes

**Asha tiffins**

Indian, South Indian, Beverages

4 kms minutes



We go to Body.js and use Link in Restro Cards

```
> components > JS Body.js > ...
You, 1 second ago | 1 author (You)
import { restaurantList } from "../contents";
import RestaurantCard from "./RestaurantCard";
import { useState, useEffect } from "react";
import Shimmer from "./Shimmer";    You, 23 h
import {Link} from "react-router-dom"
```

```
<div className="restaurant-list">
  /* You have to write logic for NO restaurnt fount here */
  {filteredRestaurants.map((restaurant) => {
    return (
      <Link
        to={`/restaurant/${restaurant.data.id}`}
        key={restaurant.data.id}
      >    You, 1 second ago • Uncommitted changes
        <RestaurantCard {...restaurant.data} />
      </Link>
    );
  })
</div>
;

export default Body;
```

Search **[Donne Biriyani Mane](#)**[Biryani, Kebabs](#)[0.8 kms minutes](#)**[Meghana Foods](#)**[Biryani, Andhra, South Indian, North Indian, Chinese, Seafood](#)[1.2 kms minutes](#)**[Kannur Food Point](#)**[Kerala, Chinese](#)[3 kms minutes](#)**[Leon's - Burgers & Wings \(Leon Grill\)](#)**[American, Snacks, Turkish, Portuguese, Continental](#)[1.2 kms minutes](#)**[Roti Wala](#)**[North Indian, Hot Food, Thalis, Chir Punjabi, South Indian, Ice Cream](#)[4 kms minutes](#)**[Zaitoon](#)**[Arabian, Biryani, North Indian, Tandoor, Chinese, Juices](#)[4.6 kms minutes](#)**[Biryani Pot](#)**[North Indian, Biryani](#)[1.8 kms minutes](#)**[Paradise Biryani](#)**[Biryani, Kebabs, North Indian, Hyderabadi](#)[4.4 kms minutes](#)**[Asha tiffins](#)**[Indian, South Indian, Beverages](#)[4 kms minutes](#)**[McDonald's](#)**[Burgers, Beverage Cafe, Desserts](#)[1.2 kms minutes](#)

Now we click on any card we see

← → ⌂ localhost:1234/restaurant/425



[Home](#) [About](#) [Contact](#) [Cart](#)

Restraunt id: 425

Hotel Empire



Koramangala

Bangalore

4.1 stars

₹450 for two

## Menu

- Gobi Manchurian
- Pepper Chicken Dry
- Grilled Chicken
- Mutton Raan
- Paneer Tikka Masala
- Paneer Butter Masala
- Dal Fry
- Egg Masala
- Egg Omelette
- Chicken Manchurian Masala
- Chicken Masala
- Chicken Tikka Masala
- Chicken Tandoori Masala
- Kadai Chicken Gravy
- Mutton Brain Masala
- Pepper Mutton Masala
- Grilled Chicken Biryani
- Ghee Rice
- Lemon Rice
- Jeera Rice
- Curd Rice
- Tandoori Roti
- Butter Roti
- Kulcha
- Tandoori Naan
- Butter Naan
- Dry Chapati
- Kerala Parotta
- Malabar Parotta
- Ceylon Parotta
- Coin Parotta
- Wheat Parotta
- Empire Butter Parotta
- Veg Thali
- Veg Fried Rice
- Egg Fried Rice
- Chicken Fried Rice
- Mutton Fried Rice
- Prawns Fried Rice
- Veg Noodles
- Egg Noodles
- Chicken Noodles
- Arabian Dry Fruit

(Interview mai chipka dena) Modular, Cleaner, Maintainable, Testable, Readable, Resuable

# Session 10

React was class based component only since beginning, hooks and functional component is new.

Earlier there was not something like functional component.

Earlier React was all about classes but developers found code to be little messy and very big as compared to what functional component is now.

Class based comp was less maintainable and less understandable for new comers

If we write same thing in functional based comp and class based comp then code of functional comp will be very small as compared to class based component.

Class based component is no longer used

If we make new project, class based component do not come now

They are almost depreciated, But there still can be many companies which have old code in class based comp, so its important for us to learn class based comp to understand that old

project.

Lot of questions are asked from Class based component also in interviews.

## How to make nested routes??

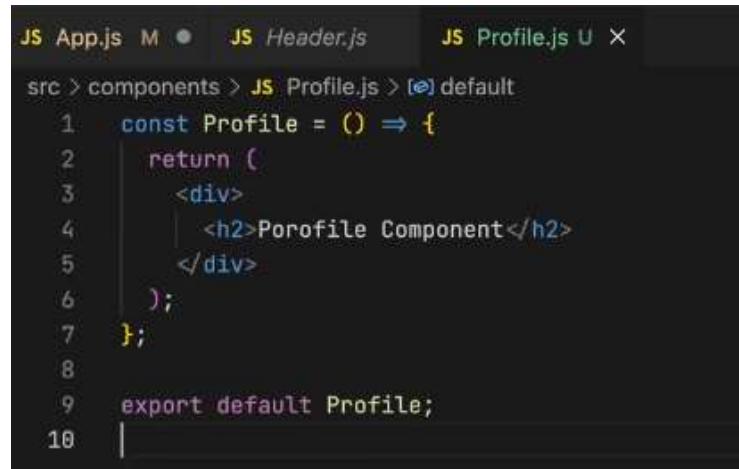
We can make children of children, but we do not write "\\" in the path

```
// parentPath/{path} => localhost:1244/about/profile
| 1 second ago × Uncommitted changes

const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/about",
        element: <About />,
        children: [
          {
            path: "profile", // localhost:1244/about/profile
            element: ...
          }
        ]
      },
    ]
  }
])
```

Now we write the profile component in element of children

Let us create Profile component



The screenshot shows a code editor with multiple tabs open. The active tab is 'Profile.js' under the 'src > components' directory. The code defines a functional component named 'Profile' that returns a simple HTML structure with a heading 'Porofile Component'. The code is numbered from 1 to 10.

```
src > components > Profile.js > (e) default
1  const Profile = () => {
2    return (
3      <div>
4        <h2>Porofile Component</h2>
5      </div>
6    );
7  };
8
9  export default Profile;
10 |
```

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <AppLayout />,
    errorElement: <Error />,
    children: [
      {
        path: "/about",
        element: <About />,
        children: [
          {
            path: "profile", // parentPath/{path}
            element: <Profile />,
          },
        ],
      },
      {
        path: "/",
        element: <Body />,
      },
      {
        path: "/contact",
        element: <Contact />,
      },
      {
        path: "/restaurant/:resId",
        element: <RestaurantMenu />,
      },
    ],
  ],
]);
```

Now we know children are rendered inside Outlet so We need to create an Outlet inside About.js

```
src > components > JS About.js > [e] About
You, now | 1 author (You)
1 import { Outlet } from "react-router-dom";
2
3 const About = () => {
4   return [
5     <div>
6       <h1>About Us Page</h1>
7       <p>
8         This is the Namaste React Live Course Chapter 07 - Finding the Path 🚀
9       </p>
10      <Outlet />      You, now + Uncommitted changes
11    </div>
12  ];
13};
14
15 export default About;
16
```

## Class Based Components

We will convert Profile component to class based component from functional component

Functional comp is a normal JS function at the end of the day  
Sameway, Class comp is a normal JS class at the end of the day

### How to make a class based component?

```
components > JS ProfileClass.js > 📁 Profile
  import React from 'react';

  class Profile extends React.Component {
    render() {
      return <h1> Profile Class Component </h1>;
    }
  }

  export default Profile;
```

### Most important part of a class based component is a **render** method

This is the only mandatory method for class components

### Render method returns some JSX

We export our function same as we did in functional component.

```
import React from "react";

class Profile extends React.Component {
  render() {
    return <h1> Profile Class Component </h1>;
  }
}

export default Profile;
```

### How to use class component in some other file?

Just import it and use it the samway as functional component

```

import { Outlet } from "react-router-dom";
import Profile from "./ProfileClass";

const About = () => {
  return (
    <div>
      <h1>About Us Page</h1>
      <p>
        This is the Namaste React Live Course Chapter
      </p>
      <Profile /> You, 10 seconds ago • Uncommitted
    </div>
  );
};

export default About;

```

### Why we use extends?

Because our component class is **inheriting** something from React.Component class which gives us some power

### Let us use our functional component and class component both together

```

import { Outlet } from "react-router-dom";
import ProfileFunctionalComponet from "./Profile";
import Profile from "./ProfileClass";

const About = () => {
  return [
    <div>
      <h1>About Us Page</h1>
      <p>
        This is the Namaste React Live Course Chapter
      </p>
      <ProfileFunctionalComponet />
      <Profile />
    </div>
  ];
};

export default About;

```

Output is



## About Us Page

This is the Namaste React Live Course Chapter 07 - Finding the Path 🚩

### Profile Component

## Profile Class Component

Footer

Let say we pass props to our functional comp

```
src > components > JS About.js > [e] About
You, 1 minute ago | 1 author (You)
1 import { Outlet } from "react-router-dom";
2 import ProfileFunctionalComponet from "./Profile";
3 import Profile from "./ProfileClass";
4
5 const About = () => {
6   return [
7     <div>
8       <h1>About Us Page</h1>
9       <p>
10      This is the Namaste React Live Course Chapter
11
12      <ProfileFunctionalComonet name={"Akshay"} />
13    </div>
14  ];
15};
16
17 export default About;
```

```
src > components > JS Profile.js > [e] Profile
1 const Profile = (props) => {
2   return (
3     <div>
4       <h2>Porofile Component</h2>
5       <h3>Name: {props.name}</h3>
6     </div>
7   );
8 };
9
10 export default Profile;
11
```

How to pass this prop in Class component?

Here we get access to something known as **this.props**

```
<Profile name={"AkshayClass"} />
```

```
import React from "react";

class Profile extends React.Component {
  render() {
    return (
      <div>
        <h1> Profile Class Component </h1>
        <h2>Name: {this.props.name}</h2>
      </div>
    );
  }
}

export default Profile;
```

**React tracks our class and it takes the prop and attach it to the "this" of our class. This process happens during reconciliation.**

## Let us create a state inside Functional comp

We make use of useState hook and create state variable

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

const Profile = (props) => {
  const [count] = useState(0);
  return (
    <div>
      <h2>Porofile Component</h2>
      <h3>Name: {props.name}</h3>
      <h3>Count: {count}</h3>
    </div>
  );
}

export default Profile;
```

Same thing in class component is like:

Class comp also has its states, class has a constructor  
Constructor takes in props

```
constructor(props){
  super(props);
}
```

### Why we do super(props) ??

**super(props) allows accessing this. props in a Constructor function.** In fact, what the super() function does is, **calls the constructor of the parent class**

### Why we are making constructor?

**Constructor is a place which is used for initialisation. When our class component is loaded, our constructor will be first one to be called so this is the best place to create state in class based component using this.state**

```
constructor(props) {
  super(props);
  // Create State
  this.state = {
    count: 0,
  };
}
```

How to use this state inside our code?

```
<h2>Count: {this.state.count}</h2>
...
import React from "react";

class Profile extends React.Component {
  constructor(props) {
    super(props);
    // Create State
    this.state = {
      count: 0,
    };
  }

  render() {
    return (
      <div>
        <h1> Profile Class Component </h1>
        <h2>Name: {this.props.name}</h2>
        <h2>XYZ: {this.props.xyz} (property)
        <h2>Count: {this.state.count}</h2>
      </div>
    );
  }
}

export default Profile;
```

We can also **destructure** it also:

```
const { count } = this.state;
```

In functional comp we can do something like this

```
const [count] = useState(0);
const [count2] = useState(0);
```

But in class comp, it is **not correct**

```
constructor(props) {
  super(props);
  // Create State
  this.state = {
    count: 0,
  };
  this.state = {
    count: 0,
  };
}
```

In class comp, all state variables are created as part of same object  
React uses **whole state is a one big object**

```
constructor(props) {
  super(props);
  // Create State
  this.state = {
    count: 0,
    count2: 0,
  };
}
```

Now we can do **setCount** like this in Functional comp

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

const Profile = (props) => {
  const [count, setCount] = useState(0)
  return (
    <div>
      <h2>Porofile Component</h2>
      <h3>Name: {props.name}</h3>
      <h3>Count: {count}</h3>
      <button onClick= {
        ()=> setCount(1)
      }>Count</button>
    </div>
  );
};

export default Profile;
```

Now, **how to do this in class component??**

We do not mutate state directly like `this.setState.count = 1`

Because react tracks about state and do reconciliation accordingly

```
return (
  <div>
    <h1> Profile Class Component </h1>
    <h2>Name: {this.props.name}</h2>
    <h2>XYZ: {this.props.xyz}</h2>
    <h2>Count: {count}</h2>
    <button
      onClick={() => {
        this.setState({
          count: 1,
        });
      }}
    >
      SetCount
    </button>
  </div>
);
```

Reconciliation works the same way for both functional and class based components.

How to set 2 states at a time in functional comp?

```
<button
  onClick={() => [
    setCount(1),
    setCount2(2),
  ]}
>
  Count

```

In class we can do this like:

```
// NEVER DO THIS!
this.setState([
  count: 1,
  count2: 2,
]);
```

We can update all states together as one whole object in class based component

What is sequence of Execution in Functional comp?

React LifeCycle

**First a constructor is called**  
**Then rendering will happen**

**Where to do API call in functional component?**

Inside useEffect() and the sequence is initial render -> useEffect is called after render

**In class based component?**

We have to follow the same sequence of execution, There is something known as  
**componentDidMount()** which will be called after our render

**When a class component renders, It has some Lifecycle methods like render, componentDidMount(), constructor() are some of the Lifecycle methods.**

**First Constructor is called**

**Then component is rendered**

**Then componentDidMount is called**

So **Best place to make an API call in class based component is componentDidMount()**

There are **lot more lifecycle methods, we will see all important ones first**

Let us **convert our About component to class based component**

Instead of writing **React.Component** we can also write **Component** and **import it as named**  
**import from "react"**

```
import { Component } from "react";      You, 11 seconds ago • Unco

You, now | 1 author (You)
class About extends Component {
  render() {
    return (
      <div>
        <h1>About Us Page</h1>
        <p>
          This is the Namaste React Live Course Chapter 07 - Findin
        </p>
        <ProfileFunctionalComponet name={"Akshay"} />
        <Profile name={"AkshayClass"} xyz="abc" />
      </div>
    );
  }
}

export default About;
```

Let us write our LifeCycle methods now, **constructor**, **componentDidMount**  
And **Sequence of call will be constructor -> render -> compDidMount**

And to make an API call we use `ComponentDidMount`

We use constructor to make states as constructor is first one to be called in class component

Now we know `About` is also a class based component and we are using `Profile` inside `About` and it is also a class component

```
import { Outlet } from "react-router-dom";
import ProfileFunctionalComponet from "./Profile";
import Profile from "./ProfileClass";
import { Component } from "react";

You, 4 minutes ago | 1 author (You)
class About extends Component {
  constructor(props) {
    super(props);
    console.log("Parent - constructor");      You, 4 minu
  }
  componentDidMount() {
    // Best place to make an Api call
    console.log("Parent - componentDidMount");
  }
  render() {
    console.log("Parent - render");
    return (
      <div>
        <h1>About Us Page</h1>
        <p>
          This is the Namaste React Live Course Chapter 0
        </p>
        <Profile name={"AkshayClass"} xyz="abc" />
      </div>
    );
  }
}

export default About;
```

What is sequence of execution now whose lifecycle methods will be called first??

1. First constructor of `About` will be called
2. Then `render` of `About` will be called
3. Now inside `render` of `About`, we have `profile` so `Constructor` of `profile` will be called
4. Now `render` of `profile` is called
5. Now `componentDidMount` of `profile` will be called
6. Now `componentDidMount` of `About` will be called

So first it will complete Lifecycle of child the complete Lifecycle of itself

```
Parent - constructor  
Parent - render  
Child - Constructor  
Child - render  
Child - componentDidMount  
Parent - componentDidMount
```

> |

### What if we have 2 children?

Now we call this analogy as "Parent", "First Child", "Second Child"

1. Parent constructor will be called
2. Parent render will be called
3. First child constructor will be called
4. First child render will be called
5. Now, Second child constructor will be called
6. Second child render will be called
7. First child compDidMount will be called
8. Second child compDidMount will be called
9. Parent compDidMount will be called

```
Parent - constructor  
Parent - render  
Child - Constructor First Child  
Child - render First Child  
Child - Constructor Second Child  
Child - render Second Child  
Child - componentDidMount First Child  
Child - componentDidMount Second Child  
Parent - componentDidMount
```

|

### When react is rendering things up

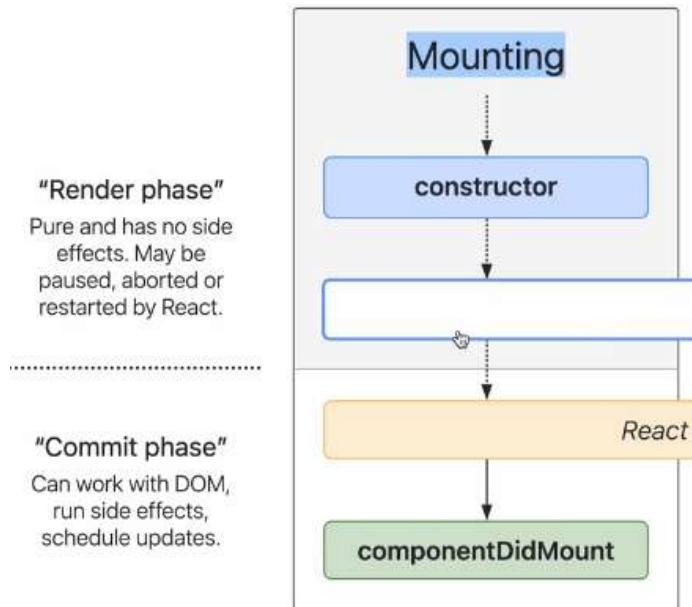
There are 2 phases:

1. Render Phase

First of all Render phase is finished, This phase includes our constructor and render method. Now we are ready to put things in the DOM.

2. Commit Phase

It is the place where React is actually updating the DOM and compDidMount is called after component has rendered.



### Render phase is faster than Commit phase

When there are 2 children, React tries to batch up the render phase of childrens

So It will first complete render phase of both children then Commit phase of both children will be called.

```

/**
 *
 * Parent Constructor
 * Parent render
 *   First Child constructor
 *   First Child render
 *   Second Child constructor
 *   Second Child render
 *
 *   DOM UPDATED for children
 *
 *   first Child componentDidMount
 *   Second Child componentDidMount
 * Parent componentDidMount
 *
 */

```

Now let us make an API call in componentDidMount inside About.js to fetch some user information for which we call github API  
 We need to make our componentDidMount async as we are using fetch()

```
async componentDidMount() {
  // API Calls
  const data = await fetch("https://api.github.com/users/akshaymarch7");
  const json = await data.json();
  this.setState({
    userInfo: json,
  });
  console.log("Child - componentDidMount" + this.props.name);
}
```

But in functional comp, we cannot make this callback function as async

```
useEffect(async () => {
  getRestaurantInfo();
}, []);
```

useEffect() is an asynchronous, non-blocking function, async callbacks cannot be made directly inside of it. To make sure the asynchronous code executes as intended, we can use a different way that declares a new function inside the "useEffect()" hook.

The **special pattern is to create a new function** that is declared inside the useEffect() hook and that contains the async function, and then we can call this new function inside the useEffect() hook. Here's an example:

```
import React, { useEffect } from 'react';

function App() {
  useEffect(() => {
    async function fetchData() {
      const response = await fetch('https://example.com/data');
      const data = await response.json();
      console.log(data);
    }

    fetchData();
  }, []);
}

return <div>Hello World</div>;
}
```

In this example, a brand-new function called "fetchData()" has been developed and is declared inside the "useEffect()" hook. The asynchronous code that we want to execute is contained in this function. Then, to make sure that it only executes once when the component mounts, we call this function from within the "useEffect()" hook and pass an empty dependency array as the second argument.

By using this pattern, we can ensure that the asynchronous function runs as expected without any issues.

Now Let us make an state to store API data with intial value "Dummy Name", "Dummy Location"

```
constructor(props) {
  super(props);
  // Create State
  this.state = {
    userInfo: {
      name: "Dummy Name",
      location: "Dummy Location",
    },
  };
  console.log("Child - Constructor" + this.props.name);
}
```

```
async componentDidMount() {
  // API Calls
  const data = await fetch("https://api.github.com/users/akshaymarch7");
  const json = await data.json();
  this.setState({
    userInfo: json,
  });
  console.log("Child - componentDidMount" + this.props.name);
}
```

Inside profile.js

```
render() {
  const { count } = this.state;
  console.log("Child - render" + this.props.name);
  return (
    <div>
      <h1> Profile Class Component </h1>
      <img src={this.state.userInfo.avatar_url} />
      <h2>Name: {this.state.userInfo.name}</h2>
      <h2>Location: {this.state.userInfo.location}</h2>
    </div>
  );
}
```

**Sequence of Execution comes as**

```

Parent - constructor
Parent - render
Child - Constructor First Child
Child - render First Child
Parent - componentDidMount ↴

▶ {login: 'akshaymarch7', id: 12824231, n
  tps://avatars.githubusercontent.com/u/1
Child - componentDidMount First Child
Child - render First Child
>

```

As our **compDidMount** of child has **async** operation going on so **compDidMount** of parent is loaded first then the child and now we are doing **setState** so now again child will be rendered and update the DOM and it will call another method called **componentDidUpdate**

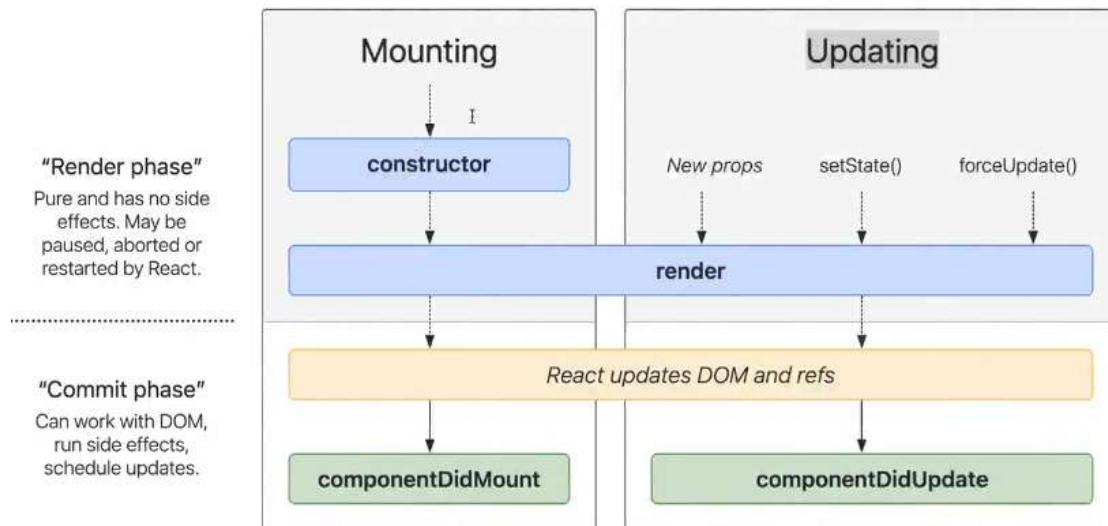
This Re-render cycle is known as **Updating**

```

componentDidupdate() {
  console.log("Component Did Update")
}

```

Now this is because functional component is so convenient.



Now after **setState**, update cycle will start and **render** will be called and now DOM is updated in our UI and **componentDidUpdate** will be called

```

Console was cleared
Child - Constructor First Child
Child - render First Child

▶ {login: 'akshaymarch7', id: 12824231, no
  tps://avatars.githubusercontent.com/u/12

Child - componentDidMount First Child
Child - render First Child
Component Did Update

```

There is something known as **Unmounting** also **(componentWillUnmount)**

**It is called when component is about to unmount, it will be called  
When it will unmount from the DOM, when we go to some other page**

```

componentWillUnmount() {
  console.log("ComponentWillUnmount");
}

```

As we go to Contact page, it profile will unmount

The screenshot shows a web application interface with a header containing a logo, 'Food Villa', and navigation links for 'Home', 'About', 'Contact', 'Cart', and 'Login'. Below the header, the text 'Contact Us Page' is displayed. On the right side of the screen, the browser's developer tools are open, specifically the 'Console' tab. The console output shows the following logs:

```

Child - Constructor First Child
Child - render First Child
▶ {login: 'akshaymarch7', id: 12824231, no
  tps://avatars.githubusercontent.com/u/12

Child - componentDidMount First Child
Child - render First Child
Component Did Update
✖ Uncaught (in promise) Error: A li
  returning true, but the message c
ComponentWillUnmount

```

## Never compare React Lifecycle Methods to React Hooks

For the first Render, component is mounted after that it is always updated not mounted.

**componentDidUpdate** is called after every render

To do an operation if value of our state changes We use to write code like this

```

componentDidUpdate(prevProps, prevState) {
  if (
    this.state.count !== prevState.count ||
    this.state.count !== prevState.count
  ) {
    console.log("Component Did Update");
  }
}

```

The above code will do the same job as

```
useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count, count2]);
```

Suppose We want to do something if count1 changes and something else if count2 changes

We will use 2 useEffects

```
useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count]);

useEffect(() => {
  // API Call
  //console.log("useEffect");
}, [count2]);
```

Same will be done like this in Class component

```
componentDidUpdate(prevProps, prevState) {
  if (
    this.state.count !== prevState.count || 
    this.state.count2 !== prevState.count2
  ) {
    // code
  }
  if (
    this.state.count2 !== prevState.count2
  ) {
    // code
  }
  console.log("Component Did Update");
}

componentWillUnmount() {
  console.log("ComponentWillUnmount");
}
```

What is **use of componentWillMount** ??

We know componentWillMount will be called when we change the page and our component goes out of screen.

We see when we go to home page from about page or vice-versa

Console.log("componentWillUnmount") will be called

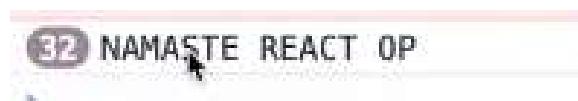
## ComponentWillUnmount

Let say if we put setTimeout in componentDidMount

```
componentDidMount() {
  setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);

  console.log("Child - componentDidMount");
}
```

Now if we change the page, timeOut is still calling after 1000ms



So, if we are changing the pages, React is just changing the components, reloading is not happening. So everytime we change pages, setTimeout replicas will be made which will make our app performance very very bad.

So To cleanup, **it is really important to do componentWillMount**

Otherwise, everytime we change pages, new setTimeout will be made and called

So In our **componentWillUnmount** , We will call our clearInterval.

Now, **when we change the pages, componentWillMount will call clearInterval and timeout will be killed**

```
componentDidMount() {
  this.timer = setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);

  console.log("Child - componentDidMount");
}

componentDidUpdate(prevProps, prevState) {...}

componentWillUnmount() {
  clearInterval(this.timer);
  console.log("ComponentWillUnmount");
}

render() {...}
```

**What will happen if we create setInterval inside useEffect??**

```
useEffect(() => {
  // API Call
  setInterval(() => {
    | console.log("NAMASTE REACT OP ");
    |}, 1000);
}, []);
```

Now, does not matter we move from About page to any page, timeOut will keep on running and it will not stop.



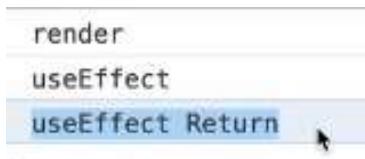
**It will not stop because we are not cleaning things up, how to destroy this in our functional Component??**

There is something known as **return callback function inside useEffect** and this callback function will be called on **Unmounting in functional component**.

```
useEffect(() => {
  // API Call
  // setInterval(() => {
  //   console.log("NAMASTE REACT OP ");
  // }, 1000);
  console.log("useEffect");

  return () => {
    | console.log("useEffect Return");
  };
}, []);
```

"useEffect return" will be called after we change the page in routing, same as we did in **componentWillUnmount**



So, we will call our clearInterval inside our callback function of return statement

```
useEffect(() => {
  // API Call
  const timer = setInterval(() => {
    console.log("NAMASTE REACT OP ");
  }, 1000);
  console.log("useEffect");

  return () => [
    clearInterval(timer),
    console.log("useEffect Return"),
  ];
}, [1]);
```

## Session 11

We will be doing code splitting, lazy loading, suspense, chunking etc etc to make our app optimised.

Most important hooks in react are useState and useEffect.

We have also used useParams till now

In this Lecture, we will see **how to build custom hooks.**

**What is hook at the end of the day?**

Its an normal JS function, and we will learn to make our own hook in this lecture.

**How, when, why to build our own hook?**

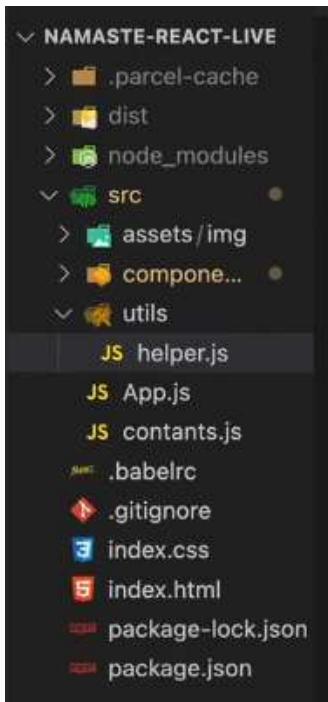
Why => For **reusability, readability, maintainability, separation of concerns, modularity (breaking code into small pieces, chunks), testable.**

We can wrap up a small logic inside a function and can use it anywhere.

**Let say we have made function filterData in body.js and we need to use this function in other files also so**

Whenever we are making a re-usable function, we should make it in a separate file so that it can be shared by others also. These are called utility function

**We create a folder "utils" inside "src", now we can create helper.js or util.js**



Now we just copy paste our function in helper.js and export it

```
utils > JS helper.js > ⓘ filterData
      export function filterData(searchText, restaurants) {
        const filterData = restaurants.filter((restaurant) =>
          restaurant?.data?.name?.toLowerCase()?.includes(searchText.toLowerCase())
        );
        return filterData;
    }
```

Now we import it in our body.js

```
import { filterData } from "../utils/helper";
```

Now **any component can use our functions. This makes our code re-usable and cleaner**

**How functions/hooks make our code maintainable?**

It is **easy to debug**.

Let us now **create our own hook**.

In our RestaurantMenu component, we have an API call which gives us Menu data for each resto with some ID that we extract using useParams()

```

const RestaurantMenu = () => {
  // how to read a dynamic URL params
  const { resId } = useParams();
  // Use proper names
  const [restaurant, setRestaurant] = useState(null);

  useEffect(() => {
    getRestaurantInfo();
  }, []);

  async function getRestaurantInfo() {
    const data = await fetch(
      "https://www.swiggy.com/dapi/menu/v4/full?lat=12.9351929&
      resId
    );
    const json = await data.json();
    console.log(json.data);
    setRestaurant(json.data);
  }

  return !restaurant ? (
    <Shimmer />
  ) : (
    <div className="menu">
      <div>

```

Let us **make a custom Hook which gets the API data of menu for us.**

So we create a new file

**Always create a hook with "use" name in front of it. That tells react that's it's a hook.**

**We can create that hook file anywhere**

**We create it inside utils**

Named "**useRestaurant.js**"

We can give named export also, to a default export but It's a good practice to pass it as default export if there is only one things exporting from one component.

useRestaurant will fetch the data for us and send us.

What getRestaurant function inside RestaurantMenu doing is

```

async function getRestaurantInfo() {
  const data = await fetch(
    "https://www.swiggy.com/dapi/menu/v4/full?lat=12.9351929&
    resId
  );
  const json = await data.json();
  console.log(json.data);
  setRestaurant(json.data);
}

```

It uses resId from useParams which comes from URL and make an API call and sets the data in the state.

Now our custom hook also want to do something like that.

We just want to pass resId to our custom hook and it should provide us with the menu API data.

```
const restraunt = useRestaurant(resId)
```

So our useRestaurant takes resId as a parameter, we want this hook to return API data

We also want API data to change according to resID

We create a state with initial value NULL

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

const useRestaurant = (resId) => {
  const [restaurant, setRestauraunt] = useState(null);

  // Get data from API
  useEffect(() => {
    getRestaurantInfo();
  }, []);

  async function getRestaurantInfo() {
    const data = await fetch(
      `https://www.swiggy.com/dapi/menu/v4/full?lat=12.9351929&lng=77.62448
      ${resId}
    );
    const json = await data.json();
    console.log(json.data);
    setRestauraunt(json.data);
  }

  // return restraint Data
  return restaurant;
};

export default useRestaurant;
```

Now our RestaurantMenu component becomes very clean using our custom hook

```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import { IMG_CDN_URL } from "../contants";
import useRestaurant from "../utils/useRestaurant";
import Shimmer from "./Shimmer";

const RestaurantMenu = () => [
  // how to read a dynamic URL params
  const { resId } = useParams();

  const restaurant = useRestaurant(resId);

  return !restaurant ? (
    <Shimmer />
  ) : (
    <div className="menu">
      <div>
        <h1>Restraunt id: {resId}</h1>
        <h2>{restaurant?.name}</h2>
        <img src={IMG_CDN_URL + restaurant?.cloudinaryImageId} />
        <h3>{restaurant?.area}</h3>
        <h3>{restaurant?.city}</h3>
        <h3>{restaurant?.avgRating} stars</h3>
        <h3>{restaurant?.costForTwoMsg}</h3>
      </div>
      <div>
        <h1>Menu</h1>
        <ul>
          {Object.values(restaurant?.menu?.items).map((item) =>
            <li key={item.id}>{item.name}</li>
          )}
        </ul>
      </div>
    </div>
  );
]
```

Functional component returns us a JSX, our custom hook is not an functional component as it not need to return any JSX, it can return any piece of data.

We can make useRestaurant more clean by keeping API URL into constants.js

```
export FETCH_MENU_URL = "https://www.swiggy.com/dapi/menu/v4/full?lat=12.9351929&lng=77.62448069999"
```

```

import { useState } from "react";
import { FETCH_MENU_URL } from "../contants";

const useRestaurant = (resId) => {
  const [restaurant, setRestauraunt] = useState(null);

  // Get data from API
  useEffect(() => {
    getRestaurantInfo();
  }, []);

  async function getRestaurantInfo() {
    const data = await fetch(FETCH_MENU_URL + resId);
    const json = await data.json();
    console.log(json.data);
    setRestauraunt(json.data);
  }

  // return restraint Data
  return restaurant;
};

export default useRestaurant;

```

**Let us make one more feature using custom hook from scratch**

**Let us say our wifi is off, chrome say "You are offline". Let us make this functionality which checks user is online or offline and based on it if user is offline, we do not let user click food cards and say "Check you internet connection"**

**How to know whether user is offline or online?**

We will use Window: online event from JS

## Window: online event

The `online` event of the `Window` interface is fired when the browser has gained access to the network and the value of `Navigator.onLine` switches to `true`.

**Note:** This event shouldn't be used to determine the availability of a particular website. Network problems or firewalls might still prevent the website from being reached.

## Syntax

Use the event name in methods like `addEventListener()`, or set an event handler property.

```

addEventListener('online', (event) => { });
ononline = (event) => { };

```

Let us make an hook `useOnline` which tells whether user is online or offline.

```

const offline = useOnline();      You, 2 minutes ago • Uncommitted chan

if (offline) {
  return <h1>🔴 Offline, please check your internet connection!!</h1>;
}

```



Now we write code inside useOnline which return us True or False based on whether user is online or offline.

We will use state variable to maintain the state of user and make it true by default.

We want this code to run only once, in the initial render and check whether user is online or offline. So we use useEffect

```

import { useState } from "react";
const useOnline = () => {
  const [isOnline, setIsOnline] = useState(true);

  useEffect(() => {
    window.addEventListener("online", () => {
      setIsOnline(true);
    });
    window.addEventListener("offline", () => {
      setIsOnline(false);
    });
  }, []);

  return isOnline; // returns true or false
};

export default useOnline;

```

Now we can also make a library of it and use it in our code.

```

import { useState } from "react";
import useOnline from "../utils/useOnline";

const isOnline = useOnline();

if (!isOnline) { 
  return <h1>🔴 Offline, please check your internet connection!!</h1>;
}

```

We can **fake it going offline** using chrome in **Network Tab**



And we are offline so **we see this**



Let say we want to optimise below code of our useOnline hook

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

const useOnline = () => {
  const [isOnline, setIsOnline] = useState(true);

  useEffect(() => {
    window.addEventListener("online", () => {
      setIsOnline(true);
    });
    window.addEventListener("offline", () => {
      setIsOnline(false);
    });
  }, []);

  return isOnline;
};

export default useOnline;
```

What we will do is, clean up the event listeners because everytime we go online to offline or vice-versa. So always clean your event listeners after they are used.

So we use callback function of return of useEffect and remove event listeners on unMounting.

We extract the logic of event Listeners callback function inside a handleOffline and handleOnline variable so that during removeEventListener we need to pass same function we used during addEventListener

```

import { useState, useEffect } from "react";

const useOnline = () => {
  const [isOnline, setIsOnline] = useState(true);

  useEffect(() => {
    const handleOnline = () => {
      setIsOnline(true);
    };
    const handleOffline = () => {
      setIsOnline(false);
    };

    window.addEventListener("online", handleOnline);
    window.addEventListener("offline", handleOffline);

    return () => {
      window.removeEventListener("online", handleOnline);
      window.removeEventListener("offline", handleOffline);
    };
  }, []);

  return isOnline;
};

export default useOnline;

```

Now suppose we want to show Offline/Online in our Header, we can just use useOnline in our Header component.

```

const Header = () => {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const isOnline = useOnline();

```

```

</div>
<h1>{isOnline ? "✓" : "✗"} /h1>
{isLoggedIn ? (

```

So, now we know how to make a feature and build our custom hook.

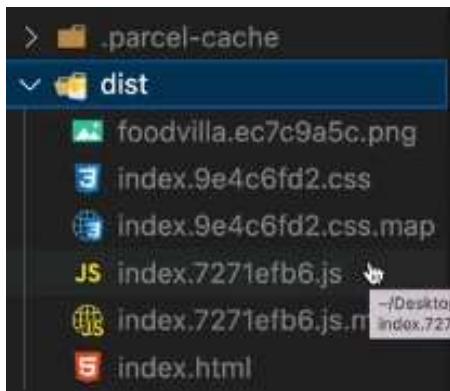
We can also build a hook for isLoggedIn or not, to check whether user is logged in or not.

The component in which we are using our custom hook need not to know the logic behind our hook. It just uses it.

We can also build a hook which gives us Access to our local storage and helps in setting and getting local storage variables.

In the second Lecture we studied about Parcel which is a bundler that optimises our app, minimization, bundle our app and make a optimised build.

Parcel produces only one JS file for all our component all the code



In the production build, size of JS file reduces further.

Let us see MakeMyTrip Website which helps us in booking hotel, cabs, train, flights etc etc.

**How many components would MMT had written in their whole app??**

Search box, Input box, hotel cards, activities etc etc.

There would be 100's of components.

Bundler takes all 100's of components and put them inside one JS file which makes app very slow.

Large production ready apps cannot be made just using one bundle. So we need to do

**Chunking / Code Splitting / Dynamic Bundling / Lazy Loading**

Earlier we were saying we should bundle our app using a bundler to optimise and make a optimised build

Now, We are saying we should do chunking of the bundles

So, **Is bundling good or bad??**

It is good to a extent, we should make logical bundles.

Just few lines of code can optimise our app to a next level.

We want to keep our bundle size small and logical

Every user on MMT website has an intent of coming, some are coming to book flights, some

are coming to book trains, Someone came to book flight will less likely to go to train section or bus section or cab section.



We build different chunks in large applications, system design is done, code splitting is done and make sure that our bundler do not include faltu code, we need to make a logical and small size bundle, we need to use bundlers.

### Feature to make

Whenever our homepage loads it should only load flights section by default

When we click on Train, then it should load train section and URL also changes. It should render the train section bundle.

When we click to Buses Section, it should load Bus bundle

This is called **On Demand Loading / Dynamic Import**

MMT is a image heavy site as there are images associated with everything so we will use cloud to store images.

In Paytm, it has different wallet such as Recharge etc etc. So every Wallet should have different bundle.

### In our FoodVilla website

We can have something like Instamart which can be a different bundle.

Let us create **Instamart component**



Suppose it's a big component which has so many components inside it.

```
const Instamart = () => {
  return (
    <div>
      <h1>Instamart</h1>
      <h1>100s of components inside it</h1>
    </div>
  );
};

export default Instamart;
```

Now Let us include it in our path in AppRouter of app.js

```
{
  path: "/instamart",
  element: <Instamart />,
},
```

Let us create a route for Instamart in our Header.js

```
<Link to="/instamart">
  <li>Instamart</li>
</Link>
```

Home About Contact Cart Instamart

Let us now try to do **Chunking**

Wherever we have import for Instamart (app.js), there we do Lazy loading for it and import it in a Lazy way.

Lazy comes from react as named import just like useState, useEffect

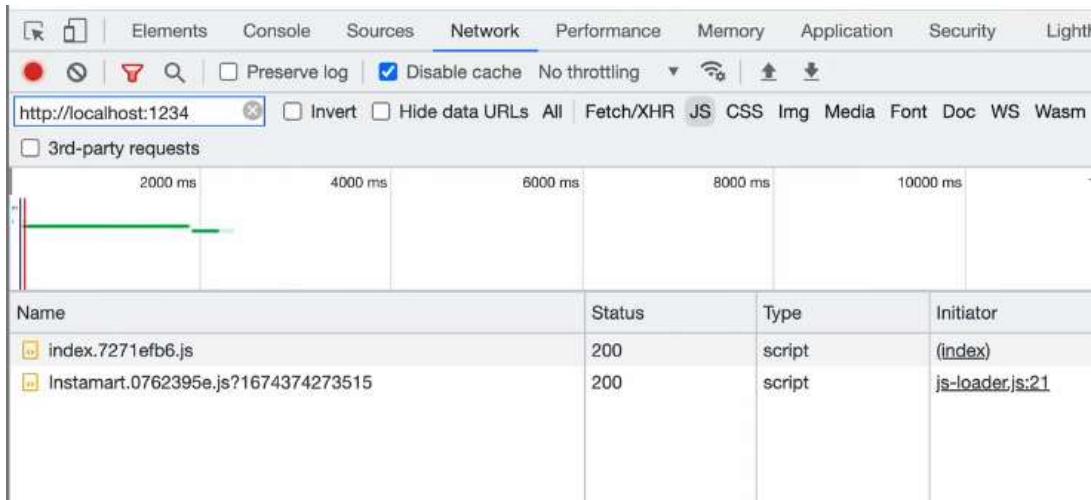
```
import React, { lazy } from "react";

const Instamart = lazy();
```

**Lazy takes a function and now we do a dynamic import**

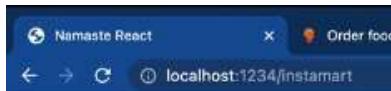
```
const Instamart = lazy(() => import("./components/Instamart"));
```

Now when we go to our normal website, Its JS file does not include Instamart code, when we go to Instamart from our header, we see a different JS file which has all the code for Instamart



So one line of code can do a lot.

But now when we load our Instamart, error pops up



**Oops!!**

**Something went wrong!!**

**undefined : undefined**

It takes some time to load the Instamart script and meanwhile react tries to render it. React tries to render something which is not present. React does not know that Instamart loads on demand, so Instamart JS file is still not there in Browser and react tries to load it so the above error comes.

Once our Instamart JS file is loaded in the Browser now, any number of time we go to Instamart. It does not give any error. But for the first time it shows error.

So this process happening for initial render is called **React will Suspend the loading**

To handle this we can use something known as **Suspense**

```
import React, { lazy, Suspense } from "react";
```

Now we load our Instamart inside Suspense

```
{  
  path: "/instamart",  
  element: () =>  
    <Suspense>  
      <Instamart />  
    </Suspense>  
,    You, 1 second  
},
```

Now React cares for it Automatically and The above error does not come.  
React waits for it to load and then shows it.  
Function inside lazy( ) is a promise so now react waits for the promise to fulfill and then loads it.

Now **Say Our Instamart is a heavy file and it takes 12s to load, Now react waits for it so What should we show for that 12s to the user?? Shimmer**

**Suspense takes a Prop named "fallback" in which we can give Shimmer**

```
{  
  path: "/instamart",  
  element: () =>  
    <Suspense fallback={<Shimmer />}>  
      <Instamart />  
    </Suspense>  
,  
},
```

**Chunking is of no use for small apps**  
**When we see our App is becoming huge like Paytm, MMT etc then Chunking is necessary**

We can make our About component as lazy also

```
const About = lazy(() => import("../components/About"));
```

```
{  
  path: "/about", // parentPath/{path} => localhost:1244/about  
  element: (  
    <Suspense fallback=<h1>Loading....</h1>>  
    | <About />  
    </Suspense>  
)  
,  
  children: [  
    {  
      path: "profile", // parentPath/{path} => localhost:1244/about/profile  
      element: <Profile />,  
    },  
  ],  
},
```

## Never Ever Dynamically Load your component inside another Component.

If you are doing something like this, then do not make Instamart as lazy.

```
const AppLayout = () => {  
  
  const Instamart = lazy(() => import("./components/Instamart"));  
  
  return (  
    <>  
    <Header />  
    <Outlet />  
    <Footer />  
  </>  
);  
};  
const appRouter = createBrowserRouter([  
  {  
    path: "/",  
    element: <AppLayout />,  
    errorElement: <Error />,  
    children: [  
      {  
        path: "about",  
        element: <Instamart />,  
      },  
    ],  
  },  
]);
```

# Session 12

We will learn Tailwind CSS in this lecture

In this lecture, we will see how to write CSS in your large web application.

Tailwind CSS is a CSS framework

Why we need CSS Frameworks?

The primary reason is to be able to write Optimised CSS and it saves time.

We will see How to style our React Components.

One way of putting CSS in our app is to make index.css and put CSS from outside.

We can also write **SCSS and SASS**

**SASS** (Syntactically Awesome Style Sheets) is a **pre-processor scripting language** that will be **compiled or interpreted into CSS**. SassScript is itself a scripting language whereas **SCSS (Sassy Cascading Style Sheets)** is the **main syntax for the SASS which builds on top of the existing CSS syntax**. It makes use of semicolons and brackets like CSS (Cascaded Style Sheets).

SASS and SCSS can import each other. **Sass actually makes CSS more powerful** with math and variable support.

Let's list down the main difference between SASS and SCSS.

- **SASS** is used when we need an **original syntax**, code syntax is not required for SCSS.
- SASS follows strict indentation, SCSS has no strict indentation.
- SASS has a loose syntax with white space and no semicolons, the SCSS resembles more to CSS style and use of semicolons and braces are mandatory.
- SASS file extension is **.sass** and SCSS file extension is **.scss**.
- SASS has more developer community and support than SCSS.
- SASS supports SassDoc to add documentation whereas SCSS allows inline documentation.
- SASS can't be used as CSS and vice-versa whereas a valid CSS code is also a valid SCSS code.
- SASS is hard to add to existing CSS projects whereas SCSS can be added easily to an existing CSS project just by adding new code.

**SCSS Example:**

```
SCSS

/* .scss file */
$bgcolor: blue;
$textcolor: red;
$fontsize: 25px;

/* Use the variables */
body {
    background-color: $bgcolor;
    color: $textcolor;
    font-size: $fontsize;
}
```

**Output CSS:**

```
body {  
    background-color: blue;  
    color: red;  
    font-size: 25px;  
}  
/* now this can apply resulting html file */
```

### SASS Example

#### SASS Example:

SASS

/\* SASS \*/

\$primary-color: green  
\$primary-bg: red

body  
color: \$primary-color  
background: \$primary-bg

### Output CSS

#### Output CSS:

```
/* CSS */  
body {  
    color: green;  
    background: red;  
}
```

Another way of putting CSS is **Inline styling** in the element itself.  
Let say we put inline styling in our input field

```
const searchBtnCSS = [ ]  
|   backgroundColor: "red"  
|]  
|
```

```
<button  
  style={searchBtnCSS}      You, 1 second ago • Uncommitted  
  onClick={() => {  
    //need to filter the data  
    const data = filterData(searchText, allRestaurants);  
    // update the state - restaurants  
    setFilteredRestaurants(data);  
  }}  
>  
  Search  
</button>
```



We can also write in a different Fashion, passing an JS object inside {} to style attribute

```
<button  
  style={{  
    backgroundColor: "red",  
  }}      You, 1 second ago • Uncommitted changes  
  onClick={() => {  
    //need to filter the data  
    const data = filterData(searchText, allRestaurants);  
    // update the state - restaurants  
    setFilteredRestaurants(data);  
  }}  
>  
  Search  
</button>
```

**Inline way of CSS is bad because its Hard-coded, Not Re-usable, Difficult to maintain, Repeats a lot, The job of processing inline CSS is heavy for our Browsers.**  
**External CSS is better than Inline CSS**

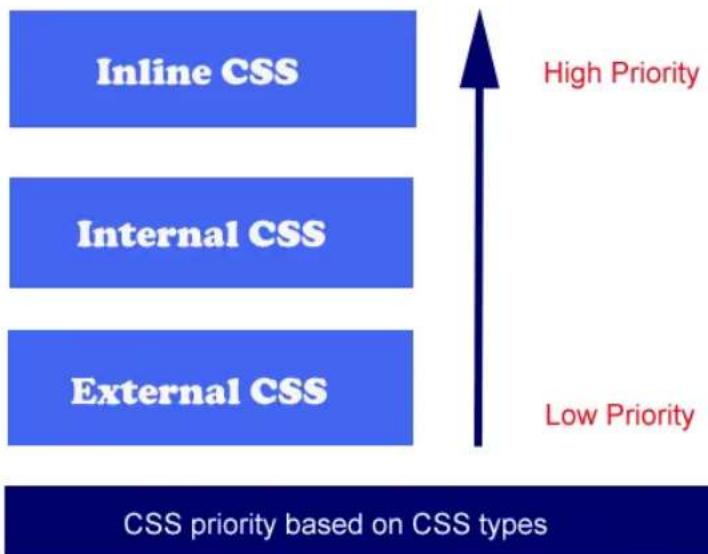
## Specificity of CSS Selectors

1. Id selector (#IdNam)
2. Classes, attributes and pseudo-classes
3. Elements and pseudo-elements
4. Universal (\*) selector

Each of the above mentioned selectors has their own weight to get priorities to apply on element.

Id selector has highest priority because of unique nature of the ID attribute definition.

We have two classes with one ID selector here it will apply **font-size:12px** and **font-weight:500** due to the specificity rule as ID selector has highest priority after inline CSS.



### Tips to remember when dealing with specificity issues

1. The universal selector \* has low or no specificity.
2. If we are writing css with !important then this property will be get applied even it has low priority in rules
3. Selectors used inside pseudo classes like :not() with highest specificity will be the specificity.
4. Child or sibling selectors (>, ~, +) add no specificity.

### Another way is to use Libraries like Material UI

When we build a website, we always have some common things like button, list, navbar etc etc.

**MUI is a fully-loaded component Library**, it has lot of pre-built components which are already built we can just copy and paste.

Lot of companies rely on libraries for their CSS.

### Should we use These Libraries and Frameworks?

### Pros of using Component Libraries/CSS Frameworks

1. Easy to use
2. Optimised
3. Re-usable
4. Saves a lot of time
5. Gives Automatic Themes
6. Gives Consistent UI (All button will be same shape etc etc) Like below we can see Button of Youtube are almost same shape. If they would have not used any Library, it would have taken lot of time & efforts to modify these button.

## 7. Gives us Responsive Design



### Which is better??

None of them is better, None of them is worst. It depends how a developer need to use it.  
All libraries are almost same just that it depends what we want to use.

### Can we use 2 CSS frameworks inside one projects??

```
npm
npm install @mui/material @emotion/react @emotion/styled
Copy
```

These Libraries are just npm packages at the end of the day

We can use multiple packages in our project so yes, we can use different CSS frameworks but its not a good practice to use multiple frameworks as it destroys the consistency of the app.

There is something known as **Chakra UI** also which is another library.

We can also use **Bootstrap**

## Cons of using CSS Frameworks / Component Libraries

1. Our bundle size increase by a lot.
2. We loose the control over our Design as now, we have to follow a pre-built design.  
Development becomes easy but customization becomes very hard.

## React styled Components

This is mainly used in React.

Using JSX we were able to bring HTML inside JS

We can also bring CSS inside JS using react styled component

Let say to make a button we can write it like:

```
const Button = styled.a`  
  /* This renders the buttons above... Edit me! */  
  display: inline-block;  
  border-radius: 3px;  
  padding: 0.5rem 0;  
  margin: 0.5rem 1rem;  
  width: 11rem;  
  background: transparent;  
  color: white;  
  border: 2px solid white;  
}  
  
/* The GitHub button is a primary button  
 * edit this to target it specifically! */  
${props => props.primary && css`  
  background: white;  
  color: black;  
}`
```

We are passing CSS as props

It is re-usable, we can use button anywhere.

It's a different way of writing code.

## Tailwind CSS

It is a open source CSS Framework.

Inline CSS use to save our time and we can write CSS in the same file, We no more need to go in a different file and toggle between different files to write CSS.

Tailwind CSS also allows us to write CSS on the go.

It is re-usable

It has lot of pre-built classes that we can automatically use.

Less Bundle Size as it offers minimal CSS to us.

It helps us in bulding very flexible UI, It is very much customisable so it gives us lot of control over our design.

### How to configure Tailwind CSS?

Great Documentation, To use Tailwind CSS in our app

We can use **CDN links**, put it in index.html in head

It works with normal CSS and JS also.

```
<link rel="stylesheet" href="index.css" />  
<script src="https://cdn.tailwindcss.com"></script>  

```

**Tailwind changes behaviour of many elements like h1 in Tailwind will be different as compared to behaviour of h1 in normal CSS.**

Below is the h1 for Tailwind

Hello world!

Tailwind says "Write CSS in Tailwind style"

Let say we want our h1 tag to be little different.

In Tailwind every style is inside className

```
<h1 class="text-xl font-bold">Hello world!</h1>
<h1 class="text-4xl font-bold">Hello world 2!</h1>
```

Hello world!  
Hello world 2!

How to know what are the classes I can give to Tailwind?

Go to Documentation, use Search (uses **Elastic search**: Elasticsearch (built in JAVA) **allows you to store, search, and analyze huge volumes of data quickly and in near real-time and give back answers in milliseconds**. It's able to achieve fast search responses because instead of searching the text directly, it searches an index.)

The screenshot shows the Tailwind CSS v3.2 documentation page. The top navigation bar includes the Tailwind CSS logo, version v3.2.4, and a search bar. The main content area has a sidebar on the left with a 'Quick search...' input field and a list of typography-related utility classes: Font Family, Font Size, Font Smoothing, Font Style, Font Weight, Font Variant Numeric, Letter Spacing, Line Height, List Style Type, List Style Position, Text Align, Text Color, Text Decoration, Text Decoration Color, Text Decoration Style, Text Decoration Thickness, Text Underline Offset, Text Transform, Text Overflow, Text Indent, Vertical Align, Whitespace, and Word Break. The 'Text Color' section is currently selected. The main content area displays the 'Text Color' utilities, their corresponding class names, and their properties. Below this, there are sections for 'Basic usage' and 'Setting the text color', followed by a preview box containing the text 'The quick brown fox jumps over the lazy dog.'

	Class	Properties
Font Family	text-green-200	color: rgb(187 247 208);
Font Size	text-green-300	color: rgb(134 239 172);
Font Smoothing	text-green-400	color: rgb(74 222 128);
Font Style	text-green-500	color: rgb(34 197 94);
Font Weight	text-green-600	color: rgb(22 163 74);
Font Variant Numeric	text-green-700	color: rgb(21 128 61);
Letter Spacing	text-green-800	color: rgb(22 101 52);
Line Height	text-green-900	color: rgb(20 83 45);

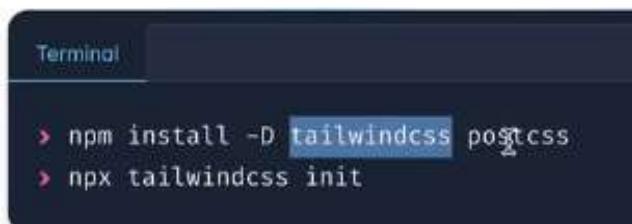
```
<h1 class="text-xl font-bold">Hello world!</h1>
<h1 class="text-2xl font-bold text-red-500">Hello world 2!</h1>
```

Hello world!  
Hello world 2!

## CDN is not the good way to use Tailwind, How to use it?

Always use it as a npm package.

To install Tailwind in Parcel, we use

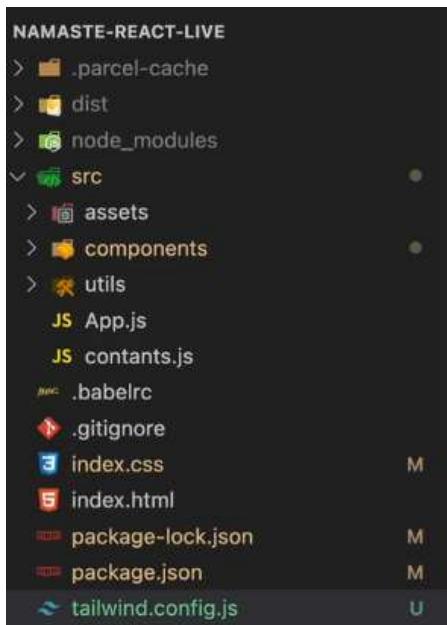


```
Terminal
> npm install -D tailwindcss postcss
> npx tailwindcss init
```

## What is postCSS?

At the end, we need to convert these class based CSS to browser readable CSS so this conversion is done by postCSS using JS functions.

As soon we do tailwindcss init, we get tailwind.config.js in our folder structure



```
tailwind.config.js > ↵ <unknown> > ↵ plugins
1  /** @type {import('tailwindcss').Config} */
2  module.exports = {
3    content: [],
4    theme: {
5      extend: {},
6    },
7    plugins: [],
8  }
9
```

The **content** section of your **tailwind.config.js** file is where you configure the **paths to all of your HTML templates, JavaScript components, and any other source files that contain Tailwind class names.**

**Tailwind CSS works by scanning all of your HTML, JavaScript components, and any other template files for class names, then generating all of the corresponding CSS for those styles.**

In order for Tailwind to generate all of the CSS you need, it needs to know about every single file in your project that contains any Tailwind class names.

Configure the paths to all of your content files in the content section of your configuration file:

```
tailwind.config.js

/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [
    './pages/**/*.{html,js}',
    './components/**/*.{html,js}'
  ],
  // ...
}
```

## Plugins

Plugins let you **register new styles for Tailwind to inject into the user's stylesheet using JavaScript instead of CSS.**

```
tailwind.config.js
```

```
const plugin = require('tailwindcss/plugin')

module.exports = {
  plugins: [
    plugin(function({ addUtilities, addComponents, e, config }) {
      // Add your custom styles here
    }),
  ]
}
```

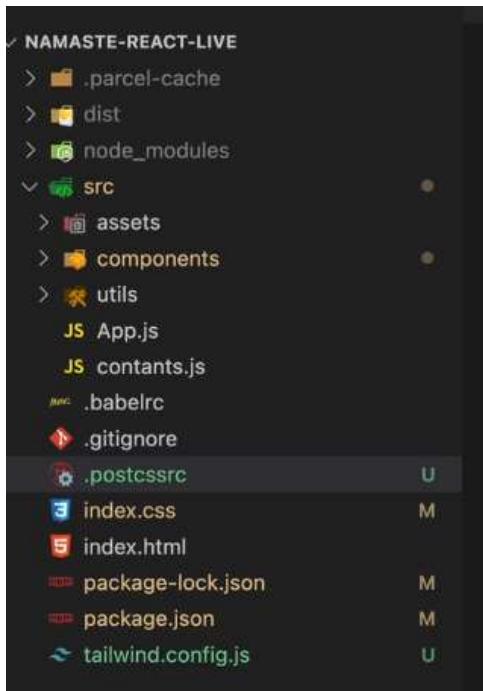
## Theme and extend

To **extend the default theme in Tailwind**, first, go to the “theme” section “tailwind. config. js” file, add the “extend” property, and **add the new desired theme properties, such as screens, colors, fonts, etc. to extend the Tailwind default theme.**

In our project we are using Tailwind in all of our components which is inside src folder, so we write this in content.

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: ['./src/**/*.{html,js}'],
  theme: {
    extend: {},
  },
  plugins: [],
};
```

Now **we configure postcss file using .postcssrc file just like we did .babelrc which tells Parcel that we are using tailwind.**



```
❶ .postcssrc > {} plugins > {} tailwindcss
1   {
2     "plugins": {
3       "tailwindcss": {}
4     }
5   }
6
```

While writing Tailwind we will not write anything inside index.css, just write

```
❷ index.css
You, now | 1 author (You)
1   @tailwind base;
2   @tailwind components;
3   @tailwind utilities;
4
```

Now we will not write any other type of CSS, we only write Tailwind CSS now.

Let us modify our header component

It is not possible to remember all classes so we use this extension in our vscode



JPEG is Joint Photographic Experts Group. The full form of PNG is Portable Network Graphics.

Unlike JPEGs, PNGs support transparent backgrounds, making them preferred for graphic design. JPG is same as JPEG.

Let say in Tailwind we want to give some value which is not present by default say we want to give 200px value to our div, then we can make use of **square bracket notation [ ]** which is also called giving **dynamic classes**

```
<div>
  <div className="w-[200px]"> You, 1 second
    <img src={IMG_CDN_URL + cloudinaryImageId} />
    <h2>{name}</h2>
    <h3>{cuisines.join(", ")}</h3>
    <h4>{lastMileTravelString} minutes</h4>
  </div>
```

Tailwind makes sure to put only those things in styles sheet during bundling which we are using.

Try to use, classes given by Tailwind and avoid dynamic classes using [ ] to make bundle size smaller.

We can style our Restro card like this

```
<div className="w-56 p-2 m-2 shadow-lg bg-pink-50">
  <img src={IMG_CDN_URL + cloudinaryImageId} />
  <h2 className="font-bold text-xl">{name}</h2>
  <h3>{cuisines.join(", ")}</h3>
  <h4>{lastMileTravelString} minutes</h4>
</div>
```

Tailwind saves lot of time for us, we can style a button on the go and makes debugging CSS very easy.

**How to change the button colour on hover?**

Just give className **hover:bg-red-500**

```
className="p-2 m-2 bg-purple-900 hover:bg-gray-500 text-white rounded-md"
```

We can also give CSS to first letter, before, after etc etc in Tailwind

**How to re-use className in Tailwind?** We have to write same className to other elements also but in the bundle, it **counts that className as just one style only.**

**On focussing our input box we want to change some CSS?**

```
<input  
  type="text"  
  className="■ focus:bg-green-200 p-2 m-2"  
  placeholder="Search"  
  value={searchText}  
  onChange={(e) =>  
    setSearchText(e.target.value);  
  }  
>
```

**How to write Media Query in Tailwind where we want our CSS to change according to our screen width?**

Suppose if our website crosses threshold of our small devices, make my header "blue"

```
■ sm:bg-blue-50"
```

For Medium devices

```
■ md:bg-yellow-50
```

```
<div className="flex justify-between ■ bg-pink-50 shadow-lg ■ sm:bg-blue-50 ■ md:bg-yellow-50">  
</div>
```

## Pros of Tailwind

1. Easy to debug
2. No duplicate CSS
3. Bundle size is small
4. Saves time
5. Gives you lot more control in your design
6. It's a newer way of writing CSS, using JSX and Tailwind we can do everything inside our JS file

## Cons of Tailwind

1. Too many classNames
2. There is a high initial learning curve related to it for a beginner.

```

-->
<div class="bg-white">
  <div class="mx-auto max-w-2xl py-16 px-4 sm:py-24 sm:px-6 lg:max-w-7xl lg:px-8">
    <h2 class="sr-only">Products</h2>

    <div class="grid grid-cols-1 gap-y-10 gap-x-6 sm:grid-cols-2 lg:grid-cols-3 xl:grid-cols-4 xl:gap-x-8">
      <a href="#" class="group">
        <div class="aspect-w-1 aspect-h-1 w-full overflow-hidden rounded-lg bg-gray-200 xl:aspect-w-7 xl:aspect-h-8">
          
        </div>
        <h3 class="mt-4 text-sm text-gray-700">Earthen Bottle</h3>
        <p class="mt-1 text-lg font-medium text-gray-900">$48</p>
      </a>

      <a href="#" class="group">
        <div class="aspect-w-1 aspect-h-1 w-full overflow-hidden rounded-lg bg-gray-200 xl:aspect-w-7 xl:aspect-h-8">
          
        </div>
        <h3 class="mt-4 text-sm text-gray-700">Nomad Tumbler</h3>
        <p class="mt-1 text-lg font-medium text-gray-900">$35</p>
      </a>

      <a href="#" class="group">
        <div class="aspect-w-1 aspect-h-1 w-full overflow-hidden rounded-lg bg-gray-200 xl:aspect-w-7 xl:aspect-h-8">
          
        </div>
        <h3 class="mt-4 text-sm text-gray-700">Focus Paper Refill</h3>
        <p class="mt-1 text-lg font-medium text-gray-900">$89</p>
      </a>

      <a href="#" class="group">
        <div class="aspect-w-1 aspect-h-1 w-full overflow-hidden rounded-lg bg-gray-200 xl:aspect-w-7 xl:aspect-h-8">
          
        </div>
      </a>
    </div>
  </div>

```

- Not much Readable, makes our code little ugly.

## Session 13

Handling data for web app is one of the most crucial part of developing a web app  
 When we have a react application, we have a UI layer whatever we see body,div,button are UI layer.

There is one more layer, Data Layer.

Both UI and Data layer combine makes our Frontend of an web app.

Everything we see on UI is UI layer which we built using JSX. JSX is just HTML at the end of the day which is converted by babel which is an Transpiler.

React works on virtual DOM which helps in reconciliation

Diffing algorithm works between previous Virtual DOM and current Virtual DOM and updates it according to new data added in actual DOM.

Whole UI layer is powered by the data layer

In react, Data Layer consist of state and the props.

We manage data using state and props.

### Difference between state and props?

Suppose we have a container, we want a variable to have a scope within container so we use state, props is the value passed between one component to another component.

So, state is the local variable which can be passed as an prop to some another component.

If at the root component i.e App.js we have a local variable (state) say userInformation

```

const AppLayout = () => {
  const [user, setUser] = useState({
    name: "Namaste React",
    email: "support@namastedev.com",
  });

```

Let say we want this information to be shown to our resto cards

How will we pass the data??

Props?

App.js do not have resto card so how to pass prop to resto card from App.js.

Resto card is inside body.js

So we pass userInfor to body.js and from body.js we pass our data to resto card.js as props

Now we use the userInfor in Resto cards.

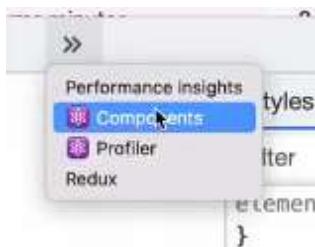


```
/**  
 * AppLayout  
 * (state=user)  
 *   - <Body user ={user}>  
 *     - RestaurantCard user={user}  
 *       - <h4>{user}</h4> You, i  
 */
```

So we are following a heirarchy and large apps, this hierarchy can grow a lot more

**When we pass data from parent to child and keep on doing it, This process is called Prop Drilling.**

[Download React Developer Tools](#)



Inside Components, it shows whole DOM tree in heirarchical fashion, it helps us in debugging our code.

We can click on a Restro Card and see what props is it getting so now we can debug very easily using dev tools.

Below is the example of **UI Layer and Data Layer**.

```

Search (text or /regex/)

Route.Provider
  AppLayout
    Header
    Outlet
      Context.Provider
        RenderedRoute
          Route.Provider
            Body
              Link key="121603"
                RestrauntCard
              Link key="201224"
                RestrauntCard
              Link key="118278"
                RestrauntCard
              Link key="527091"
                RestrauntCard
              Link key="107476"
                RestrauntCard

```

```

adTrackingID: "cid=5777394~p=1~eid=00000185-fc15-0e0b-1f78-995500560144"
address: "6/21,9TH CROSS ,1ST MAIN, VENKATESHWARA LAYOUT,SG PALYA, BENGALURU,
aggregatedDiscountInfo: {descriptionList: Array(2), header: "50% off", head...
area: "Tavarekere"
availability: {nextCloseMessage: "", nextOpenMessage: "", opened:...}
avgRating: "3.9"
badges: {imageBased: Array(0), textBased: Array(0), textExt...
chain: []
city: "1"
cityState: "1"
cloudinaryImageId: "bmwn4n4bn6n1tcpc8x2h"
costForTwo: 3000
costForTwoString: "₹300 FOR TWO"
cuisines: ["Kerala", "Chinese"]
deliveryTime: 27
favorite: false
feesMeta: {amount: "", fees: Array(0), icon: "", message: ""}, ...

```

We passed userInfor as props from App.js to body.js and In dev tools we can see that It also shows states, there are 3 states below because we have used 3 useState in our body.js.

It also shows we have also use an useEffect

```

props
  user: {email: "support@namastedev.com", name: "Namaste Re..."}
    email: "support@namastedev.com"
    name: "Namaste React"
    new entry: ""
  new entry: ""

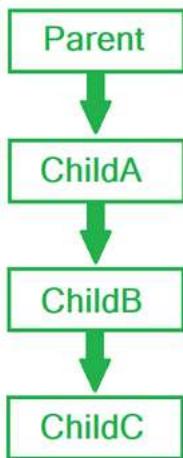
hooks
  1 State: [{}]
  2 State: [{}]
  3 State: ""
  4 Effect: f () {}

```

## Props Drilling

In the React ecosystem, it's common to divide an application into multiple components to

make it more modular and reusable. As the component hierarchy grows, **the challenge of passing data and state between components that are not directly nested arises**, known as “prop drilling”.



### What if we want to pass the data from child to parent. How to do it?

There are various ways to do it.

There are very less cases where we need data from children to parent.

We can use local storage but its not a good way.

We can build a custom hook and do it like we use "useParams" anywhere in the code.

Data Management is the most important thing if you want to make your app scalable so use your data efficiently.

Props can be another way, but as we saw in above example body.js do not need userInfor but we needed to pass it so if our hierarchy is very large then its not a good way.

Another disadvantage of props drilling is that, if we change infor in one component it will re-render so many component un-necessarily.

We can also use Redux (next lecture)

Another approach is

Let us say we want to show Instamart information inside Instamart component and we have an component <AboutInstaMart/> in it.

Another is <Details/>, <Teams/>, <Product/>, <Careers/>

```
You, 1 second ago | 1 author (You)
const Instamart = () => {
  return [
    <div>
      <AboutInstaMart/>
      <DetailsofInstaart/>
      <TeamInstamart/>
      <Product/>
      <Careers/>          You, 1 second ago
    </div>           ||
  ];
}

export default Instamart;
```

We can build different component for all these but we can also make an re-usable component.

```
const Section = ({ title }) => {      You, i
  return <h3>{title}</h3>;
};

const Instamart = () => {
  return (
    <div>
      <Section title="About Instamart" />

      {/* <AboutInstaMart/>
      <DetailsofInstamart/>
      <TeamInstamart/>
      <Product/>
      <Careers/> */}
    </div>
  );
};

export default Instamart;
```

Let us Give Heading and description also to it and pass it as a prop.

```
const Section = ({ title, description }) => {
  return (
    <div className="border border-black p-2 m-2">
      <h3>{title}</h3>
      <p>{description}</p>
    </div>
  );
};
```

```

const Instamart = () => {
  return (
    <div>
      <h1 className="text-3xl p-2 m-2 font-bold"> Instamart</h1>
      <Section
        title={"About Instamart"}
        description={"This is the about section of Instamart"}>
      </Section>
      <Section
        title={"Team Instamart"}
        description={[
          "This is the team section of Instamart. The team has 50 members...."]
      }>
      </Section>
      /* <AboutInstaMart/>
      <DetailsofInstamart/>
      <TeamInstamart/>
      <Product/>
      <Careers/> */
    </div>
  );
}

export default Instamart;

```



We see that the description would be a large description say of 200 words. And we want to make "show and hide" and not always show the description.

How will we do that?

We make a state for that

```

const Section = ({ title, description }) => {
  const [isVisible, setIsVisible] = useState(false);
  return (
    <div className="border border-black p-2 m-2">
      <h3 className="font-bold text-xl">{title}</h3>
      <button
        onClick={() => setIsVisible(true)}
        className="cursor-pointer underline"
      >
        Show
      </button>
      {isVisible && <p>{description}</p>}
    </div>
  );
};

```

Let us make it a Toggle Button, Show and Hide button both

```

const Section = ({ title, description }) => {
  const [isVisible, setIsVisible] = useState(false);
  return [
    <div className="border border-black p-2 m-2">
      <h3 className="font-bold text-xl">{title}</h3>
      <button
        onClick={() => setIsVisible(true)}
        className="cursor-pointer underline"
      >
        Show
      </button> You, 1 second ago · Uncommitted
      <button
        onClick={() => setIsVisible(false)}
        className="cursor-pointer underline"
      >
        Hide
      </button>
      {isVisible && <p>{description}</p>}
    </div>
  ];
};

```

The screenshot shows a website interface with the following structure:

- Section 1:** **About Instamart** (Section title)  
ShowHide (button)
- Section 2:** **Team Instamart** (Section title)  
ShowHide (button)
- Section 3:** **Careers** (Section title)  
ShowHide (button)

Below these sections is a footer area labeled "Footer".

Suppose Desc is already shown we do not want "show" button in UI and same with "hide" button.

```

const Section = ({ title, description }) => {
  const [isVisible, setIsVisible] = useState(false);
  return (
    <div className="border border-black p-2 m-2">
      <h3 className="font-bold text-xl">{title}</h3>
      {isVisible ? (
        <button
          onClick={() => setIsVisible(false)}
          className="cursor-pointer underline"
        >
          Hide
        </button>
      ) : (
        <button
          onClick={() => setIsVisible(true)}
          className="cursor-pointer underline"
        >
          Show
        </button>
      )}
      {isVisible && <p>{description}</p>}
    </div>
  );
};

```

We want to build a feature, If one section is already open and we open another section then it should automatically close the other section and open the new section for us.  
I mean Only section should open at a time. This is called Collapsible Accordion

We see every section has its own states and own props. Every time we create one section, it has its own State and props.

When isVisible of Section 1 is false, isVisible state of Section 2 and 3 are true  
So all these Section maintain their own state.

**What we want to do is, if we click on one Section, we need to modify state of its sibling also.**

How to do this in code?

We cannot directly modify state of another sibling from a component. We need to play smart here.

Instead of each section maintaining its own state, we can maintain these state in the parent and let parent control what to show and what to hide.

Analogy: Instead of boy, girls controlling their life, we give control of their life to their parents and now parents decide what boy or girl will do and what not.

This concept is known as Lifting the state up

So Parent Instamart will decide whose isVisible is true and whose isVisible is false.

```
const Section = ({ title, description }) => {
  const [isVisible, setIsVisible] = useState(false);
```

We declare this state inside our Instamart Component and pass it as prop to the section accordingly.

```
const Section = ({ title, description, isVisible }) => {
```

We want Parent to maintain state of its children.

We will create a state for each children.

```
const Instamart = () => {
  const [sectionConfig, setSectionConfig] = useState({
    showAbout: true,
    showTeam: false,
    showCareers: true,
  });
  ...
  return (
    <div>
      <h1 className="text-3xl p-2 m-2 font-bold"> Instamart </h1>
      <Section
        title={"About Instamart"}
        description={
          "On the other hand, we denounce with righteous indignation"
        }
        isVisible={sectionConfig.showAbout}
      />

      <Section
        title={"Team Instamart"}
        description={
          "On the other hand, we denounce with righteous indignation"
        }
        isVisible={sectionConfig.showTeam}
      />

      <Section
        title={"Careers "}
        description={
          "On the other hand, we denounce with righteous indignation"
        }
        isVisible={sectionConfig.showCareers}
      />

    /* <AboutInstaMart/>
```

Now we can control what to show and what to hide.

Now if we want to show only one and hide others.

We will setSectionConfig on onclick of show and hide button.

But "show" and "hide" button is inside Section component so now we want to pass data from child to parent. **How to do that?**

We pass **setIsvisible as props** and in each component we show that component and hide all

others.

```
<Section
  title={"Team Instamart"}
  description={[
    "On the other hand, we denounce with righteous
  ]}
  isVisible={sectionConfig.showTeam}
  setIsVisible={() =>
    setSectionConfig({
      showAbout: false,
      showTeam: true,
      showCareers: false,
    })
  }
/>

<Section
  title={"Careers "}
  description={[
    "On the other hand, we denounce with righteous
  ]}
  isVisible={sectionConfig.showCareers}
  setIsVisible={() =>
    setSectionConfig([
      showAbout: false,
      showTeam: false,
      showCareers: true,
    ])
  }
/>
```

Our section looks like

```
const Section = ({ title, description, isVisible, setIsVisible }) => {
  return (
    <div className="border border-black p-2 m-2">
      <h3 className="font-bold text-xl">{title}</h3>
      {isVisible ? (
        <button
          onClick={() => setIsVisible(false)}
          className="cursor-pointer underline"
        >
          Hide
        </button>
      ) : (
        <button
          onClick={() => setIsVisible(true)}
          className="cursor-pointer underline"
        >
          Show
        </button>
      )}
      {isVisible && <p>{description}</p>}
    </div>
  );
};
```

Suppose we have more sections, product section and details section

Now we again make state for them and do same thing again and again

```
const [sectionConfig, setSectionConfig] = useState([  
  showAbout: false,  
  showTeam: false,  
  showCareers: false,  
  showProduct: false,  
  showDetails: false  
]);  
  
setIsVisible={() =>  
  setSectionConfig([  
    showAbout: false,  
    showTeam: false,  
    showCareers: false,  
    showProduct: true, is false  
    showDetails: false is FontFaceSet  
  ]);  
}  
/>
```

This above code is Shit

It looks good from UI, it works fine also but code is not maintainable  
We just want to show one data at a time and we just need to track what we need to show.

Instead of maintaining whole sectionConfig we maintain what section will be visible  
We can keep indexes or keys for each section

```
const [visibleSection, setIsVisibleSection] = useState("about");
```

Now inside each section we can isVisible.  
visibleSection == "carrer" means if its career, pass true inside isVisible

```
<Section  
  title={"Careers "}  
  description={  
    "On the other hand, we denounce with right  
  }  
  isVisible={visibleSection === "career"}>
```

```

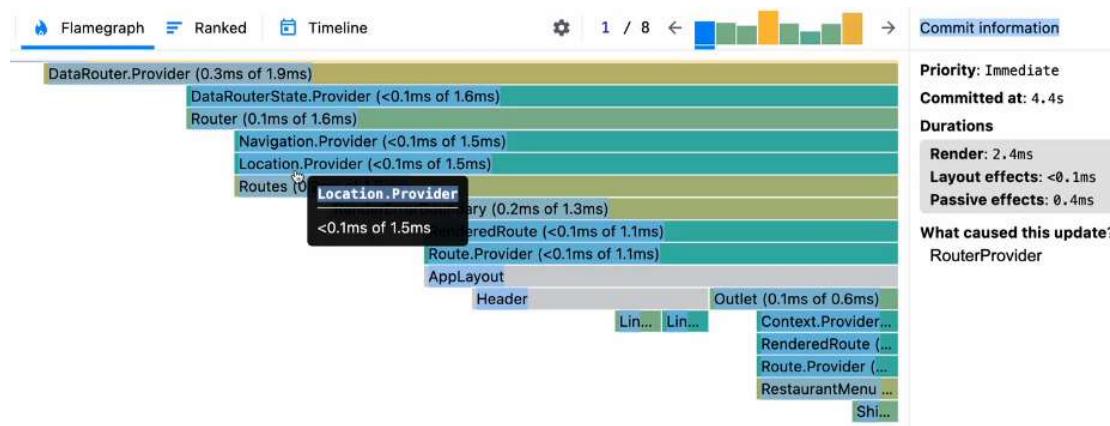
const Instamart = () => {
  const [visibleSection, setIsVisibleSection] = useState("team");
  return (
    <div>
      <h1 className="text-3xl p-2 m-2 font-bold"> Instamart</h1>
      <Section
        title={"About Instamart"}
        description={
          "On the other hand, we denounce with righteous indignation and dis"
        }
        isVisible={visibleSection === "about"} You, 2 minutes ago • Unc
        setIsVisible={() => setIsVisibleSection("about")}
      />

      <Section
        title={"Team Instamart"}
        description={
          "On the other hand, we denounce with righteous indignation and dis"
        }
        isVisible={visibleSection === "team"}
        setIsVisible={() => setIsVisibleSection("team")}
      />
    </div>
  );
}

```

**React dev tools also have** something known as **Profiler**

Profiler tracking and watching whatever we did in our website and tracks everything, tracks how much time it took any component to load and at what time which component loaded and once we stop it. We can rank and see which component takes more time to load and using it we can optimise things or uses these metrics to debug. It shows



**Suppose** we have a Logged In userInfo in our App.js

```

const AppLayout = () => {
  const [user, setUser] = useState({
    name: "Akshay Saini",
    email: "support@namastedev.com"
  });

```

Now say we want this userInfo in our card, then we need to do prop drilling.

Suppose we want this userInfo all across our app anywhere, header, footer, menu  
Then we need to pass our props everywhere but this is not a good practice.

If we have some piece of data that we need to render anywhere across our app, we need to store that data in some central storage, can be localStorage also but updating localStorage is a costly operation.

React gives us access to a central store called **Context**.

Some companies also use **Redux store**

So we will create Context for userInfo. We can also create a global variable but React will not be able to track it.

Let us create a context in utils folder called userContext



We can keep it anywhere but we should keep it in a separate file.

### How to create a context??

Import createContext from react

createContext is a function basically that takes in data that we need all across our application.

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

const UserContext = createContext({
  user: {
    name: "Dummy Name",
    email: "dummy@gmail.com",
  },
};

export default UserContext;
```

### How to use this context??

Let say we want to use this data inside Header.js

Let us go to our Header.js

Import useContext from react

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

useContext is just a hook which is a function at the end of the day.

Now to get the userInfo

Can we have multiple context in our app?

Yes, we can have context for card or for any other thing also which are needed all across our app.

```
import UserContext from "../utils/UserContext";
```

We are getting an object {user:{name,email}} so we extract user as {user} from our context

```
const [ user ] = useContext(UserContext);
```

Now we use it

```
</div>
<h1>{isOnline ? "✅" : "🔴"}</h1>
{user.name}
{isLoggedIn ? (
  You, 2 weeks ago
) : (
  <button onClick={() => setIsLoggedIn(true)}>
    Log In
  </button>
)}
</div>
```

Our context is not Bind to any component, its not tied to any component. Its in some central location.

State and props are tied to a component

Can we use Our user inside Restro Card?

Yes, let see how

```
import { IMG_CDN_URL } from "../contants";
import { useContext } from "react";
import UserContext from "../utils/UserContext";

const RestrauntCard = ({  
  name,  
  cuisines,  
  cloudinaryImageId,  
  lastMileTravelString,  
) => {  
  const { user } = useContext(UserContext);  
  return [  
    <div className="w-56 p-2 m-2 shadow-lg bg-pink-50">  
      <img src={IMG_CDN_URL + cloudinaryImageId} />  
      <h2 className="font-bold text-xl">{name}</h2>  
      <h3>{cuisines.join(", ")}</h3>  
      <h4>{lastMileTravelString}</h4>  
      <h5 className="font-bold">{user.name}</h5>  
    </div>  
  ];  
};  
  
export default RestrauntCard;
```

We can also put in Footer also.

```
import { useContext } from "react";
import UserContext from "../utils/UserContext";

const Footer = () => {
  const { user } = useContext(UserContext);

  return (
    <h4 className="p-10 m-10">
      This site is developed by {user.name} - {user.email}
    </h4>
  );
};

export default Footer;
```

Output

3 kms <b>Dummy Name -</b> dummy@gmail.com	Indian, North Indian, Chinese, Seafood 1.2 kms <b>Dummy Name -</b> dummy@gmail.com	2.9 kms <b>Dummy Name -</b> dummy@gmail.com	0.8 kms <b>Dummy Name -</b> dummy@gmail.com
			
<b>Anjappar</b> Chettinad, Thalis, Biryani, Chinese, Tandoor, South Indian, North Indian 0.6 kms <b>Dummy Name -</b> dummy@gmail.com	<b>Biryani Pot</b> North Indian, Biryani 1.8 kms <b>Dummy Name -</b> dummy@gmail.com	<b>Asha tiffins</b> Indian, South Indian, Beverages 4.4 kms <b>Dummy Name -</b> dummy@gmail.com	<b>Chickpet donne biriyani house</b> Biryani 1.2 kms <b>Dummy Name -</b> dummy@gmail.com
			
<b>Namaste</b> South Indian, Thalis, Snacks, Biryani, Indian, Chinese, Desserts, Beverages 4 kms <b>Dummy Name -</b> dummy@gmail.com			

This site is developed by Dummy Name - dummy@gmail.com

## Why to use props, if there are context already??

We do not use Context for everything, we use context for data which is needed in our whole app.

For Class Component how to do this?

```
import { Component } from 'react';
import UserContext from "../utils/UserContext";
```

We will use UserContext as a component in our class component.

We now use UserContext.Consumer and it takes a JS with a function which has value of our context.

```
<UserContext.Consumer>
  {(value) => console.log(value)}
</UserContext.Consumer>
```

```
Console was cleared
▼ {user: ...} ⓘ
  ▼ user:
    email: "dummy@gmail.com"
    name: "Dummy Name"
  ► [[Prototype]]: Object
  ► [[Prototype]]: Object

You, 6 seconds ago | 1 author (You)
import { Outlet } from "react-router-dom";
import ProfileFunctionalComponet from "./Profile";
import Profile from "./ProfileClass";
import { Component } from "react";
import UserContext from "../utils/UserContext";

You, 6 seconds ago | 1 author (You)
class About extends Component {
  constructor(props) {
    super(props);

    //console.log("Parent - constructor");
  }
  componentDidMount() {
    // Best place to make an Api call
    //console.log("Parent - componentDidMount");
  }
  render() {
    //console.log("Parent - render");
    return [
      <div>
        <h1>About Us Page</h1>

        <UserContext.Consumer>
          {(value) => console.log(value)}
        </UserContext.Consumer>

        <p>      You, last week + chapter-08 ~
          | This is the Namaste React Live Course Chap
        </p>
        <Profile />
    ];
  }
}
```

```
<UserContext.Consumer>
  {|({ user }) => <h4 className="font-bold text-xl p-10">{user.name}- {user.email}</h4>}
</UserContext.Consumer>
```

Suppose we have an API in app.js that gets us the data of the logged IN user and that data is dynamic.

Now we need to pass the dynamic data to our context. **How to do it??**

We have something as **userContext.Provider**

```

const AppLayout = () => {
  const [user, setUser] = useState({
    name: "Akshay Saini",
    email: "support@namastedev.com",
  });

  return (
    <UserContext.Provider>
      <Header />
      <Outlet />
      <Footer />
    </UserContext.Provider>
  );
};

```

We can now pass our data as prop in value attribute

```

const [user, setUser] = useState({
  name: "Akshay Saini",
  email: "support@namastedev.com",
});

```

Say our API use the setUser and sets the user which we pass as our prop to the context. We wrap Header, Footer, Outlet inside Provider so that everybody of them can use our context.

```

<UserContext.Provider
  value={[
    user,
  ]}
>
  <Header />
  <Outlet />
  <Footer />
</UserContext.Provider>

```

Suppose Header is outside Provider. It takes context value from Utils context  
So We can modify the context data for different components.

**If we put something inside Provider and pass data as value, then that data will pass to those wrapped inside Provider. Else, data is passed from utils ka createContext if we have used it inside our Header.**

**Let us create a input field near search box which modifies our context dynamically in our app.**

We have search bar in our body.js

We will control our Context from body.js now

Let us first create input filed whose value comes from context.

```
    Search
  </button>
  <input value={user.name}></input>
```

But we need to put a onChange to be able to modify it.

We need to set our context inside our onChange.

### How to do it?

Just like, we have passed user, we can also pass setUser from App.js to our Provider

```
<UserContext.Provider
  value={{ user,
  setUser
}}
>
  <Header />
  <Outlet />
  <Footer />
</UserContext.Provider>
/>
```

Now we get setUser inside body.js

```
const { searchContext, setSearchContext } = useState();
const [user, setUser] = useContext(UserContext);
```

```
<input value={user.name} onChange= {
  e => setUser([
    name: e.target.value,
    email: "newemail@gmail.com",
  ])
}></input>
```

To track our context by name inside our dev tools we can do something like this inside our file where we are making context.

UserContext.displayName = "UserContext"

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

const UserContext = createContext({
  user: {
    name: "Dummy Name",
    email: "dummy@gmail.com",
  },
});
UserContext.displayName = "UserContext";
export default UserContext;
```

Now our react dev tool recognizes this context by name "UserContext" and It helps us in

debugging where we have many contexts.



## Can we have Nested Contexts?? Means Context inside Context??

Yes, it is possible to override the value of a context by another context inside it.

```
import React, { createContext, useContext } from 'react';

export const AnalyticsContext = createContext({
  category: 'Page'
});

const ButtonInCtx = () => {
  const { category } = useContext(AnalyticsContext);
  return <button type='button'>Context: {category}</button>;
};

export default function App() {
  return (
    <AnalyticsContext.Provider value={{ category: 'Page' }}>
      <AnalyticsContext.Provider value={{ category: 'Header' }}>
        <ButtonInCtx />
      </AnalyticsContext.Provider>
      <AnalyticsContext.Provider value={{ category: 'Body' }}>
        <ButtonInCtx />
      </AnalyticsContext.Provider>
      <ButtonInCtx />
      <AnalyticsContext.Provider value={{ category: 'Footer' }}>
        <ButtonInCtx />
      </AnalyticsContext.Provider>
    </AnalyticsContext.Provider>
  );
}
```

When you nest a context provider inside a context provider of itself, you can override the previous context.

```
const CarouselProvider = ({ children }) => {
  const { category } = useContext(AnalyticsContext);

  return (
    <AnalyticsContext.Provider value={{ category: `${category} - Carousel` }}>
      {children}
    </AnalyticsContext.Provider>
  );
};
```

**UI Layer and Data Layer are different.**

**Data Layer always stays there only the UI layer does the Reconciliation and all things**

## Session 14

### React Redux

More Efficient and more complex way of handling data inside our app.

For managing large data for a production ready application, Companies use a library known as Redux.

#### Why did we use context?

To avoid prop drilling.

Context is a central place which has some data which can be accessed anywhere in the app by any component.

We can make context for anything and any number of context.

We can use context using dark/light mode.

When our application grows in size, redux comes to play.

We can create a redux store.

Redux is used for data management.

But there are some issues of using redux.

For small applications, context is ok

But if we are making many contexts then instead of context, we should use redux as it gives more efficient way of managing our data but redux has some disadvantages also.

It is very complex to setup, it has huge learning curve, It has lot of code to be done through copy-pasting.

If your application is small, do not use redux as it comes with lot of complexities.

We need lot of libraries and tools like bundlers, testing packages, react router dom etc etc to make an successful production level application. React alone cannot help we need other libraries.

Earlier, Redux was difficult to understand so redux developers came with "**Redux Toolkit**"

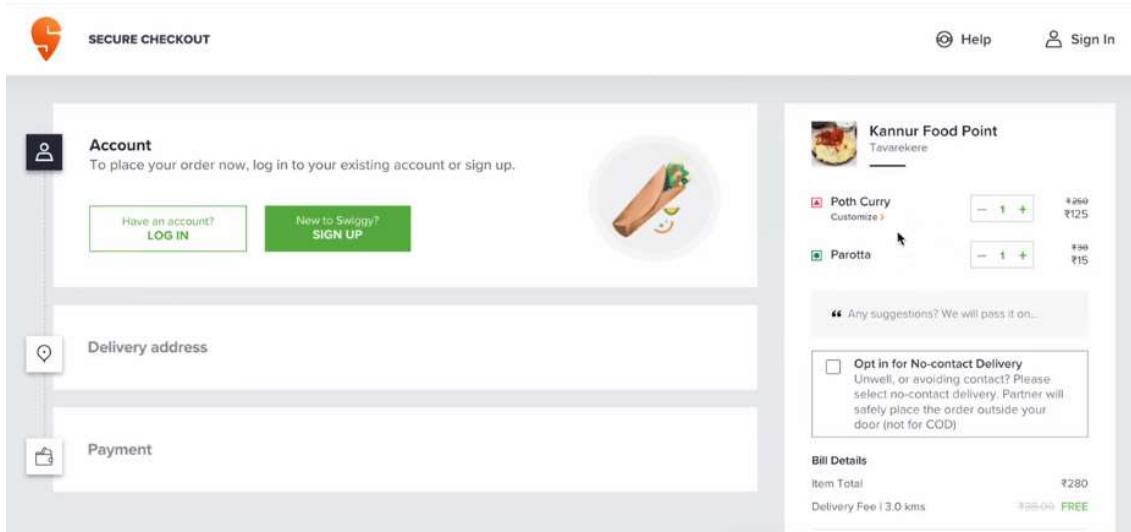
The **Redux Toolkit** package is intended to be the standard way to write **Redux** logic. It was originally created to help address three common concerns about **Redux**:

- "Configuring a Redux store is too complicated"
- "I have to add a lot of packages to get Redux to do anything useful"
- "Redux requires too much boilerplate code"

## Architecture of Redux

### What we will build today?

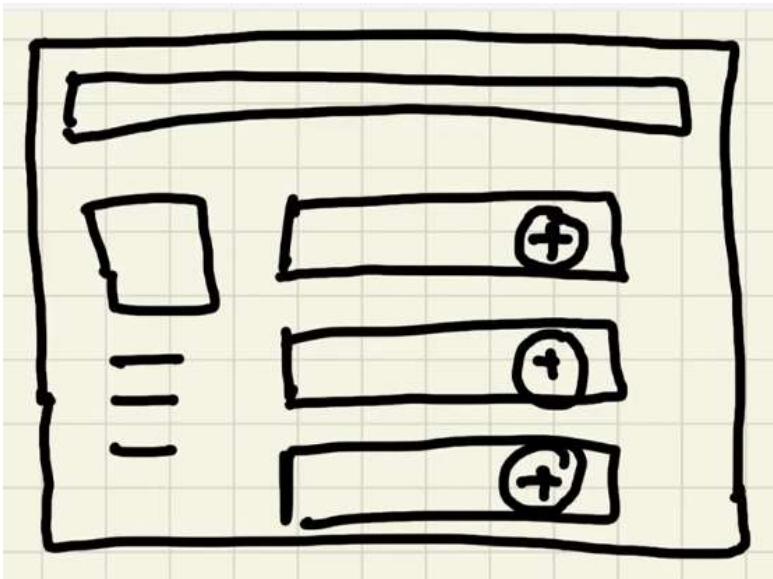
Today, we will build the cart for food order.



Let say we click any restaurant, we go to its menu.

We have header, we have a cart in it which shows number of products in our cart

We have resto menu and add button to each menu item



## Redux Store

Redux store is a big object which has small sections. It is accessible to all the components.

When we create a state variable it has scope of that component only.

Props can be passed to other components but it is also component dependent.

Store and context are separate entity out of our application. They are independent. They can be accessed from anywhere.

**Can we have multiple context?** Yes

But in Redux we have only one store which has data regarding everything. Its like a big store with all the products.

**We will create slices / logical separation of our store.**



**What all slice can our app has?**

It can have a user slice, auth slice, theme slice, cart slice

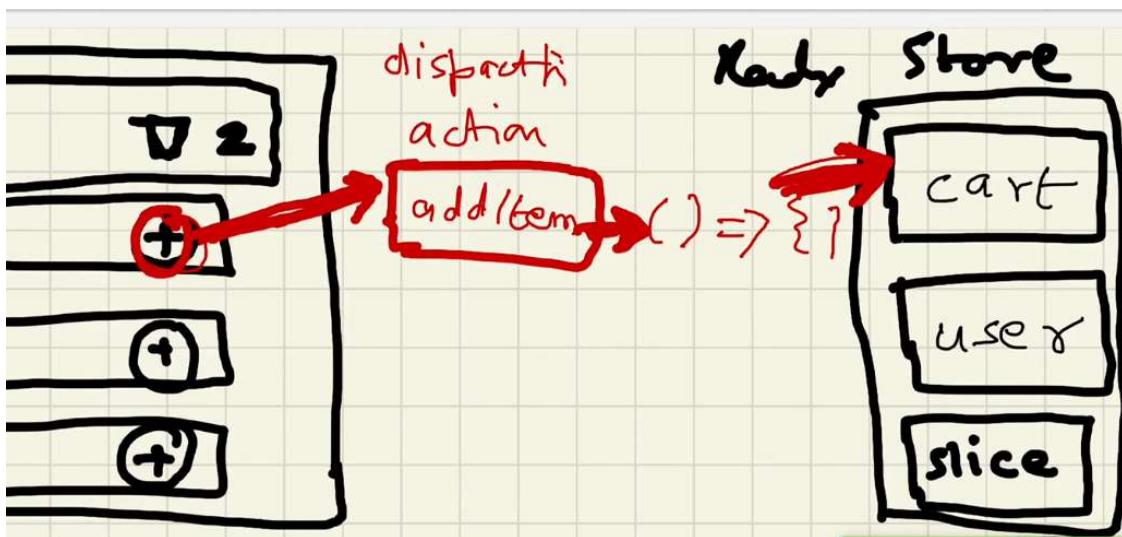
So we can have multiple such slices. Slice is a small portion of our store.

Our component cannot directly modify the store.

We have to dispatch an action, action say "ADD\_ITEM" say when we click + button in our menu card, it **dispatch an action** called "ADD\_ITEM"

Now, this action will call a function which is a normal JS function and this function will modify our cart which is there in the store.

We cannot directly modify our cart. We dispatch an action for it.

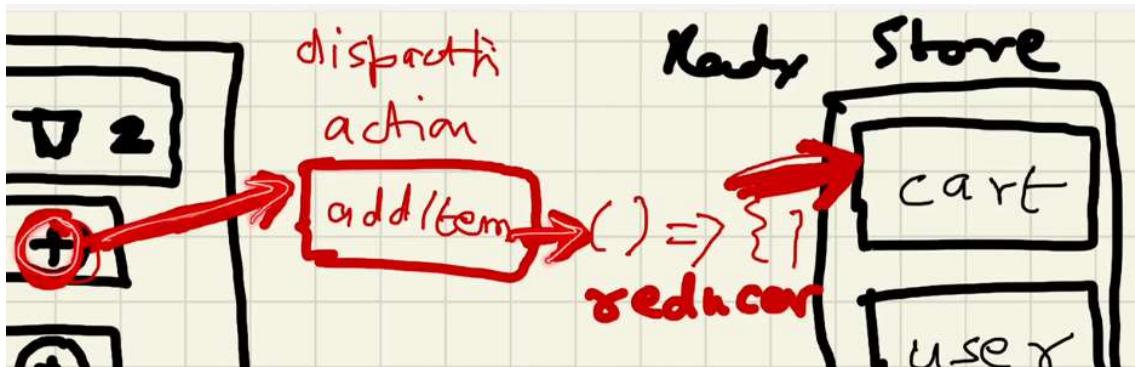


### Why can't we directly modify our store?

In a large application, we do not want any random component to modify our store. We want a process to be associated with it.

Its same as having an Micro-services architecture, it helps in separation of concern.

The function we are using to modify the cart is known as **Reducer**



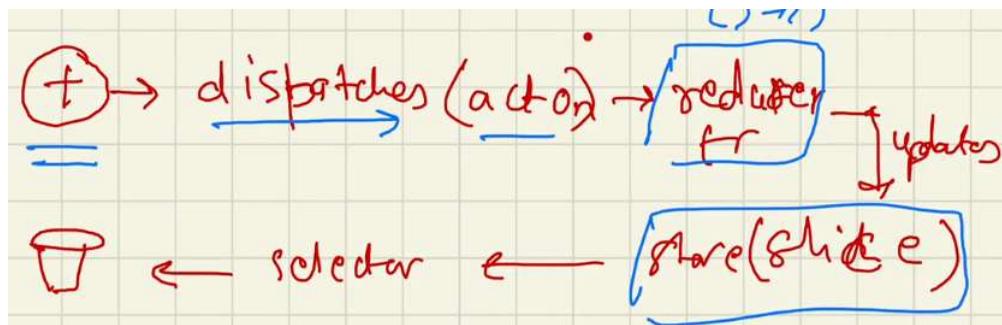
When we click the + button in menu card of an food item, it dispatches an action which calls the reducer function and updates the slice of our redux store.

How to get the Number near our cart icon in header??

There is something known as **Selector**.

If we want to read Cart we will call Selector and it gives the reading to us.

Selector means selecting a portion of the store. It is used to read data.  
At the end of the day, Selector is a hook and hooks are just normal functions.



When we use Selector, it is known as **Subscribing to the Store**  
Means Cart component has subscribed to the store means it is reading from the store now, whenever our cart modifies in the store, it automatically modifies cart in our UI also.

Let us now **code** this,

First of all, we install Redux toolkit in our project.

```
$ npm i @reduxjs/toolkit
```

We will install one more library

```
$ npm i react-redux
```

### Why 2 libraries??

Redux core job is to manage the store so first library does that and second library is a bridge between react and redux. Same as react and ReactDOM.

Some things we will import from one library and some things will be imported from other library.

We are creating store.js in utils folder.



For store we use redux toolkit.

```
JS store.js U X
src > utils > JS store.js > ...
1 import { configureStore } from "@reduxjs/toolkit";
2
3 const store = configureStore({});
4
5 export default store;
6
```

Now, we have created the store which has an object which contains our slices.  
Now, we need a Provider and we wrap our application app.js in it.

Go to app.js

Import Provider from "react-redux"

```
import {Provider} from "react-redux";
```

Same as userContext.Provider we use Provider and pass store to it.

Import store and pass it as prop

```

return (
  <Provider store={store}>
    <UserContext.Provider
      value={{ 
        user: user,
        setUser: setUser,
      }}
    >
      <Header />
      <Outlet />
      <Footer />
    </UserContext.Provider>
  </Provider>
);

```

Now we are ready for action, reducer but first, let us fill something in our store and create Slices.

We can create slices in store.js but we will create a different file inside utils named "cartSlice.js"

We will use "**createSlice**" in which we put name of slice, We give **initialState (empty array for now [ ])**

```

JS store.js U      JS cartSlice.js U ●  JS App.js M
src > utils > JS cartSlice.js > [o]cartSlice > ↴ initialState
1   import { createSlice } from "@reduxjs/toolkit";
2
3
4   const cartSlice = createSlice({
5     name: 'cart',
6     initialState: [
7       items: [],
8     ],
9   })

```

### How to modify our cart?

By creating a reducer function so we create our reducers now,

### When are these reducers called?

On dispatching of an action, reducers are called.

addItem is one of the action inside our reducer

Reducer function **takes state and action**

**State is previous state of our app and action is data which is coming in so we modify the previous state by pushing data from action into state.**

Let us write some logic to put data inside state.

```
import { createSlice } from "@reduxjs/toolkit";

const cartSlice = createSlice({
  name: 'cart',
  initialState: {
    items: [],
  },
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
    }
  }
});
```

We can have more actions and reducers like `clearCart` which makes our state empty again. These functions like `addItem`, `clearCart` do not return anything. They just take the state and modify it.

```
import { createSlice } from "@reduxjs/toolkit";

const cartSlice = createSlice({
  name: "cart",
  initialState: {
    items: ["Banana", "Apples"],
  },
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
    },
    clearCart: (state) => {
      state.items = [];
    },
  },
});
```

Let us create `removeItem` also

```
import { createSlice } from "@reduxjs/toolkit";

const cartSlice = createSlice({
  name: "cart",
  initialState: {
    items: ["Banana", "Apples"],
  },
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
    },
    removeItem: (state, action) => {
      state.items.pop();
    },
    clearCart: (state) => {
      state.items = [];
    },
  },
});

export default cartSlice.reducer;
```

We have created our slice, now we will export the actions and reducers from this slice.

**It will be export cartSlice.reducer not export cartSlice.reducers**

**We will also export the actions we have created.**

**Actions are exported by name and Reducer are exported by default.**

```
import { createSlice } from "@reduxjs/toolkit";

const cartSlice = createSlice({
  name: "cart",
  initialState: {
    items: ["Banana", "Apples"],
  },
  reducers: {
    addItem: (state, action) => {
      state.items.push(action.payload);
    },
    removeItem: (state, action) => {
      state.items.pop();
    },
    clearCart: (state) => {
      state.items = [];
    },
  },
});

export const { addItem, removeItem, clearCart } = cartSlice.actions;

export default cartSlice.reducer;
```

Our store does not know about our slice so we need to put this slice in our store.  
We exported reducer by default, we will put all reducer inside our store

```
src > utils > JS store.js > [e] store > ↴ reducer
  1 import { configureStore } from "@reduxjs/toolkit";
  2 import cartSlice from "./cartSlice";
  3
  4 const store = configureStore({
  5   reducer: {
  6     cart: cartSlice,
  7   },
  8 });
  9
 10 export default store;
 11
```

## Steps we have followed till now:

```
/*
 * Slice
 * - RTK - createSlice({
 *   name: "",
 *   initialState:
 *   reducers: {
 *     addItem: (state, action) => { state = action.payload }
 *   }
 * })
 * export const {addItem, removeItem} = cartSlice.actions;
 * export default cartSlice.reducer;
 *
 * Put that Slice into Store
 * - {
 *   reducer: {
 *     cart: cartSlice,
 *     user: userSlice
 *   }
 * }
 */
*/
```

Let us now **Subscribe to our store**

We go to Header component and show some items in front of our cart.

```
<li className="px-2">Cart- 4 items</li>
```

We will use **useSelector hook** from "react-redux" which acts as bridge between react and redux.

```
const cartItems = useSelector(store => store.cart.items);
```

```
<li className="px-2">Cart- {cartItems.length} items</li>
```

Cart- 2 items

Now we go to Food Menu component and add a + button with food items and dispatch an action

```
<div className="menu">
  <div>
    <h1>Restraunt id: {resId}</h1>
    <h2>{restaurant?.name}</h2>
    <img src={IMG_CDN_URL + restaurant?.cloudinaryImageId} />
    <h3>{restaurant?.area}</h3>
    <h3>{restaurant?.city}</h3>
    <h3>{restaurant?.avgRating} stars</h3>
    <h3>{restaurant?.costForTwoMsg}</h3>
  </div>
  <div>
    <button className="p-2 m-5 bg-green-100">Add Item</button>
  </div>
  <div>
    <h1>Menu</h1>
    <ul>
      {Object.values(restaurant?.menu?.items).map((item) => (
        <li key={item.id}>{item.name}</li>
      ))}
    </ul>
  </div>
</div>
;
```

Let us see what this button does onClick

```
<button
  className="p-2 m-5 bg-green-100"
  onClick={() => handleAddItem()}
>
  Add Item

```

"Grapes" is an Payload and addItem is our action.

```
const handleAddItem = () => {
  dispatch(addItem("Grapes"));
}
```

addItem comes from cartSlice

```
import { addItem } from "../utils/cartSlice";
```

Dispatch comes from another important hook known as useDispatch() imported from react-redux.

```
import { useDispatch } from "react-redux";
```

```
const dispatch = useDispatch();

const handleAddItem = () => {
  dispatch(addItem("Grapes"));
};
```

There is something known as **Cache Behaviour in Redux** also

Cache behavior in React and Redux typically refers to how data is stored and managed within a Redux store and how that data is accessed and updated by React components. Redux is a state management library commonly used with React to manage the state of an application in a predictable and centralized manner.

Here's a breakdown of cache behavior in React Redux:

1. **State Caching:** Redux stores the application's state in a single JavaScript object called the "store." This store acts as a cache of the application's data. When you dispatch actions to update the state, Redux updates this cache by applying reducers, which are pure functions responsible for specifying how the state should change in response to actions.
2. **Immutable Data:** Redux encourages the use of immutable data structures. This means that the data stored in the Redux store should not be directly mutated. Instead, a new state object is created with the changes, which helps in tracking changes and simplifies debugging.
3. **Selectors:** To access data from the Redux store, React components can use selectors. Selectors are functions that retrieve specific pieces of data from the store's cache. They allow you to encapsulate the logic for accessing data, which helps in keeping the components decoupled from the specific structure of the state.
4. **Memoization:** To optimize performance, selectors often use memoization techniques, such as the reselect library, to ensure that the same output is returned for the same input values without re-computing the result. Memoization can significantly reduce unnecessary calculations and re-renders in React components.
5. **Subscription and React-Redux:** React components can subscribe to changes in the Redux store using the useSelector hook or the connect higher-order component provided by the react-redux library. This subscription mechanism ensures that components re-render when relevant parts of the state change, keeping the UI in sync with the cached data.
6. **Caching Remote Data:** In some cases, you might want to cache remote data fetched from an API in the Redux store to avoid unnecessary network requests. This involves storing fetched data in the Redux store and updating it when needed, such as when the data is stale or when a specific action triggers a refresh.

Let us now put this Add Item in all of our food item cards

```
    <div>
      <button
        className="p-2 m-5 bg-green-100"
        onClick={() => handleAddItem()}
      >
        Add Item
      </button>
    </div>
    <div>
      <h1>Menu</h1>
      <ul>
        {Object.values(restaurant?.menu?.items).map((item) => (
          <li key={item.id}>
            {item.name} - <button className="p-1 bg-green-50" onClick={() => addFoodItem(item)}>Add</button>
          </li>
        ))}
      </ul>
    </div>
  </div>
);

export default RestaurantMenu;
```

```
<div className="p-5">
  <h1>Menu</h1>
  <ul>
    {Object.values(restaurant?.menu?.items).map((item) => (
      <li key={item.id}>
        {item.name} -{" "}
        <button className="p-1 bg-green-50" onClick={() => addFoodItem(item)}>
          Add
        </button>
      </li>
    ))}
  </ul>
</div>
/div>
```

We pass item to the function so that we can add actual item inside our state.

```
const dispatch = useDispatch();

const addFoodItem = (item) => {
  dispatch(addItem(item));
};
```

Let us build a cart component.

```
✓ NAMASTE-REACT-LIVE
  > .parcel-cache
  > dist
  > node_modules
  ✓ src
    > assets
    > components
      JS About.js
      JS Body.js
      JS Cart.js
      JS Contact.js
      JS Error.js
      JS Footer.js
      JS Header.js
      JS Instamart.js
      JS Profile.js
      JS ProfileClass.js
      JS RestaurantCar...
      JS Restraunt...
      JS Shimmer.js
```

Add the Cart page in routes

Let us Link it in the Header also using <Link> from react-router-dom

```
[  
  {  
    path: "/cart",  
    element: <Cart/>  
  },  
]
```

Now let us get our Food items from our store using useSelector in Cart Component.

useSelector is used to subscribe to the store. This is the place where we tell what is it subscribing to.

We do useSelector(store => store.cart.items) it tells that we are subscribing to just the items inside store that's it.

**This will also work but it gives us access to store**

```
const store = useSelector((store) => store);
```

The above code degrades our app performance because now, as we are subscribed to our store. An store has different slices so everytime store changes, Our cart component reloads which creates an Unnecessary load in the app. In a bigger app, store is huge so it's a huge performance issue.

**Performance Optimisation, For Performance optimisation, we subscribe to items array only**

```
const cartItems = useSelector((store) => store.cart.items);
```

**Using Above code, we only Subscribe to what is needed to us. Subscribe to only the specific portion of our app. This makes major optimisation in performance of the app.**

Let us now, show the menu items also

We make another component FoodItem to show "Added food items in the cart"

The screenshot shows a code editor with a dark theme. On the left is a tree view of the project structure:

- NAMASTE-REACT-LIVE
  - .parcel-cache
  - dist
  - node\_modules
  - src
    - assets
    - components
      - About.js
      - Body.js
      - Cart.js
      - Contact.js
      - Error.js
      - FoodItem.js
      - Footer.js
      - Header.js
      - Instamart.js
      - Profile.js
      - ProfileClass.js
      - RestaurantCard.js
      - RestrauntMenu.js
      - Shimmer.js

On the right is the content of the FoodItem.js file:

```
src > components > JS FoodItem.js > ...
1 import { IMG_CDN_URL } from "../contants";
2
3 const FoodItem = ({ name, description, cloudinaryImageId, price }) => {
4   return (
5     <div className="w-56 p-2 m-2 shadow-lg bg-pink-50">
6       <img src={IMG_CDN_URL + cloudinaryImageId} />
7       <h2 className="font-bold text-xl">{name}</h2>
8       <h3>{description}</h3>
9       <h4>Rupees: {price / 100}</h4>
10    </div>
11  );
12};
13
14 export default FoodItem;
15
```

Let us now import FoodItem component inside Cart component

```

import { useSelector } from "react-redux";
import FoodItem from "./FoodItem";

const Cart = () => {
  const cartItems = useSelector((store) => store.cart.items);

  //const store = useSelector((store) => store);

  return [
    <div>
      <h1 className="font-bold text-3xl"> Cart Items - {cartItems.length}</h1>
      <div className="flex">
        {cartItems.map((item) => (
          <FoodItem key={item.id} {...item} />
        ))}
      </div>
    </div>
  ];
};

export default Cart;

```

Let us make a **Clear Cart Button**

```

<button
  className="btn-green-100 p-2 m-5"
  onClick={() => handleClearCart()}
>
  Clear Cart
</button>

```

```

const dispatch = useDispatch();

const handleClearCart = () => {
  dispatch(clearCart());
}

```

## Redux Devtools

It tells us whether website we are in uses redux or not.



- **Middleware in Redux:**

Middleware in Redux is a piece of software that sits between the action dispatch and the reducer. It allows you to intercept, process, and potentially modify actions before they reach

the reducer. Middleware is commonly used for handling asynchronous operations, logging, routing, and other side effects that are not pure functions.

Popular middleware in Redux includes Redux Thunk, Redux Saga, and Redux-observable.

Middleware enables you to write asynchronous logic while keeping your reducers pure and predictable.

- **Redux Thunk:**

Redux Thunk is a popular middleware for handling asynchronous actions in Redux. It allows you to dispatch functions (thunks) instead of plain action objects. These thunk functions can contain asynchronous code and have access to the dispatch and getState functions, which makes it easy to perform asynchronous operations like making API requests.

- **RTK Query:**

RTK Query is part of Redux Toolkit (RTK), which is an official package from the Redux team to simplify common Redux patterns and reduce boilerplate code. RTK Query is designed specifically for handling API requests and state management in a more streamlined and standardized way.

RTK Query provides a powerful API for defining API endpoints and automatically generating actions, reducers, and selectors for data fetching, caching, and updating. It abstracts away much of the manual setup required for typical Redux data fetching.

RTK Query simplifies data fetching and state management by generating much of the necessary Redux code for you, resulting in a more declarative and efficient way to work with APIs in your application.

## Session 15

**How to write test cases for our react app?**

**What is jest?**

**What is React testing library?**

React Testing library uses jest behind the scenes.

In create-react-app we already have testing files.

Why do we need test cases?

Generally, when we make a large scale application, There are lot of components, data passing etc etc and these components are integrated.

Sometimes, some component might break the code for another component.

We write test cases to check if our new feature/code is not breaking the existing code.

To make sure our existing code is working fine after adding new code

It also makes the code maintainable so that a large team can work on it.

### Test Driven Development

**We write test cases even before we write our code. It is very good but companies does not use this approach generally as this approach makes development process very slow although it enhances the development experience.**

We can write test case for "check sum of 2 numbers" which will try to cover our most of the

test cases.

```
test("check sum of 2 numbers", () => {
  expect(sum(2,3)).toBe(5);
  expect(sum(-2,3)).toBe(1);
  expect(sum()).toBe("Please check arguments");
})
```

Now we will write our sum function

```
export const sum = (a, b) => a + b;
```

### What are different types of Testing?

1. **Manual Testing:** It requires humans to test. Whole testing team is setup.
2. **Automated Testing:** Code is testing a code. There are no humans involved like Tools are Selenium for Java.
3. **End to End Testing:** We will test the whole flow of code from user login to everything. It covers entire user journey. In most of the companies this part is given to QAE team. There is something known as Headless Browser, which is kind of like a actual browser but it does not have an UI. This Headless Browser is used for testing and executing test cases faster as there is no UI so now painting of DOM is required, it is used where We do not need UI to test. UI is important for Manual Testing.
4. **Unit Testing:** It's the core job of developers, it means testing small units. Is our logo coming properly or not, Is our card coming properly or not.
5. **Intregation Testing:** It means testing the integration between the components.

There are so many types of other testing also but as a developer, **we are more concerned about unit testing and intregation testing.** For other types of testing there is a dedicated team.

Testing is just like development, it takes time to learn but knowing the basics is important.

## Jest

To test JS code we use jest.



Jest is a full-fledged framework in itself but React testing library uses Jest behind the scenes.

We will be using react testing library which makes testing very easy for us in a react app.

Let us setup the react testing library in react.

```
npm install --save-dev @testing-library/react
```

Along with React testing library we also intall Jest

```
npm i -D jest
```

In package.json

```
"@testing-library/react": "^13.4.0",
"babel-plugin-transform-remove-console": "^6.9.4"
"jest": "^29.4.1",
```

Now, we will configure jest.

We will create a jest.config file

Or just to **npx jest --init**

Jest is a package, We do npx instead of npm because we need to execute it once.

```
npx jest --init
```

Now it will ask some questions

If using TSX say "Yes"

Node/jsdom (browser-like) -> go with jsdom

Do you want to add coverage report -> yes (recommended)

Which provider to use for coverage -> V8 / babel -> we use babel

Automatically clear mock calls -> yes

Now jest.config.js is created

```
✓ NAMASTE-REACT-LIVE
  JS Body.js
  JS Cart.js
  JS Contact.js
  JS Error.js
  JS FoodItem.js
  JS Footer.js
  JS Header.js
  JS Instamart.js
  JS Profile.js
  JS ProfileClass.js
  JS RestaurantCard.js
  JS RestrauntMenu.js
  JS Shimmer.js
  JS sum.js      U
> 📁 utils
  JS App.js
  JS contants.js
  .babelrc
  .gitignore
  .postcssrc
  index.css
  index.html
  jest.config.js      U
  package-lock.... M
  package.json      M
  tailwind.config.js
```

It contains all options we choose.

### How to run these test cases?

In package.json we have a test command,

```
▷ Debug
"scripts": {
  "start": "parcel index.html",
  "build": "parcel build index.html",
  "test": "jest"      You, last month
},
```

So we write "**npm run test**"

We get these errors

```
● Validation Error:
Test environment jest-environment-jsdom cannot be found. Make sure the testEnvironment configuration option is set in your test code module.

Configuration Documentation:
https://jestjs.io/docs/configuration

As of Jest 28 "jest-environment-jsdom" is no longer shipped by default, make sure to install it separately.
akshaysaini@Akshay's-MacBook-Pro namaste-react-live %
```

### How to fix this error?

```
As of Jest 28 "jest-environment-jasmine" is no longer shipped by default, make sure to install it separately.
```

We need to install jest environment jsdom

```
npm i -D jest-environment-javascript
```

```
base plugin transform remove context
"jest": "^29.4.1",
"jest-environment-javascript": "^29.4.1",
```

Now again "**npm run test**"

It tells I checked on 30 files and did not find any test

```
No tests found, exiting with code 1
Run with `--passWithNoTests` to exit with code 0
In /Users/akshaysaini/Desktop/namaste-react-live
  30 files checked
    testMatch: **/_tests_/**/*.[jt]s?(x), **/?(*.)+(spec|test).[tj]s?(x) - 0 matches
    testPathIgnorePatterns: /node_modules/ - 30 matches
    testRegex: - 0 matches
    Pattern: - 0 matches
```

We need to create our first test file

Inside components, we make a folder named "**\_tests\_**" (name is necessary)

Now jest will recognise it as a testing file.

### What is **\_ name \_ Dunder**

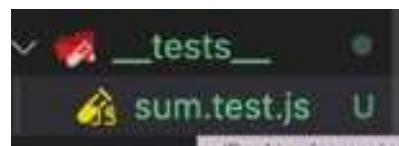
**the name that we give, to attributes whose names start and end with a double underscore.** So we call above **\_tests\_** as dunder test file.

Jest is a JS testing Framework so **Let us first see JS testing**

Let us write test cases for our sum file

We create **sum.test.js** inside **\_tests\_**

There is a **convention** of naming test files as **something.test.js** so remember it



If we write any file name inside **\_tests\_** folder or something.test.js then jest easily finds that it's a test file.

Now Let us write inside sum.test.js

```
Test("name of test case", function which will have code that our test case will execute);
```

Inside function, we will expect something, every test case should have some expectations.

```
test("Check sum of 2 positive numbers", () => {
  | expect(sum(2, 5)).toBe(5);
});
```

We are using sum function so we import sum from sum.js

```
import { sum } from "../sum";

test("Check sum of 2 positive numbers", () => {
  | expect(sum(2, 5)).toBe(5);
});
```

We do not need to import "test" and "expect" as Jest understands it automatically.

Now Let us run "npm run test"

```
SyntaxError: Cannot use import statement outside a module
```

We get above error because we have used "import" inside our JS file and our normal JS file do not understand "import"

We will take help of babel in this, What we will do is

Configure Babel with jest

The screenshot shows the Jest documentation page with the 'Using Babel' section highlighted. The left sidebar has a 'Getting Started' link under 'Introduction'. The main content area shows instructions to install Babel dependencies using npm or Yarn, and provides a template for a babel.config.js file.

```
npm install --save-dev babel-jest @babel/core @babel/preset-env
```

```
module.exports = {
  presets: [['@babel/preset-env', {targets: {node: 'current'}}]],
};
```

Let us install these packages and we configure it so that babel will tell Jest that there is something called module.

```
babel.config.js
```

```
module.exports = {
  presets: [['@babel/preset-env', {targets: {node: 'current'}}]],
};
```

Now, either we can make "babel.config.js" or we can write our code in ".babelrc" which we

made already



In our .babelrc, it gives error although it's a valid JS object

```
.babelrc > [ ]presets
You, 1 second ago | 1 author (You)
1  {
2    |   presets: [[ "@babel/preset-env", { targets: { node: "current" } } ]]
3  }
```

Why??

.babelrc requires a JSON and JS object & JSON are different

## JavaScript Objects VS JSON

Though the syntax of JSON is similar to the JavaScript object, JSON is different from JavaScript objects.

JSON	JavaScript Object
The key in key/value pair should be in double quotes.	The key in key/value pair can be without double quotes.
JSON cannot contain functions.	JavaScript objects can contain functions.
JSON can be created and used by other programming languages.	JavaScript objects can only be used in JavaScript.

## Converting JSON to JavaScript Object

You can convert JSON data to a JavaScript object using the built-in **JSON.parse()** function.

## Converting JavaScript Object to JSON

You can also convert JavaScript objects to JSON format using the JavaScript built-in **JSON.stringify()** function.

## Use of JSON

JSON is the most commonly used format for **transmitting data (data interchange) from a server to a client and vice-versa**. JSON data are very **easy to parse and use. It is fast** to access and manipulate JSON data as they only contain texts.

JSON is **language independent**. You can create and use JSON in other programming languages too.

So we convert it to JSON by giving " " in everything as JSON takes double comma in key-value pairs.

```
{  
  "presets": [[ "@babel/preset-env", { "targets": { "node": "current" } } ]]
```

Now we have configured our babel

Let us run "npm run test"

Again error

```

FAIL src/components/__tests__/sum.test.js
  ✕ Check sum of 2 positive numbers (1 ms)

● Check sum of 2 positive numbers

  expect(received).toBe(expected) // Object.is equality

    Expected: 5
    Received: 7

      2 |
      3 | test("Check sum of 2 positive numbers", () => {
> 4 |   expect(sum(2, 5)).toBe(5);
      |
      5 | });
      6 |

    at Object.toBe (src/components/__tests__/sum.test.js:4:21)

```

It says I expected answer to be "5" but its giving "7" although our function is correct just we deliberately wrote wrong test case.

This way we can write test case for negative numbers and other. We need to modify our function that way.

**While we are writing "npm run test" our app is not running so Are these test running on browser??**

NO, Its not rendering in our browser. They are running inside a Environment of Jest that is why we needed to configure babel so that Jest understands what we are writing in our code.

```
// The test environment that will be used for testing
testEnvironment: "jsdom",
```

We also see, coverage report in our Terminal as we have set it to true while initialising Jest

```

PASS src/components/__tests__/sum.test.js
  ✓ Check sum of 2 positive numbers (1 ms)

-----|-----|-----|-----|-----|-----|
File   | % Stmt | % Branch | % Funcs | % Lines | Uncovered Line #
-----|-----|-----|-----|-----|-----|
All files | 100 | 100 | 100 | 100 |
sum.js | 100 | 100 | 100 | 100 |
-----|-----|-----|-----|-----|-----|
Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 0.563 s, estimated 1 s
Ran all test suites.

```

Coverage report tells us how much code has been covered for testing in our code.

As we write more and more test cases, this coverage report will change.

We need to put coverage inside .gitignore as we only need this in our VS CODE only

```
.gitignore
42
43  # Large media files
44  *.mp4
45  *.tiff
46  *.avi
47  *.flv
48  *.mov
49  *.wmv
50
51
52
53  .parcel-cache
54
55  /coverage
```

Let us now move to unit testing but first let us run our code side by side.

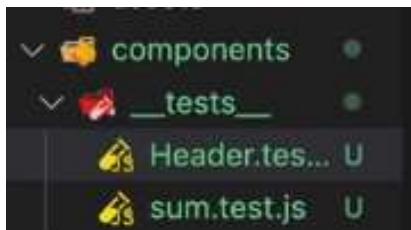
Unit testing means testing individual small portions or components of our app.

### Let us write test cases for header.

We can we expect when we load the header?

When we load the header, Our logo should be there, our cart should be 0 items, our default state should be "online"

We create Header.test.js inside \_\_tests\_\_ folder



Our test case name should be very descriptive

Now what we do inside function

#### 1. Load the Header only

Import the header component.

Code is running inside jsdom and its very small so let us only render Header component, so we use {render} which comes from react-testing-library

```
import { render } from "@testing-library/react";
import Header from "../Header";

test("Logo should load on rendering header", () => {
  // Load Header
  render();
  // Check if logo is loaded
});
```

```
const header = render(<Header/>)
```

We know react does everything inside root element.

Inside jsdom we do not have a root but testing library gives us a small container called render inside which we can load our header component.

We get an **error our code does not understand JSX**

```
For information about custom transformations, see:  
https://jestjs.io/docs/code-transformation  
  
Details:  
  
SyntaxError: /Users/akshaysaini/Desktop/namaste-react-live/src/components/_tests_/Header.test.js: Support for the experimental syntax 'jsx' isn't currently enabled (6:25):  
  
4 | test("Logo should load on rendering header", () => {  
5 |   // Load Header  
> 6 |   const header = render(<Header />);  
|     ^  
7 |  
8 |   console.log(header);  
9 |  
  
  Add @babel/preset-react (https://github.com/babel/babel/tree/main/packages/babel-preset-react) to the 'presets' section of your Babel config to enable transformation.  
  If you want to leave it as-is, add @babel/plugin-syntax-jsx (https://github.com/babel/babel/tree/main/packages/babel-plugin-syntax-jsx) to the 'plugins' section to enable parsing
```

Babel will help us now. Earlier we use babel/preset-env to run JS / ES6 stuff

Now, We will need another Babel package babel/preset-react to make Jest able to understand JSX.

```
npm i -D @babel/preset-react
```

```
{  
  "presets": [  
    ["@babel/preset-env", { "targets": { "node": "current" } }],  
    ["@babel/preset-react", { "runtime": "automatic"}]  
  ]  
}
```

This is just a configuration, which we need to set once,

Again we run "npm run test"

Again error comes as it is trying to run image as JS whereas image in PNG

```
Details:  
  
SyntaxError: /Users/akshaysaini/Desktop/namaste-react-live/src/assets/img/foodvilla.png: Unexpected character '\u202a'. (1:0)  
  
> 1 | \u00d0PNG  
| ^  
2 |  
3 |  
4 | IHDR\u00d07~\u00d0?\u00d0IDATx\u00d0w\u00d0T\u00d0E\u00d0  
  
1 | import { useState, useContext } from "react";  
> 2 | import Logo from "../assets/img/foodvilla.png";  
| ^  
3 | import { Link } from "react-router-dom";  
4 | import useOnline from "../utils/useOnline";  
5 | import UserContext from "../utils/UserContext";  
  
at instantiate (node_modules/@babel/parser/src/parse-error/credentials.ts:1:0)  
at instantiate (node_modules/@babel/parser/src/parse-error.ts:60:12)
```

## How to solve it??

We will create a dummy image (Mock)  
We will create folder inside src named as "mocks"  
And we create a file as dummyLogo.js



**How will our code get to know, we are using dummy images??**

We will add something inside Jest configuration.

## Inside jest.config.js

In `moduleNameMapper` we will tell Jest to replace all PNG image with dummy images inside `dummy.js`

```
moduleNameMapper: {
  "\\\\.png": "../mocks/dummyLogo.js",
},
```

If we want to put JPG also, we do something like

```
moduleNameMapper: [
```

Now we run command "npm run test"

It again failed

It says we have used redux but it does not have provider, as we are only rendering Header component so this error comes

```

● Logo should load on rendering header

  could not find react-redux context value; please ensure the component is wrapped in a <
Provider>

  22 |   const { user } = useContext(UserContext);
  23 |
  > 24 |   const cartItems = useSelector((store) => store.cart.items);
      |
  25 |   console.log(cartItems);
  26 |
  27 |   return (

```

### What to do??

We will add a Provider to our Header and provide store to it just like we did earlier.

```

import { render } from "@testing-library/react";
import Header from "../Header";
import { Provider } from "react-redux";
import store from "../../../utils/store";

test("Logo should load on rendering header", () => {
  // Load Header
  const header = render(
    <Provider store={store}>
      <Header />
    </Provider>
  );

  console.log(header);

  // Check if logo is loaded
});

```

Again run command,  
Failed once again

```

● Logo should load on rendering header

useHref() may be used only in the context of a <Router> component.

  6 | test("Logo should load on rendering header", () => {
  7 |   // Load Header
  > 8 |   const header = render(
      |
  9 |     <Provider store={store}>
 10 |       <Header />
 11 |     </Provider>

```

It gives this error because we have used Link in Header component from react-router-dom  
So we have to give router to it also.  
In normal app we used createBrowserRouter but jsdom is not a browser so we use something called as **StaticRouter** which is given to us by "react-router-dom/server"

This router can work without browser,  
We provide staticRouter to our app

```
import { render } from "@testing-library/react";
import Header from "../Header";
import { Provider } from "react-redux";
import store from "../../utils/store";
import { StaticRouter } from "react-router-dom/server";

test("Logo should load on rendering header", () => {
  // Load Header
  const header = render(
    <StaticRouter>
      <Provider store={store}>
        <Header />
      </Provider>
    </StaticRouter>
  );
  console.log(header);
}
```

Now again we run "npm run test"  
And it passes now.

```
mocks | 0 | 0 | 0 | 0 |
dummyLogo.js | 0 | 0 | 0 | 0 |
utils | 76.19 | 100 | 37.5 | 76.19 |
UserContext.js | 100 | 100 | 100 | 100 |
cartSlice.js | 40 | 100 | 0 | 40 | 10-16 |
store.js | 100 | 100 | 100 | 100 |
useOnline.js | 84.61 | 100 | 60 | 84.61 | 8,11 |

Test Suites: 2 passed, 2 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 0.84 s, estimated 1 s
```

We have rendered Header component successfully.  
Now we write what we are expecting

How to get our logo now, In DOM we do document.getElementById  
Samway we have getAllById which gives us Array of all the element having ID "logo"  
Id is a test id which we did not write earlier in our img tag so now we write it.

```
const Title = () => [
  <a href="/">
    <img data-testid="logo" className="h-28 p-2" alt="logo" src={Logo} />
  </a>
];

const logo = header.getAllById("logo");
```

Now we console.log(logo) to see it out and we see

We get HTML element of type 'img'

```
console.log
[
  <ref *1> HTMLImageElement {
    '_reactFiber$89a2d2razrb': FiberNode {
      tag: 5,
      key: null,
      elementType: 'img',
      type: 'img',
      stateNode: [Circular *1],
      return: [FiberNode],
      child: null,
      sibling: null,
      index: 0,
      ref: null,
      pendingProps: [Object],
      memoizedProps: [Object],
      updateQueue: null,
      memoizedState: null,
      dependencies: null,
```

Now we expect something, we write logo[0] because logo is an Array as getAllById gives us Array, logo[0] gives us our logo image

```
console.log(logo[0]);
expect(logo[0].src).toBe("dummyLogo.png");
```

It failed, because we are not writing full URL

```
● Logo should load on rendering header

expect(received).toBe(expected) // Object.is equality

Expected: "dummyLogo.png"
Received: "http://localhost/dummy.png"

  19 |   console.log(logo[0]);
  20 |
> 21 |   expect(logo[0].src).toBe("dummyLogo.png");
|     ^
```

Testing is a time-taking job and we can encounter so many errors  
To resolve above error, we do

```
expect(logo[0].src).toBe("http://localhost/dummy.png");
```

Now its passed

dummyLogo.js	0	0	0	0
utils	76.19	100	37.5	76.19
UserContext.js	100	100	100	100
cartSlice.js	48	100	0	48
store.js	100	100	100	100
useOnline.js	84.61	100	60	84.61
	--	--	--	--
Test Suites:	2 passed	2 total		
Tests:	2 passed	2 total		
Snapshots:	0 total			
Time:	0.88 s	estimated 1 s		

## 2. Let us write test to check for "Online"

Now Jest will automatically test our useOnline custom hook also as if it's was wrong we would have got wrong results.

Add a test ID to our element inside Header.js

```
<h1 data-testid="online-status">{isOnline ? "✅" : "🔴"}</h1>
```

Let us write a new test case for it.

```
test("Online Status should be green on rendering header", () =>
  // Load Header
  const header = render(
    <StaticRouter>
      <Provider store={store}>
        <Header />
      </Provider>
    </StaticRouter>
  );

  // Check if logo is loaded
  const onlineStatus = header.getAllByTestId("online-status");

  expect(onlineStatus.innerHTML).toBe("✅");
);
```

Instead of taking first element of Array everytime like onlineStatus[0] we can also use getByTestId instead of getAllByTestId and now we do not have any Array we get just one element which matches the ID.

```
// Check if logo is loaded
const onlineStatus = header.getByTestId("online-status");

expect(onlineStatus.innerHTML).toBe("✅");
```

## 3. Let us write one more test case for cart should have 0 items

Jest will test useSelector also automatically as Number of cart items comes from useSelector

Give test ID to cart

```
<Link to="/cart">
  <li className="px-2" data-testid="cart">
    Cart- {cartItems.length} items
  </li>
</Link>
```

```
test("Cart should have 0 items on rendering header", () => {
  // Load Header
  const header = render(
    <StaticRouter>
      <Provider store={store}>
        <Header />
      </Provider>
    </StaticRouter>
  );
  // Check if logo is loaded
  const cart = header.getByTestId("cart");
  expect(cart.innerHTML).toBe("0");
});
```

Error comes our Jest tells us

```
● Cart should have 0 items on rendering header

expect(received).toBe(expected) // Object.is equality

Expected: "0"
Received: "Cart- 0 items"
```

So instead of toBe("0") we use toBe("Cart- 0 items") now

```
// Check if logo is loaded
const cart = header.getByTestId("cart");

expect(cart.innerHTML).toBe("Cart- 0 items");
```

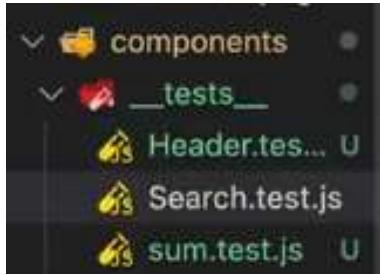
## Let us Take complex test cases now

Till now we did "unit Testing" we will now do "Integration testing"

Let us do testing in Search component

Our search component is connected to so many components so it automatically tests for all those components also.

We make Search.test.js inside \_\_tests\_\_



Let us write our test case

```
test("Search Results on Homepage", () => {  
})
```

We will render our app's Body

We are using react-redux so we will give Provider to our Body and provide store to it,  
We will import our StaticRouter also from react-router-dom/server

```
import { render } from "@testing-library/react";  
import Body from "../Body";  
import { Provider } from "react-redux";  
import store from "../utils/store.js";  
import { StaticRouter } from "react-router-dom/server";  
  
test("Search Results on Homepage", () => {  
  const body = render(  
    <StaticRouter>  
      <Provider store={store}>  
        <Body />  
      </Provider>  
    </StaticRouter>  
  );  
  console.log(body);  
});
```

Let us write "npm run test" we see

```
ReferenceError: fetch is not defined  
  at getRestaurants (/Users/akshaysaini/Desktop/namaste-react-live/src/co  
:624:31)  
  at /Users/akshaysaini/Desktop/namaste-react-live/src/components/Body.js  
  at commitHookEffectListMount (/Users/akshaysaini/Desktop/namaste-react-  
s/react-dom/cjs/react-dom.development.js:23150:26)  
  at commitPassiveMountOnFiber (/Users/akshaysaini/Desktop/namaste-react-  
s/react-dom/cjs/react-dom.development.js:24931:11)  
  at commitPassiveMountEffects_complete (/Users/akshaysaini/Desktop/namas  
de_modules/react-dom/cjs/react-dom.development.js:24891:9)  
  at commitPassiveMountEffects_begin (/Users/akshaysaini/Desktop/namaste-  
modules/react-dom/cjs/react-dom.development.js:24878:7)  
  at commitPassiveMountEffects (/Users/akshaysaini/Desktop/namaste-react-  
s/react-dom/cjs/react-dom.development.js:24866:3)  
  at flushPassiveEffectsImpl (/Users/akshaysaini/Desktop/namaste-react-l
```

Jest does not know fetch as fetch is a browser thing so we need to Mock our fetch also

We write something known as global.fetch and we use a dummy function given to us by Jest

### What will this function do?

We write whatever fetch does in general, inside this function

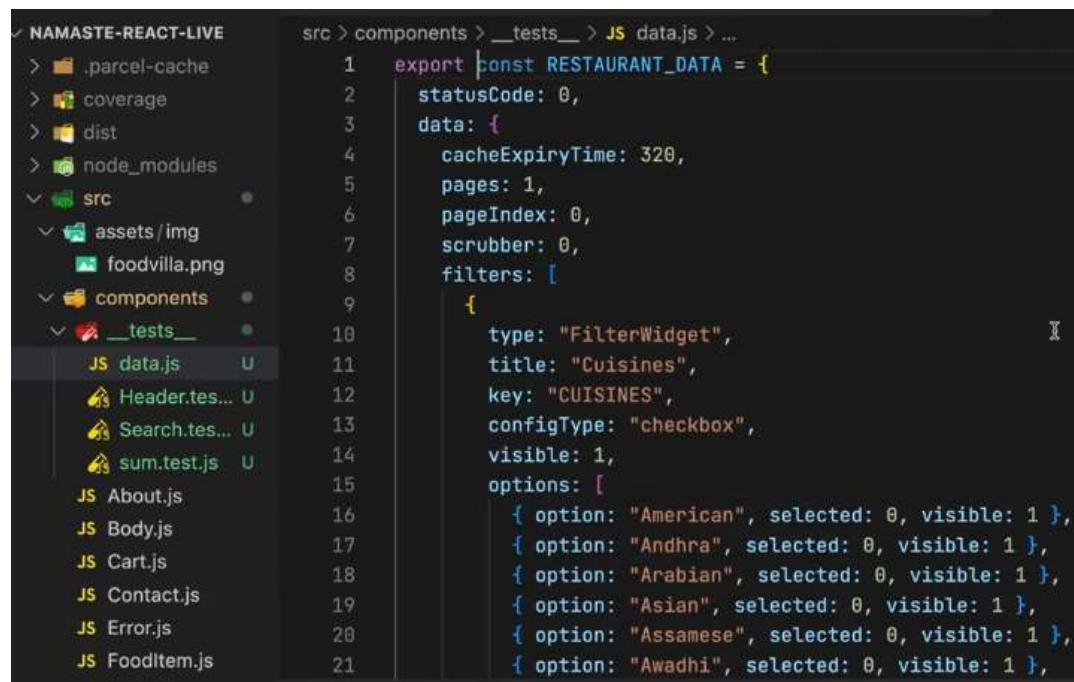
Fetch returns a promise so we resolve a Promise inside the function.

Fetch returns us a readable stream and we convert that stream to JSON

```
global.fetch = jest.fn(()=>{
  Promise.resolve({
    json: Promise.resolve()
  });
});
```

data.json() again gives us Promise so we resolve this Promise and we pass data which we need to mock inside it.

Let us take that data from browser API response and put it inside a new file data.js inside \_\_tests\_\_



The screenshot shows a code editor with two panes. On the left is a file tree for a project named 'NAMASTE-REACT-LIVE'. The tree includes directories like '.parcel-cache', 'coverage', 'dist', 'node\_modules', and 'src'. Under 'src', there are 'assets/img' (containing 'foodvilla.png'), 'components' (containing 'About.js', 'Body.js', 'Cart.js', 'Contact.js', 'Error.js', 'FoodItem.js', 'Header.test.js', 'Search.test.js', and 'sum.test.js'), and '\_\_tests\_\_' (containing 'data.js'). On the right is a code editor pane displaying a JavaScript file 'data.js'. The code defines a constant RESTAURANT\_DATA with properties like statusCode, data (which contains cacheExpiryTime, pages, pageIndex, scrubber, and filters), and filters (which contains type, title, key, configType, visible, and options). The code is numbered from 1 to 21.

```
✓ NAMASTE-REACT-LIVE
  src > components > __tests__ > JS data.js > ...
    1  export const RESTAURANT_DATA = {
    2    statusCode: 0,
    3    data: {
    4      cacheExpiryTime: 320,
    5      pages: 1,
    6      pageIndex: 0,
    7      scrubber: 0,
    8      filters: [
    9        {
    10          type: "FilterWidget",
    11          title: "Cuisines",
    12          key: "CUISINES",
    13          configType: "checkbox",
    14          visible: 1,
    15          options: [
    16            { option: "American", selected: 0, visible: 1 },
    17            { option: "Andhra", selected: 0, visible: 1 },
    18            { option: "Arabian", selected: 0, visible: 1 },
    19            { option: "Asian", selected: 0, visible: 1 },
    20            { option: "Assamese", selected: 0, visible: 1 },
    21            { option: "Awadhi", selected: 0, visible: 1 },
    22          ]
    23        }
    24      ]
    25    }
    26  }
```

We need to Mock this data as we are faking this API call for testing

Let us now use Restro data inside our Promise

```
import { render } from "@testing-library/react";
import Body from "../Body";
import { Provider } from "react-redux";
import store from "../../utils/store.js";
import { StaticRouter } from "react-router-dom/server";
import { RESTAURANT_DATA } from "./data";

global.fetch = jest.fn(()=>{
  Promise.resolve({
    json: Promise.resolve(RESTAURANT_DATA)
  })
});
```

We write fetch using global.fetch so that it is accessible to our jest.

In above **global.fetch** we are using 2 promises because while using fetch API in JS we use await 2 times means we have 2 Promise returned

Await fetch gives one promise

Await data.json() gives us another promise

And In our above code, we write result for Promise.resolve

```
async function getRestaurants() {
  const data = await fetch(
    "https://www.swiggy.com/dapi/restaurants/list/v5?lat=12.9351929&lon=77.616071"
  );
  const json = await data.json();
  setAllRestaurants(json?.data?.cards[2]?.data?.data?.cards);
  setFilteredRestaurants(json?.data?.cards[2]?.data?.data?.data?.cards);
}
if (!allRestaurants) return null;
```

This is how our Search.test.js looks like

```

components/__tests__/Search.test.js
import { render } from "@testing-library/react";
import Body from "../Body";
import { Provider } from "react-redux";
import store from "../../utils/store.js";
import { StaticRouter } from "react-router-dom/server";
import { RESTAURANT_DATA } from "./data";

global.fetch = jest.fn(() => {
  return Promise.resolve({
    json: Promise.resolve(RESTAURANT_DATA),
  });
});

test("Search Results on Homepage", () => {
  const body = render(
    <StaticRouter>
      <Provider store={store}>
        <Body />
      </Provider>
    </StaticRouter>
  );
  console.log(body);
});

```

Now, we get another error

```

RUNS src/components/__tests__/Search.test.js
/Users/akshaysaini/Desktop/namaste-react-live/src/components/Body.js:627
  (cov_108nemfgl8().s[8]++, await data.json());
                                         ^

TypeError: data.json is not a function
  at getRestaurants (/Users/akshaysaini/Desktop/namaste-react-live/src/comp
:627:42)

```

We need to make our json as a function inside global.fetch

```

global.fetch = jest.fn(() => {
  return Promise.resolve({
    json: () => {
      return Promise.resolve(RESTAURANT_DATA);
    },
  });
});

```

It again fails

```
24 |     const json = await data.json();
25 |     setAllRestaurants(json?.data?.cards[0]?.data?.data?.cards);
> 26 |     setFilteredRestaurants(json?.data?.cards[2]?.data?.data?.cards);
|     ^
27 |   }
28 |   if (!allRestaurants) return null;
29 | 
```

at printWarning (node\_modules/react-dom/cjs/react-dom.development.js:86:36)  
at error (node\_modules/react-dom/cjs/react-dom.development.js:60:7)  
at warnIfUpdatesNotWrappedWithActDEV (node\_modules/react-dom/cjs/react-dom.development.js:60:1)

Let us first expect somethinng. Let us find our search button inside body

We give testID to search button inside body.js

```
/* 
<button
  data-testid="search-btn"
  className="p-2 m-2 bg-purple-900 hover:bg-gray-500 text-white"
  onClick={() => {
    //need to filter the data
    const data = filterData(searchText, allRestaurants);
    // update the state - restaurants
    setFilteredRestaurants(data);
  }}
>
  Search
</button>
```

```
const searchBtn = body.getByTestId("search-btn");

console.log(searchBtn);
```

It gives error as it considers data.js also as a test file so we should paste data.js inside Mocks only.

```
PASS src/components/__tests__/sum.test.js
FAIL src/components/__tests__/data.js
● Test suite failed to run

Your test suite must contain at least one test.
```

Again it gives error because it is first loading Shimmer in our body.js so Let us create a test case for shimmer also.

```
● Search Results on Homepage

TestingLibraryElementError: Unable to find an element by: [data-testid="search-btn"]

Ignored nodes: comments, script, style
<body>
  <div>
```

```
test("Shimmer should load on Homepage",
```

Let us find our Shimmer first

We give test-id to shimmer

```
src > components > JS Shimmer.js > [e] Shimmer
...
1 const Shimmer = () => {
2   return (
3     <div className="restaurant-list" data-testid="shimmer">
4       {Array(10)
5         .fill("")
6         .map((e, index) => (
7           <div key={index} className="shimmer-card"></div>
8         )))
9     </div>
10   );
11 };
12
13 export default Shimmer;
14
```

Let us test Shimmer

```
test("Shimmer should load on Homepage", () => {
  const body = render(
    <StaticRouter>
      <Provider store={store}>
        <Body />
      </Provider>
    </StaticRouter>
  );

  const shimmer = body.getByTestId("shimmer");

  console.log(shimmer);
});
```

We see we have to write "npm run test" everytime so What if we could have something like hot module replacement for Jest also??

We go to Package.json and we have a command in jest for that

```
"scripts": [
  "start": "parcel index.html",
  "build": "parcel build index.html",
  "test": "jest",
  "watch-test": "jest --watch"
],
```

```
npm run watch-test
```

Now it will keep running itself

Let us test Shimmer now

There is something like `toBeInTheDocument` to check whether that thing is in the document or not.

`toBeInTheDocument` comes from

```
import "@testing-library/jest-dom";
```

So Let us install the library

```
const shimmer = body.getByTestId("shimmer");

expect(shimmer).toBeInTheDocument();

console.log(shimmer);
```

But `toBeInTheDocument` is not a good way, Let us write it in another way.

We can use `shimmer.children` then it will give us all children of shimmer component

```
expect(shimmer.children).toBeInTheDocument();
```

```
received value must be an HTMLElement or an SVGElement.
Received has type: object
Received has value: [<div class="shimmer-card" />, <div class="shimmer-card" />]
```

We can do soemthing like

```
expect(shimmer.children.length).toBe(10);
```

Let us now test for "**Restro should load on homepage**"

For that we need to **ignore our shimmer, how to do it??**

We need to wait till our data loads so we use "await" given by react testing library

We will wait till we have search-button

We make the function async

```

test("Restaurants should load on Homepage", async() => {
  const body = render(
    <StaticRouter>
      <Provider store={store}>
        <Body />
      </Provider>
    </StaticRouter>
  );

  await waitFor(() => expect(screen.getByTestId("search-btn")));

  const shimmer = body.getByTestId("shimmer");

  expect(shimmer.children.length).toBe(10);

  console.log(shimmer);
});

```

Now we test for resto-list

We give testID to resto list

```

<div>
  <div className="flex flex-wrap " data-testid="res-list">
    /* You have to write logic for NO restraint fount here
    {filteredRestaurants.map((restaurant) => {
      return (
        <Link
          to={"/restaurant/" + restaurant.data.id}
          key={restaurant.data.id}
        >
          <RestaurantCard {...restaurant.data} />
        </Link>
      );
    })}
  </div>

```

Now we search for resList

```

await waitFor(() => expect(body.getByTestId("search-btn")));

const resList = body.getByTestId("res-list");

expect(resList.children.length).toBe(15);

```

Let us find input box in body and type something into it and click Search and see what is happening

We give testID to our input

```
<input  
  data-testid="search-input" placeholder=""  
  type="text" placeholder="Search"  
  value={searchText}  
  onChange={(e) => {  
    setSearchText(e.target.value);  
  }}>
```

```
test("Search for string(food) on Homepage", async () => {  
  const body = render(  
    <StaticRouter>  
      <Provider store={store}>  
        <Body />  
      </Provider>  
    </StaticRouter>  
  );  
  
  await waitFor(() => expect(body.getByTestId("search-btn")));  
  
  const input = body.getByTestId("search-input");
```

We need to type something inside input so we Mock it by firing an event

There is something known as fireEvent given by testing library

```
import { render, waitFor, fireEvent } from "@testing-library/react";
```

We need to fire event for onChange of input box so

Inside onChange we do something like e.target.value so here we give hardcoded value = "food" to our input field

```
fireEvent.change(input, {  
  target: {  
    value: "food",  
  },  
});
```

Now we fireEvent a click on Search Button

```

await waitFor(() => expect(body.getByTestId("search-btn")));

const input = body.getByTestId("search-input");

fireEvent.change(input, {
  target: {
    value: "food",
  },
});

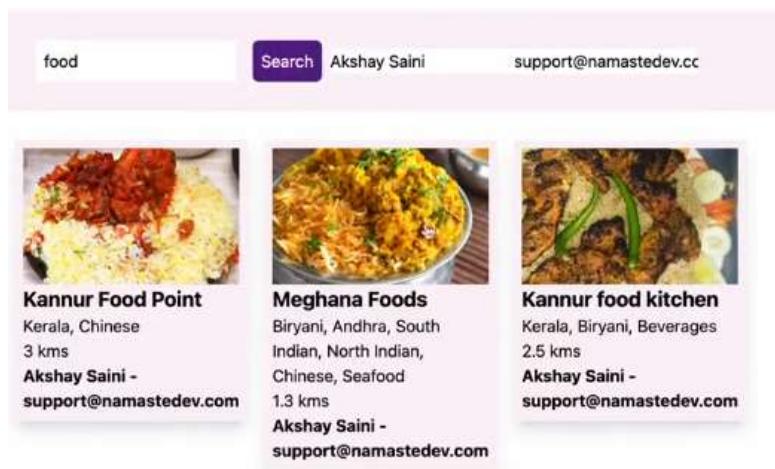
const searchBtn = body.getByTestId("search-btn");

fireEvent.click(searchBtn);

console.log(shimmer);
});

```

Now we should expect our Search to give us list of 4 resto for input = "food"



Let us see if our test passes or not

```

fireEvent.click(searchBtn);

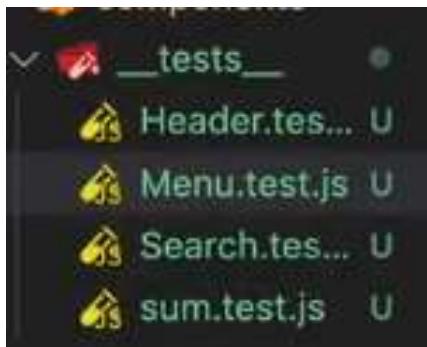
const resList = body.getByTestId("res-list");

expect(resList.children.length).toBe(4);

```

Let us **test for "We click "Add" in Menu and our Cart gets updated"**

We make Menu.test.js



We need different API data for it so Mock data  
Rest code for fetch etc is same as previous

```
global.fetch = jest.fn(() => {
  return Promise.resolve({
    json: () => {
      return Promise.resolve(MENU_DATA);
    },
  });
});

test("Add Items to Cart", async () => {
  const body = render(
    <StaticRouter>
      <Provider store={store}>
        <Restaurant>menu| />
      </Provider>
    </StaticRouter>
  );
});
```

Give testID to menu

```
<ul data-testid="menu">
```

A screenshot of a code editor showing a portion of a React component. It contains an 

 element with the attribute `data-testid="menu"`. Inside the 

 element, there is a `{Object.values(rest abc menu}` block, followed by several   - elements. Each   - element has a key prop set to `item.id`, a name prop set to `{item.name}`, and a button with a `onClick={() = abc Menus` prop.

Let us import Header also to get access of Cart

We give testID to ADD button

```

  {item.name} + " "
  <button
    data-id="addBtn" data-
    className="p-1 bg-green-50"
    onClick={() => addFoodItem(item)}
  >
    Add
  </button>

```

We do addBtn[0] to get ADD of first menu

```

await waitFor(() => expect(body.getByTestId("menu")));

const addBtn = body.getAllByTestId("addBtn");

fireEvent.click(addBtn[0]);

const cart = body.getByTestId("cart");

expect(cart.innerHTML).toBe(["Cart - 1 item"]);

```

If we fire event for one more ADD, we should get 2 items in Cart

```

await waitFor(() => expect(body.getByTestId("menu")));

const addBtn = body.getAllByTestId("addBtn");

fireEvent.click(addBtn[0]);
fireEvent.click(addBtn[2]);

const cart = body.getByTestId("cart");

expect(cart.innerHTML).toBe(["Cart- 2 items"]);
});

```

We can also render our Cart Component and see what coming inside Cart component

## Session 16

### Machine coding round

Some companies take machine coding in HTML, CSS, JS only

Some companies take machine coding round in Tech stack they are hiring you for.

Round consists of building an app using react

You might be asked to make Fetch API, infinite scroll, Todo list, Tic Tac Toe, Forms, E-commerce website, Small Swiggy website

## Toughest part of Machine coding interview

Managing your time, Live coding in front of interviewer, We have to build an app with a timer running continuously.

We are given 1-2 hours and we have build whole app in 1-2 hours and interviewew is watching us.

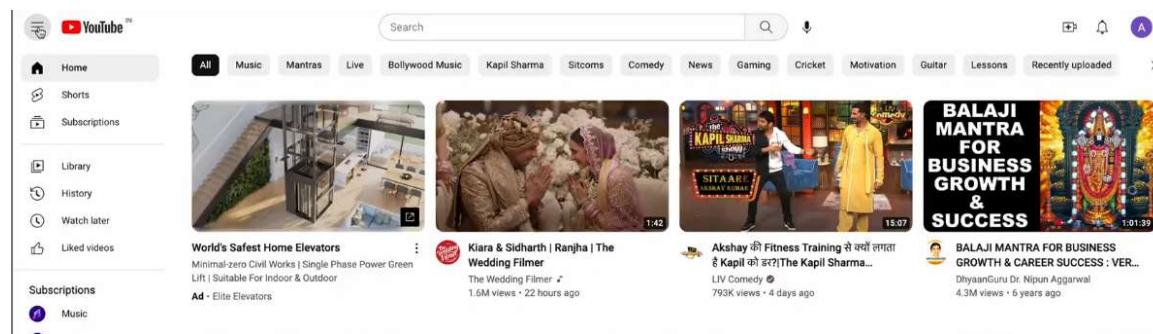
We can avoid this pressure by praticing before interview, In the interview Planning will help us.

## We will Mock a machine coding round of interview

We will **build Youtube**

### How will we start??

Do not start your IDE straight away, Ask your interviewer about the requirements  
What features to develop, we cannot build whole youtube in 2 hours.



We have search bar, comment section, buttons, Hamburger menu, video cards etc etc.  
Discuss what to build with your interviewer.

**Tell tech stack to interviewer.** I would use tailwind with React or use redux or context API  
If our app is small we can use context API, if application is big we can use redux.  
We can use Tailwind for CSS

Now tell him why you are using context or redux or context or MUI. **Give justification about every tech stack you are choosing.**

Suppose our app uses forms, we can tell we are using formik for forms and react router dom for routing, we can discuss about bundlers we are choosing, what testing library we are using etc etc.

### How much time to spend on this??

5 minutes only

### Can we do goggle search or not?

Yes, we can but we loose time in doing so

## Planning

Spend next 5 minutes in planning now, don't start writing code now

Plan your LLD, Structure your component, UI and data layer, make an blueprint

We will discuss our approach with the interviewer, There can be 1000 ways of making

Youtube, we need to tell him our approach

We will have an Header

We have an Ham-burger which has home, subscription etc etc

Ham-burger collapses and opens on click

We have logo

We have search bar

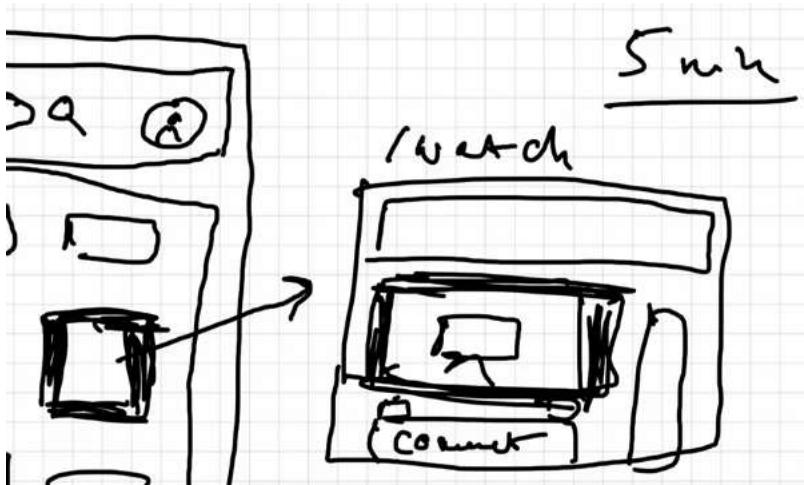
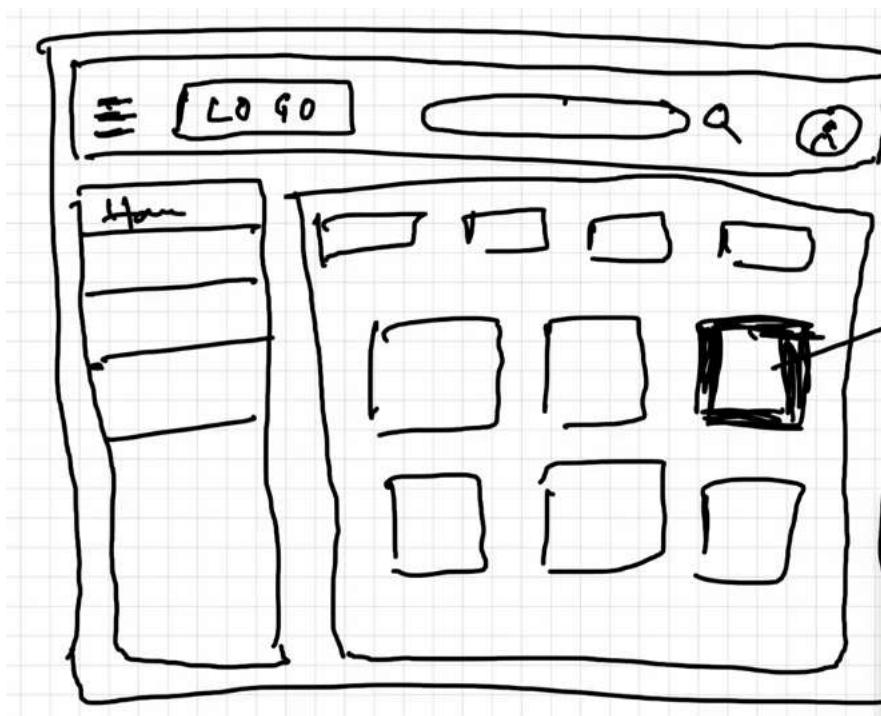
We have user icon

We have body which has button list or filters

We have videos

When we click on a video, it opens up a new page which is /watch URL

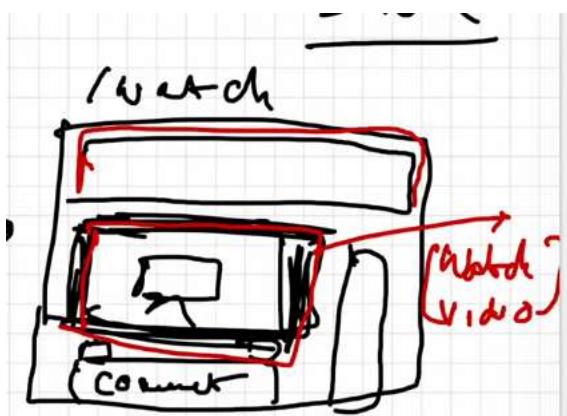
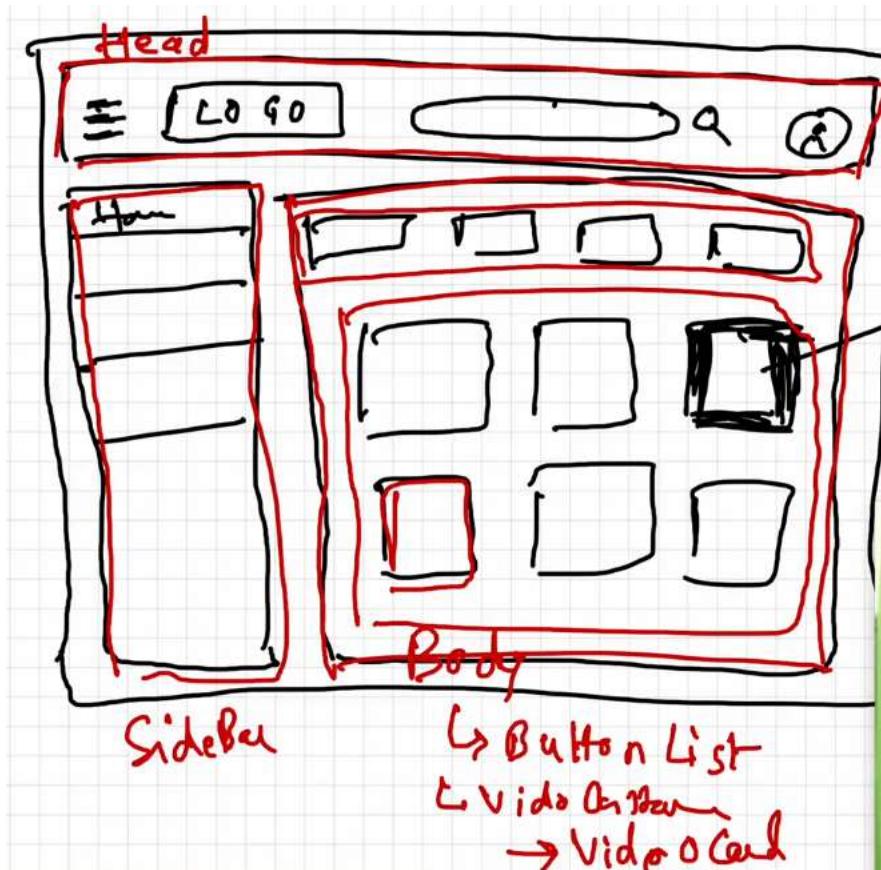
/watch will have same Header, video playing, comment section, recommendation section also, like subscribe etc button.



Discuss this for 5 minutes

More planning will we do, better code we write

We can go one level deep in planning and tell them about how will we distinguish in our component with name



## This planning is very very important

If we discussing react testing library, does not mean we have to write test cases. It just tells the interviewer that he knows about testing also.

Tell him for routes we will use react router dom.

## Now Let us just code things up

To start a new project, either we can start from scratch making parcel bundler etc but we have a time crunch so we can discuss to interviewer about bundler, parcel etc etc but To save time we need to use create-react-app

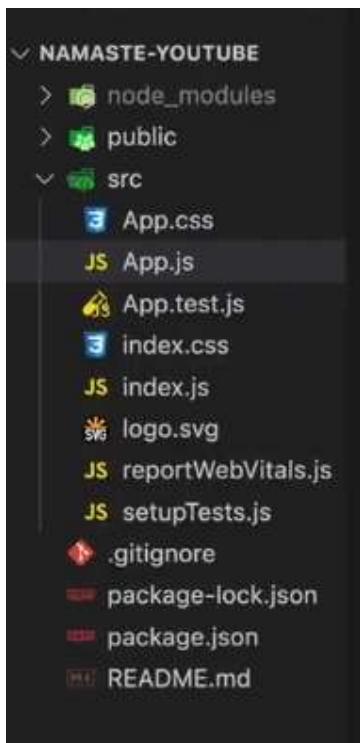
```
npx create-react-app youtube
```

create-react-app is a package which we are executing using npx  
We want to execute it only once so we use npx

Earlier, We ourself setup everything but when we do create-react-app, it does everything pre-written for us

We now do "npm run start" to start our react app.

Let us take a deep dive into create-react-app



What is `reportWebVitals` in React used for?

`reportWebVitals` are a set of useful metrics that aim to **capture the user experience of a web page**

`setupTests.js` is to set up test cases

Now we just focus on `app.js` and we write all our code

Setup Tailwind first

```
npm i - D tailwindcss
```

Now we build tailwind.config

```
npx tailwindcss init
```

Now we tell our Tailwind what files to track

```
tailwind.config.js > [e] <unknown> > ⚡ content
1  /** @type {import('tailwindcss').Config} */
2  module.exports = {
3    content: ['./src/**/*.{js,jsx,ts,tsx}'],
4    theme: {
5      extend: {},
6    },
7    plugins: [],
8  };
9
```

Inside App.css we write

```
src > App.css
...
1  @tailwind base;
2  @tailwind components;
3  @tailwind utilities;
```

App.js looks like

```
src > App.js > ...
You, 1 second ago | 1 author (You)
1 import "./App.css";
2
3 function App() {
4   return (
5     <div>
6       <h1 className="bg-red">Namaste React</h1>
7     </div>
8   );
9 }
10
11 export default App;
12 |
```

Our app structure looks like

```
1 /**
2  * Head
3  * Body
4  * Sidebar
5  *   MenuItems
6  * MainContainer
7  *   ButtonsList
8  *   VideoContainer
9  *     VideoCard
10 */
11 */
```

Let us make our Header component  
We create src/components/Head.js

Load Head component inside App.js

We create a body component and load it in App.js

Inside Body we need sidebar component

Inside Body we have MainContainer which contains video cards

We should have sidebar in left and maincontainer in right

Maincontainer should have button list

After buttonList we should have videoContainer in mainContainer

```
Head
SidebarButtonList
  VideoContainer
```



## Let us build our Head.js



## First section of header

Let us use Hamburger image

Youtube logo

## Second section of header

## Search box with button Search

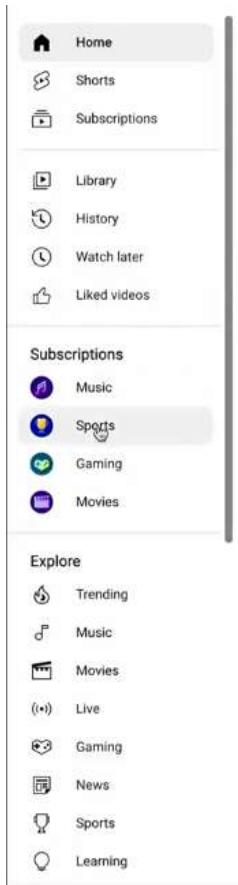
### Third section of header

## User icon

```
import React from "react";

const Head = () => {
  return [
    <div>
      <div>
        
      </div>
      <div>
        <input type="text" />
        <button>Search</button>
      </div>
      <div>
        
  <input
    className="w-1/2 border border-gray-400 p-2 rounded-l-full"
    type="text"
  />
  <button className="border border-gray-400 px-5 py-2 rounded-r-full bg-gray-100">
    | 
    </button>
</div>
```

Let us build our sidebar now



We have different sections in our sidebar so we can give list in sidebar

```
src > components > JS Sidebar.js > [e] Sidebar
1 import React from "react";
2
3 const Sidebar = () => {
4   return (
5     <div className="p-5 shadow-lg w-48">
6       <ul>
7         <li> Home</li>
8         <li> Shorts</li>
9         <li> Videos</li>
10        <li> Live</li>
11      </ul>
12      <h1 className="font-bold">Subscriptions</h1>
13      <ul>
14        <li> Music</li>
15        <li> Sports</li>
16        <li> Gaming</li>
17        <li> Movies</li>
18      </ul>
19      <h1 className="font-bold pt-5">Watch Later</h1>
20      <ul>
21        <li> Music</li>
22        <li> Sports</li>
23        <li> Gaming</li>
24        <li> Movies</li>
25      </ul>
```

We can also add icons if we wish to

## Should we spend this much time on CSS in machine coding round??

NO, Spend more time on functionality that if we click hamburger our sidebar collapses etc etc. Don't spend much time on CSS

Let us focus on hamburger collapse and open thing.

We can make an state for this, but this functionality can be used anywhere in such a big app like Youtube so let us store this in global space. We will use redux store

Install react redux and redux toolkit

We will make utils folder and make store.js inside utils



```
src > utils > JS store.js > ...
1 import { configureStore } from "@reduxjs/toolkit";
2
3 const store = configureStore({});
4
5 export default store;
```

Make a slice appSlice.js

We use createSlice API

```
src > utils > JS appSlice.js > ...
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const appSlice = createSlice({
4   name: "app",
5   initialState: {
6     isMenuOpen: true,
7   },
8   reducers: {
9     toggleMenu: (state) => {
10       state.isMenuOpen = !state.isMenuOpen;
11     },
12   },
13 });
14
15 export const [toggleMenu] = appSlice.actions;
16 export default appSlice.reducer;
17
```

Use appSlice in our store

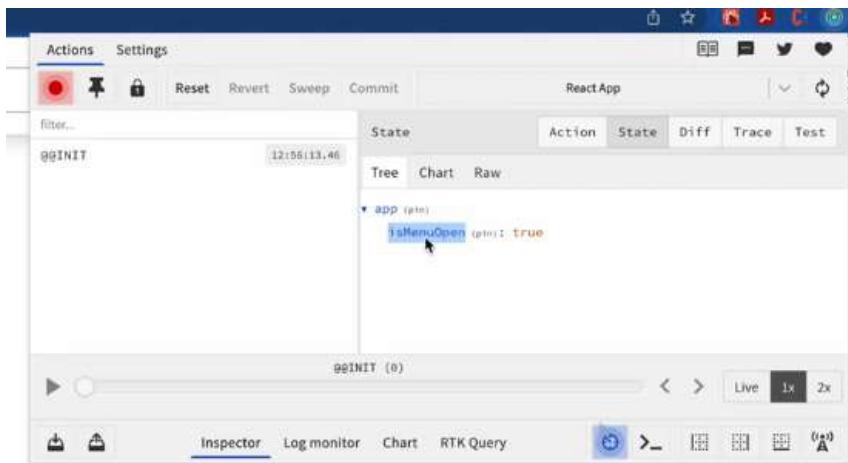
```
src > utils > JS store.js > [o] store > ↴ reducer
1 import { configureStore } from "@reduxjs/toolkit";
2 import appSlice from "./appSlice";
3
4 const store = configureStore({
5   reducer: {
6     app: appSlice,
7   },
8 });
9
10 export default store;
11
```

Use Provider in App.js

```
src > JS App.js > ...
You, 1 second ago | 1 author (You)
1 import { Provider } from "react-redux";
2 import "./App.css";
3 import Body from "./components/Body";
4 import Head from "./components/Head";
5 import store from "./components/store";
6
7 function App() {
8   return (
9     <Provider store={store}>
10       <div>
11         <Head />
12         <Body />
13
14         /**
15          *
16          * Head
17          * Body
18          * Sidebar
19          *   MenuItem
20          *   MainContainer
21          *     ButtonsList
22          *     VideoContainer
23          *     VideoCard
24          *
25          */
26     </div>
27   );
28 }
29
30 export default App;
```

### How to check store is working properly or not??

We can either make use of useSelector us use our extension



Let us dispatch an action in Header component

We do not pass anything to toggleMenu as it does not have any payload in AppSlice.js

```
import React from "react";
import { useDispatch } from "react-redux";
import { toggleMenu } from "../utils/appSlice";

const Head = () => {
  const dispatch = useDispatch();

  const toggleMenuHandler = () => {
    dispatch(toggleMenu());
  };

  return (
    <div className="grid grid-flow-col p-5 m-2 shadow-lg">
      <div className="flex col-span-1">
        <img
          onClick={() => toggleMenuHandler()}
          className="h-8"
          alt="menu"
          src="data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAO
        />
    
```

Now we subscribe to our store and use that information to show or hide our sidebar

So now we use useSelector

We need to subscribe to specific part of our store, we do not need to subscribe to whole store

```
src > components > JS Sidebar.js > Sidebar
1 import React from "react";
2 import { useSelector } from "react-redux";
3
4 const Sidebar = () => [
5   const isMenuOpen = useSelector((store) => store.app.isMenuOpen);
6 ]
```

Now to render sidebar on conditional basis either we can do

**Early return**

```
const isMenuOpen = useSelector((store) => store.app.isMenuOpen);

// Early Return pattern
if (!isMenuOpen) return null;

return (
  <div className="p-5 shadow-lg w-48">
    <ul>
      <li> Home</li>
      <li> Shorts</li>
      <li> Videos</li>
      <li> Live</li>
    </ul>
    <h1 className="font-bold pt-5">Subscriptions</h1>
    <ul>
      <li> Music</li>
      <li> Sports</li>
      <li> Gaming</li>
```

Or we can use ternary operator like

```
return !isMenuOpen ? null : [
  <div className="p-5 shadow-lg w-48">
    <ul>
      <li> Home</li>
      <li> Shorts</li>
      <li> Videos</li>
      <li> Live</li>
    </ul>
    <h1 className="font-bold pt-5">Subscriptions</h1>
    <ul>
      <li> Music</li>
      <li> Sports</li>
      <li> Gaming</li>
      <li> Movies</li>
    </ul>
    <h1 className="font-bold pt-5">Watch Later</h1>
  </div>
```

Let us build our mainContainer now

Let us make button list now

We make a button component in which we have and we have a buttonList component inside which we have list of button components.

```
src > components > JS Button.js > [e] Button
1 import React from "react";
2
3 const Button = () => {
4   return [
5     <div>
6       <button className="px-5 py-2 m-5 bg-gray-200 rounded-lg">All</button>
7     </div>
8   ];
9 }
10
11 export default Button;
12
```

Now we render our buttonList in mainContainer and to change text inside button dynamically we can pass props.

We make a buttonList component which calls button component for us

```
src > components > JS ButtonList.js > [e] ButtonList
1 import React from "react";
2 import Button from "./Button";
3
4 const ButtonList = () => {
5   return [
6     <div className="flex">
7       <Button name="All"/>
8       <Button name="Gaming"/>
9       <Button name="Songs"/>
10      <Button name="Live"/>
11      <Button name="Cricket"/>
12     </div>
13   ];
14 }
15
16 export default ButtonList;
17
```

We can make these button scrollable also,

### Now for videos

**Ask Interviewer, How should we get our data for videos, Hard-coded or call API ??**

We will use **Youtube API**

The screenshot shows the YouTube Data API documentation. On the left, there's a sidebar with a tree view of API resources. Under the 'Videos' category, the 'list' method is selected. The main content area displays examples for three methods: 'list (multiple video IDs)', 'list (most popular videos)', and 'list (my liked videos)'. Each example includes a code snippet and a detailed description.

Method	Description
<code>list (multiple video IDs)</code>	This example retrieves info for a comma-separated list of YouTube video IDs. It also includes additional information about each video.
<code>list (most popular videos)</code>	This example retrieves a list of the most popular videos. It identifies the country for which the video was most popular and returns the most popular video for that country. You can use the <code>regionCode</code> parameter to retrieve the most popular video for a specific region.
<code>list (my liked videos)</code>	This example retrieves a list of the user's liked videos. You can use the <code>rating</code> parameter to filter the results by rating.

**Request**

**HTTP request**

```
GET https://www.googleapis.com/youtube/v3/videos?part=snippet&chart=mostPopular&key={API_KEY}
```

**Parameters**

The following table lists the parameters that this query supports:

Parameters	
Required parameters	
<code>part</code>	<code>string</code> The <code>part</code> parameter specifies the type of data that the API response will include.
<code>chart</code>	<code>string</code> If the <code>chart</code> parameter identifies a particular chart, it is included in the response. For example, if you specify <code>chart=mostPopular</code> , the API response includes the most popular video for each country.
<code>key</code>	<code>string</code> The <code>key</code> parameter identifies the API key used to make the request.

Let us show the most popular videos API in youtube

Let us keep the API URL inside constant.js

```
src > utils > JS contents.js > [e] YOUTUBE_VIDEOS_API
1 const YOUTUBE_VIDEOS_API =
2   "https://youtube.googleapis.com/youtube/v3/videos?part=snippet%2ContentDetails%2Cstatistics"
3
```

It takes your API key also, just search on google and get it.

We use our API keys and export our Youtube API

```
src > utils > JS contants.js > ...
1  const GOOGLE_API_KEY = "AIzaSyCfxXzU7wimn1HnC_vecAniss7Ss0DDyUs";
2
3  const YOUTUBE_VIDEOS_API =
4    "https://youtube.googleapis.com/youtube/v3/videos?part=snippet%2ContentDetails%2Cstatistics&chart"
5    GOOGLE_API_KEY;
6
```

You should not keep your API key like this, we hsould keep it in env file

Inside VideoContainer we make an API call

```
src > components > JS VideoContainer.js > [o] VideoContainer
1  import React, { useEffect } from "react";
2  import { YOUTUBE_VIDEOS_API } from "../utils/contants";
3
4  const VideoContainer = () => {
5    useEffect(() => {
6      getVideos();
7    }, [1]);
8
9    const getVideos = async () => {
10      const data = await fetch(YOUTUBE_VIDEOS_API);
11      const json = await data.json();
12      console.log(json);
13    };
14
15    return <div>VideoContainer</div>;
16  };
17
18  export default VideoContainer;
19
```

Our API gives us this data

```

  ▶ {kind: 'youtube#videoListResponse', etag: 'G4bLe0jlAulsQZifz8SXgbUrbyc', items: Array(5), nextPageToken: 'CAUQAA', pageInfo: {...}
    etag: "G4bLe0jlAulsQZifz8SXgbUrbyc"
  ▶ items: Array(5)
  ▶ 0:
    ▶ contentDetails: {duration: 'PT3M43S', dimension: '2d', definition: 'hd', caption: 'true', licensedContent: true, ...}
      etag: "tz3yN4XlbPrkVvehZ0S3ftkazH0"
      id: "32RAq6JzY-w"
      kind: "youtube#video"
    ▶ snippet: {publishedAt: '2023-02-10T15:44:35Z', channelId: 'UCJCx8a0rdx_ueXpmxD2od0', title: 'FAST X | Official Trailer', ...}
    ▶ statistics: {viewCount: '8228549', likeCount: '153714', favoriteCount: '0', commentCount: '14998'}
    ▶ [[Prototype]]: Object
  ▶ 1: {kind: 'youtube#video', etag: 'dk37kDGm17kG9NyN-CHxQ6JYJM0', id: 'ujM487iP518', snippet: {...}, contentDetails: {...}, ...}
  ▶ 2: {kind: 'youtube#video', etag: 'JZSu0QtcfyTWULCVewSnjqMSc0K', id: 'TNK_J0kuSVY', snippet: {...}, contentDetails: {...}, ...}
  ▶ 3: {kind: 'youtube#video', etag: 'FKoghiWXS5tV_7M3luecherRMARq', id: 's1s33eHW70', snippet: {...}, contentDetails: {...}, ...}
  ▶ 4: {kind: 'youtube#video', etag: 'vfi2nXff9yeNChp3aFLSzSMHngk', id: 'p50ARgkwESg', snippet: {...}, contentDetails: {...}, ...}
  length: 5
  ▶ [[Prototype]]: Array(0)
  kind: "youtube#videoListResponse"
  nextPageToken: "CAUQAA"
  ▶ pageInfo: {totalResults: 200, resultsPerPage: 5}
  ▶ [[Prototype]]: Object
  >

```

Now Let us save this data inside a state variable and map over it on our video cards

```

components / 35 VideoContainer.js / 102 VideoContainer / 103 videos
import React, { useEffect, useState } from "react";
import { YOUTUBE_VIDEOS_API } from "../utils/contants";

const VideoContainer = () => {
  const [videos, setVideos] = useState([]);

  useEffect(() => {
    getVideos();
  }, []);

  const getVideos = async () => {
    const data = await fetch(YOUTUBE_VIDEOS_API);
    const json = await data.json();
    console.log(json.items);
    setVideos(json.items);
  };

  return <div>VideoContainer</div>;
};

export default VideoContainer;

```

Let us build VideoCard component and pass video data as props

For test, we pass first element of data array

```

<VideoCard info={videos[0]} />

```

First, make it work for 1, then scale it.

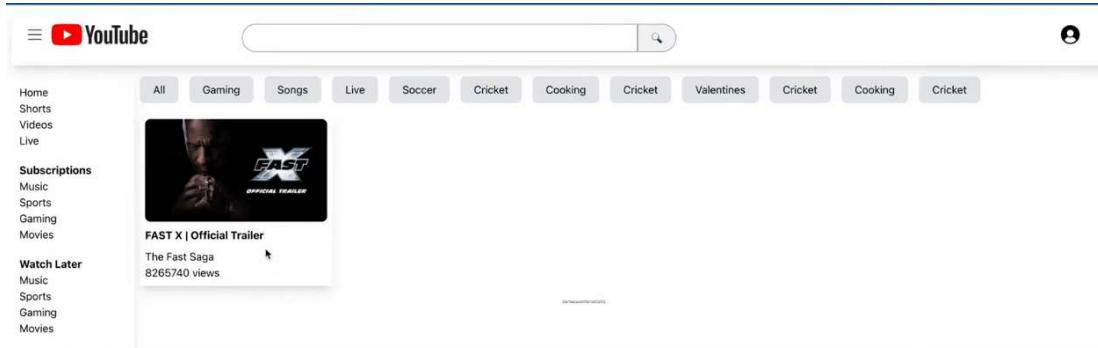
Let us build our video card now.

Now we extract data from our API and map our video cards

```

src > components > JS VideoCard.js > [o] VideoCard
1  import React from "react";
2
3  const VideoCard = ({ info }) => {
4    console.log(info);
5    const { snippet, statistics } = info;
6    const { channelTitle, title, thumbnails } = snippet;
7
8    return (
9      <div className="p-2 m-2 w-72 shadow-lg">
10        <img className="rounded-lg" alt="thumbnail" src={thumbnails.medium.url} />
11        <ul>
12          <li className="font-bold py-2">{title}</li>
13          <li>{channelTitle}</li>
14          <li>{statistics.viewCount} views</li>
15        </ul>
16      </div>
17    );
18  };
19
20  export default VideoCard;
21

```



Let us now map all our videos in VideoContainer.js

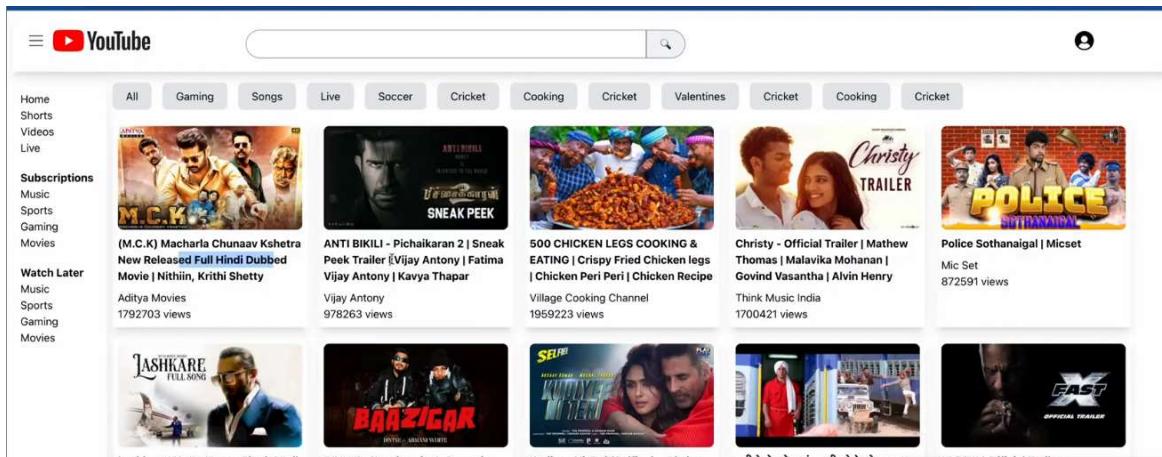
```

return (
  <div>
    {videos.map(video =><VideoCard key={video.id} info={video} />)}
  </div>
);
}

export default VideoContainer;

```

Output



Let us build watch page, but before that we need to setup the routing  
So "npm install react-router-dom"

Let us make the BrowserRouter in our App.js

If our URL changes we only want Body to change, sidebar and header remains same

We need our Body to be shown on route so

```
src > JS App.js > appRouter > element
You, 1 second ago | 1 author (You)
1 import { Provider } from "react-redux";
2 import { createBrowserRouter, RouterProvider } from "react-router-dom";
3 import "./App.css";
4 import Body from "./components/Body";
5 import Head from "./components/Head";
6 import store from "./utils/store";
7
8 const appRouter = createBrowserRouter([
9   path: "/",
10   element: <Body/>
11 ])
12
13 function App() {
14   return (
15     <Provider store={store}>
16       <div>
17         <Head />
18         <RouterProvider router={appRouter}>/>
19       </div>
20     </Provider>
21   )
22 }
23
24 export default App;
```

Our body has 2 children, Sidebar and MainContainer  
We want on change of Router, MainContainer should get replaced with videoCard  
So we use outlet in body.js instead of MainContainer or VideoCard

```

src > components > JS Body.js > [o] Body
1  import React from "react";
2  import { Outlet } from "react-router-dom";
3  import MainContainer from "./MainContainer";
4  import Sidebar from "./Sidebar";
5
6  const Body = () => {
7    return [
8      <div className="flex">
9        <Sidebar />
10       <Outlet />
11     </div>
12   ];
13 };
14
15 export default Body;
16

```

If our path is "/" then MainContainer should be shown inside body  
If our path is "/watch" then WatchPage should be shown

```

src > JS App.js > [o] appRouter
5  import Head from "./components/Head";
6  import MainContainer from "./components/MainContainer";
7  import WatchPage from "./components/WatchPage";
8  import store from "./utils/store";
9
10 const appRouter = createBrowserRouter([
11   {
12     path: "/",
13     element: <Body />,
14     children: [
15       {
16         path: "/",
17         element: <MainContainer />,
18       },
19       {
20         path: "watch",
21         element: <WatchPage />,
22       },
23     ],
24   },
25 ]);
26
27 function App() {
28   return (
29     <Provider store={store}>
30       <div>

```

Now we need to get the query param to load any particular video.  
Everything after ? is query. We need this video.id to provide as query so that we can show any particular video in our watch page



Let us make all video cards clickable

```
return (
  <div className="flex flex-wrap">
    {videos.map((video) => (
      <Link to={`/watch?v=${video.id}`}>
        <VideoCard key={video.id} info={video} />
      </Link>
    )));
  </div>
);

export default VideoContainer;
```

Now we want when we go to watch page, we want our sidebar to close completely.  
So we dispatch an action once, when we go to watch page using useEffect  
We do not want our sidebar to toggle, we just need to close it completely so  
So we make another action of closing Sidebar inside appSlice.js

```
src > utils > JS appSlice.js > [o] closeMenu
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const appSlice = createSlice({
4   name: "app",
5   initialState: {
6     isMenuOpen: true,
7   },
8   reducers: {
9     toggleMenu: (state) => {
10       state.isMenuOpen = !state.isMenuOpen;
11     },
12     closeMenu: (state) => {
13       state.isMenuOpen = false;
14     },
15   },
16 );
17
18 export const [ toggleMenu, closeMenu ] = appSlice.actions;
19 export default appSlice.reducer;
```

Let us dispatch the action in WatcherPage.js  
We dispatch action inside useEffect as we only want this action to be dispatched only once.

```

src > components > JS WatchPage.js > ...
1 import React, { useEffect } from "react";
2 import { useDispatch } from "react-redux";
3 import { closeMenu } from "../utils/appSlice";
4
5 const WatchPage = () => {
6   const dispatch = useDispatch();
7   useEffect(()=>{
8     dispatch( closeMenu )
9   },[])
10  return <div>WatchPage</div>;
11};
12
13 export default WatchPage;
14

```

Inside our watch page, first we use useSearchParams from react-router-dom

Example:

```

import * as React from "react";
import { useSearchParams } from "react-router-dom";

function App() {
  let [searchParams, setSearchParams] = useSearchParams();

  function handleSubmit(event) {
    event.preventDefault();
    // The serialize function here would be responsible for
    // creating an object of { key: value } pairs from the
    // fields in the form that make up the query.
    let params = serializeFormQuery(event.target);
    setSearchParams(params);
  }

  return (
    <div>
      <form onSubmit={handleSubmit}>{/* ... */}</form>
    </div>
  );
}

```

We have some function like searchParams.get("v")

Using this it will give us whatever is after "v=" in our Route and that is what we need actually.

Either we can do the API call and filter the data based on ID we have and show that particular video or we can embed it using frame by going in share button of video

Share

<https://youtu.be/1uDl6cjH6Mc>

Start at 0:00

FAST X | Official Hindi Trailer (Universal Studios) - HD

Watch later

**OFFICIAL TRAILER**

Embed Video

```
<iframe width="560" height="315"
src="https://www.youtube.com/embed/1uDl6cjH6Mc" title="YouTube video
player" frameborder="0"
allow="accelerometer; autoplay;
clipboard-write; encrypted-media;
gyroscope; picture-in-picture; web-
share" allowfullscreen></iframe>
```

Start at 0:00

EMBED OPTIONS:

Show player controls.

```

const dispatch = useDispatch();
useEffect(() => {
  dispatch(closeMenu());
}, []);
return [
  <div>
    <iframe
      width="560"
      height="315"
      src="https://www.youtube.com/embed/1uDl6cjH6Mc"
      title="YouTube video player"
      frameborder="0"
      allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share"
      allowfullscreen
    ></iframe>
  </div>
];
};

export default WatchPage;

```

Now we can make API call and get title etc and show it in our watchPage

## Session 17

### Higher Order components

A component which takes an component and returns a new component and component is

just a function only

Let us make one function which takes a VideoCard component and returns VideoCard component.

### What Is use of it??

Suppose if we need some modification inside our videoCard, suppose we want border around our video card, we wrap our return statement inside div and style it.

```
const RedBorderVideoCard = (VideoCard) => {
  return (
    <div className="p-1 m-1 border border-red-900">
      <VideoCard />
    </div>
  );
};

export default VideoCard;
```

### Where to use it?

In Youtube, there is one Advertisement card and then there is video card

In the advertisement card we have little different structure, although its same as video card but little different functioning.



IIT Madras CCE - Data Science & AI  
Real-world Experience from 50+ Projects & Case Studies. Get Certified from IIT Madras CCE.  
Ad - Intellipaat

Full movie YouTube pe daal do... 6 Billion views done 🎉  
BnTV 🎯  
402K views • 1 day ago

So we can make AdVideoCard and we modify videoCard inside it and render it.

```
const AdVideoCard = (VideoCard) => {
  return (
    <div className="p-1 m-1 border border-red-900">
      <VideoCard />
    </div>
  );
};

export default VideoCard;
```

Let us call our AdVideoCard inside videoContainer

```

export const AdVideoCard = ({ info }) => {
  return [
    <div className="p-1 m-1 border border-red-900">
      <VideoCard info={info} />
    </div>
  ];
};

export default VideoCard;

```

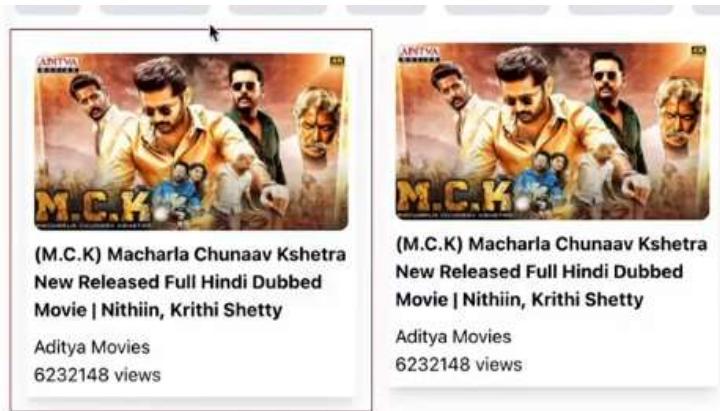
```

return (
  <div className="flex flex-wrap">
    {videos[0] && <AdVideoCard info={videos[0]} />}
    {videos.map((video) => (
      <Link key={video.id} to={"/watch?v=" + video.id}>
        <VideoCard info={video} />
      </Link>
    ))}
  </div>
);
};

export default VideoContainer;

```

## Output



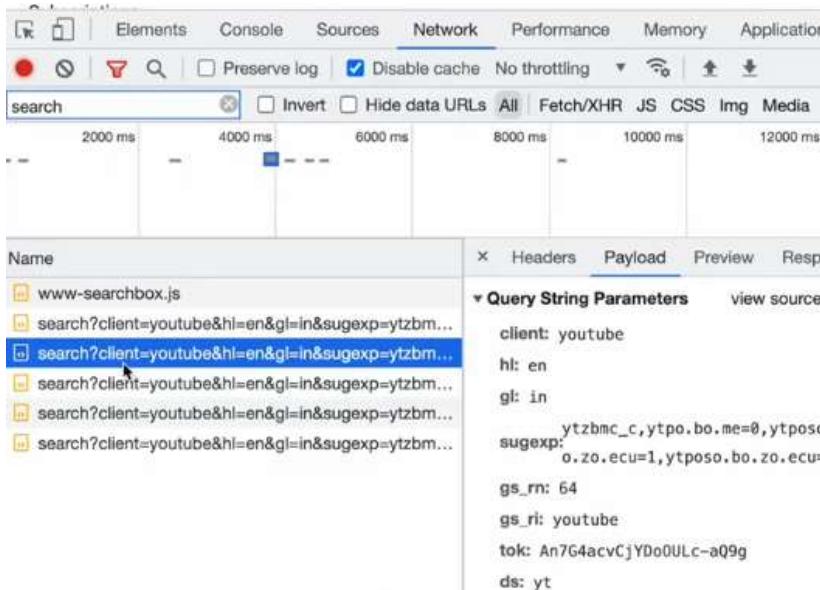
So higher order component takes up an component and modifies it and return it.

## Build a Search Bar in your Youtube App

Cool thing about search bar of Youtube is

You write "india" and it shows search related to india and makes an API call  
We write something further, it again makes API call and shows data.

When we are writing it does not make an API call on each and every key press.

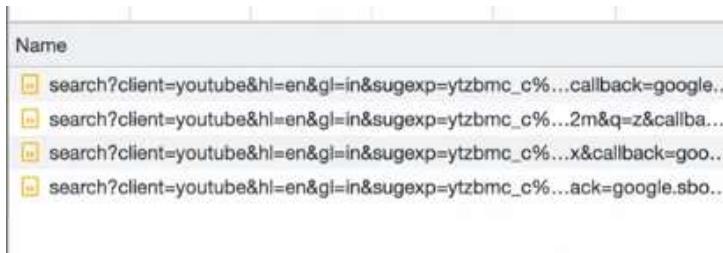


Let us say we typed very fast

We typed 7 letters



It makes 3 API calls for 7 key strokes



Let us see when it does API calls

Zeroth API call on " " empty

First API call on press of "z"

Second API call on "zx"

Third API call on "zxcvbnm"

So when we are typing very fast, it skips some event. This is called **Debouncing**

If we type slowly it makes API call for each and every key strokes.

Concept Behind this is **Debouncing**

And it says, when we are typing very fast, the difference between 2 key strokes is very less and when we type slow, difference is large.

When someone is typing very fast, do we need to see intermediate results??

No, so we make less API calls and it becomes performance efficient for us.

When 1L people make 3 API call each, it makes a huge performance optimisation.

Why are they even showing result? Can they make search on pressing Search button only?  
To enhance User Experience

**Debouncing with 200ms means between 2 key strokes, difference is less than 200ms then Decline the API call. If its greater than 200ms, make an API call.**

Debouncing time for different websites is different.

Debouncing of Flipkart may be different than Youtube, Their API may be more optimised,  
They do this for better User Experience.

### Let us **build this Debouncing Feature**

Search google search API in internet.  
We give different queries and we get results accordingly.



```
<?xml version="1.0" encoding="UTF-8"?>
<toplevel>
  <CompleteSuggestion>
    <suggestion data="iphone 13"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 11"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 14"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 12"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 14 pro max"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 14 pro"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone x"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone "/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 13 pro max"/>
  </CompleteSuggestion>
  <CompleteSuggestion>
    <suggestion data="iphone 13 pro"/>
  </CompleteSuggestion>
</toplevel>
```

Let us try to fetch data from this API in our Browser itself.



We see response in Network Tab

The screenshot shows the Network tab in the Chrome DevTools interface. At the top, there are several filter options: 'Preserve log' (checked), 'Disable cache' (checked), and 'No throttling'. Below the filters, a 'Filter' input field is present. The main area displays a table of network requests. The columns are: Name, Headers, Payload, Preview, Response, and Initiator. The 'Preview' column shows the XML structure of a response, specifically focusing on a 'CompleteSuggestion' node with a 'data' attribute set to 'iphone 13'. The 'Payload' column also shows the same XML structure. The 'Name' column lists various request URLs, such as '/log\_event?alt=json&key=AlzaSyAO\_FJ2SlqU8Q4ST...', '/GenerateIT', and '/search?output=toolbar&hl=en&q=iphone'. The bottom of the table shows summary statistics: '30 / 288 requests' and '447 kB / 13.2 MB transferred'.

**Let us use a Youtube Search Suggestion API which returns data in JSON**

Ok I found this URL:

<http://suggestqueries.google.com/complete/search?client=firefox&ds=yt&q=Query>

It isn't part of Youtube API, but still works, returns a JSON response.

The screenshot shows the Network tab in the Chrome DevTools. The requests listed are:

- id
- Create
- log\_event?alt=json&key=AlzaSyAO\_FJ2SlqU8Q4ST...
- GenerateIT
- log\_event?alt=json&key=AlzaSyAO\_FJ2SlqU8Q4ST...
- log\_event?alt=json&key=AlzaSyAO\_FJ2SlqU8Q4ST...
- videos?part=snippet%2CcontentDetails%2Cstatistics
- videos?part=snippet%2CcontentDetails%2Cstatistics
- search?output=toolbar&hl=en&q=iphone
- search?client=firefox&ds=yt&q=iphone

For the last request, the Headers, Payload, Preview, Response, and Initiator tabs are visible. The Preview tab shows the JSON payload:

```
["iphone",...]
  0: "iphone"
  ▾ 1: ["iphone 14", "iphone ringtone", "iphone 14"
    0: "iphone 14"
    1: "iphone ringtone"
    2: "iphone 14 pro max"
    3: "iphone"
    4: "iphone 13"
    5: "iphone 11"
    6: "iphone 12"
    7: "iphone 14 pro"
    8: "iphone 15"
    9: "iphone 7"
  2: []
  ▾ 3: { ... }
```

Let us export API URL in constant.js, for now, we leave the query empty

```
src > utils > JS contents.js > ...
You, 1 second ago | 1 author (You)
1 const GOOGLE_API_KEY = "AIzaSyBh724vdsU4vjXtUqZjkF2mMJwYqoVB2LQ";
2
3 export const YOUTUBE_VIDEOS_API =
4   "https://youtube.googleapis.com/youtube/v3/videos?part=snippet%26contentDetails"
5   GOOGLE_API_KEY;
6
7 export const YOUTUBE_SEARCH_API =
8   "http://suggestqueries.google.com/complete/search?client=firefox&ds=yt&q=";
9
```

Now we write logic for input field of search bar.

We make a state variable and we write an onChange function.

Initial state = ""

```
const [searchQuery, setSearchQuery] = useState("");
```

```
<input
  className="w-1/2 border border-gray-400 p-2 rounded-l-full"
  type="text"
  value={searchQuery}
  onChange={(e) => setSearchQuery(e.target.value)}>
```

Let us make useEffect to make an API call

We need to make an API call everytime Search query changes

```
useEffect(() => {
  // API call
  console.log(searchQuery);
}, [searchQuery]);
```

We need to make an API call if difference between 2 key strokes is more than 200ms otherwise decline it.

```
// make an api call after every key press
// but if the difference between 2 API calls is <200ms
// decline the API call and
```

Let us make the API call

```

useEffect(() => [
  // API call
  console.log(searchQuery);

  // make an api call after ever key press
  // but if the difference between 2 APi calls is <200ms
  // decline the API call
  getSearchSugsestions();
], [searchQuery]);

const getSearchSugsestions = async () => {
  const data = await fetch(YOUTUBE_SEARCH_API + searchQuery);
  const json = await data.json();
  console.log(json);
};

```

If difference between 2 API call is <200ms decline it, how to do this??

We use setTimeout for 200ms

But our useEffect depends on SearchQuery so everytime we change searchQuery, an setTimeout is generated and API call is made

```

useEffect(() => {
  // API call
  console.log(searchQuery);

  // make an api call after ever key press
  // but if the difference between 2 APi calls is <200ms
  // decline the API call

  setTimeout(() => getSearchSugsestions(), 200);      You
}, [searchQuery]);

```

So on every key press, a new 200ms timeout is made due to useEffect

```

/*
 * key - i
 * - render the component
 * - useEffect();
 * - start timer => make api call after 200 ms
 *
 * key - ip
 * - render the component
 * - useEffect()
 * - start timer => make api call after 200 ms
 */

```

So, we need to clear setTimeout also inside useEffect under ComponentWillUnmount

```

useEffect(() => {
  // API call
  console.log(searchQuery);

  // make an api call after ever key press
  // but if the difference between 2 APi calls is <200ms
  // decline the API call

  const timer = setTimeout(() => getSearchSugsestions(), 200);

  return () => [
    clearTimeout(timer),      You, 1 second ago + Uncommitted changes
  ];
}, [searchQuery]);

```

Now suppose if we press another key before 200ms, It will first destroy previous timeout and start another fresh timeout.

When timeout exceed 200ms, it makes API call

```

useEffect(() => [
  // API call
  console.log(searchQuery);
  You, 18 seconds ago + Uncommitted changes
  const timer = setTimeout(() => getSearchSugsestions(), 200);

  return () => [
    clearTimeout(timer),  []
  ];
], [searchQuery]);

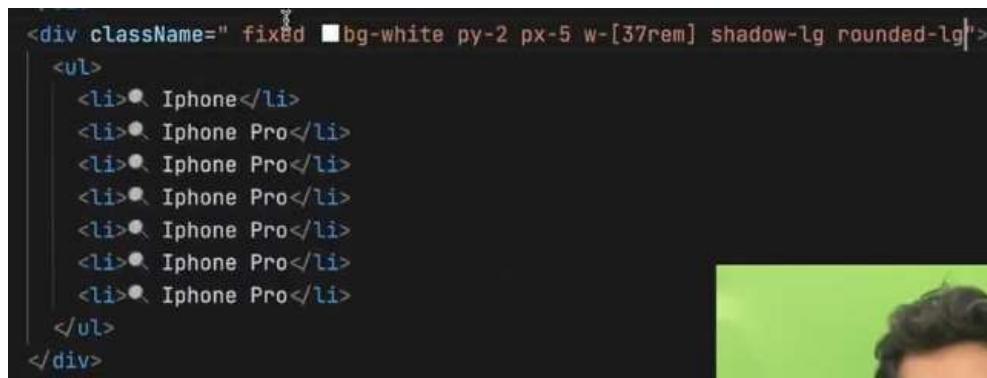
const getSearchSugsestions = async () => {
  const data = await fetch(YOUTUBE_SEARCH_API + searchQuery);
  const json = await data.json();
  console.log(json[1]);
};

```

In youtube when we type something on search, it shows suggestions like this

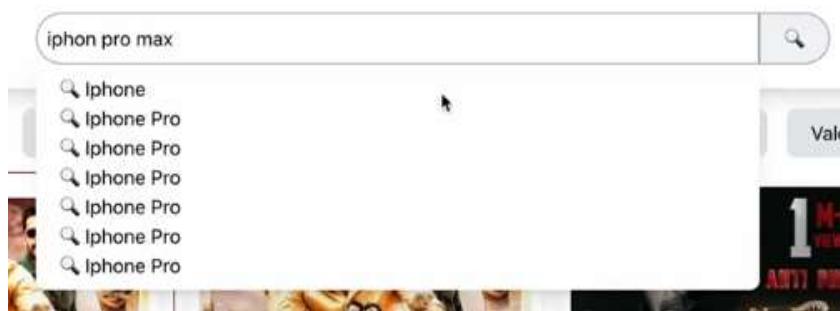


Let us make another div and populate our search results in it.



```
<div className="fixed bg-white py-2 px-5 w-[37rem] shadow-lg rounded-lg">
  <ul>
    <li>● Iphone</li>
    <li>● Iphone Pro</li>
    <li>● Iphone Pro</li>
  </ul>
</div>
```

Output



We take suggestion in a state variable

```
const [searchQuery, setSearchQuery] = useState("");
const [suggestions, setSuggestions] = useState([]);
```

```

const getSearchSuggestions = async () => {
  console.log("API CALL - " + searchQuery);
  const data = await fetch(YOUTUBE_SEARCH_API + searchQuery);
  const json = await data.json();
  //console.log(json[1]);
  setSuggestions(json[1]);
};


```

We map on these suggestions

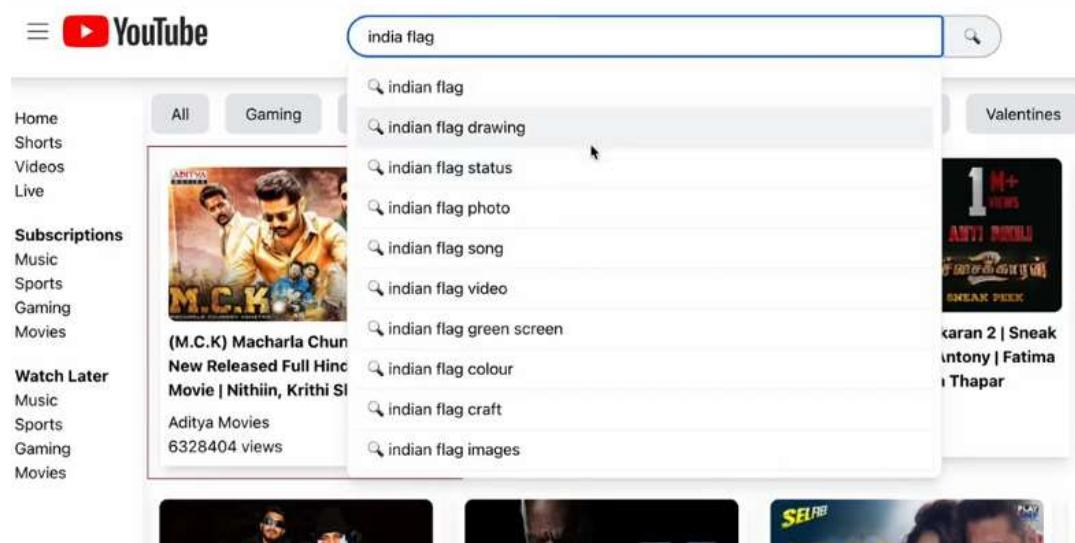
```

<div className="fixed bg-white py-2 px-2 w-[37rem] shadow-lg rounded-lg border border-gray-200">
  <ul>
    {suggestions.map((s) => (
      <li key={s} className="py-2 px-3 shadow-sm hover:bg-gray-100">
        {s}
      </li>
    ))}
  </ul>
</div>

```



Output



There are some issues with the CSS, if we click outside search box, our suggestion should turn off so we make another state for showSuggestions

```

const [showSuggestions, setShowSuggestions] = useState(false);

{showSuggestions && [
  <div className="fixed bg-white py-2 px-2 w-[37rem] shadow-lg rounded-lg border border-gray-200">
    <ul>
      {suggestions.map((s) => (
        <li key={s} className="py-2 px-3 shadow-sm hover:bg-gray-100">
          {s}
        </li>
      ))}
    </ul>
  </div>
]
}


```

In our input, we use onFocus and make setShow = true and onBlur means on clicking out, make it false

```
<input  
  className="px-5 w-1/2 border border-gray-400 p-2 rounded-l-full"  
  type="text"  
  value={searchQuery}  
  onChange={(e) => setSearchQuery(e.target.value)}  
  onFocus={() => setShowSuggestions(true)}  
  onBlur={() => setShowSuggestions(false)}>  
</div>
```

When we write "iphone"

We make optimised API calls and get the results

But when we remove "e" it again makes API call

We remove "n" also we have "iphon" it makes another API call

We remove "o" also we have "iph" It makes another API call and so on.

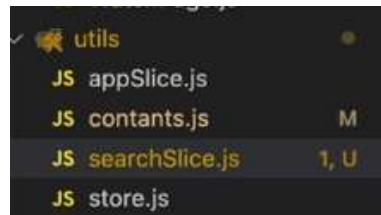
It should not make an API call as we have already made API call for that data.

So let us optimise Search more by **Caching**

**Redux helps us in doing Caching.**

Whenever we search something, it stores the API call response inside Redux and help us in caching.

Let us make another Slice in store SearchSlice



Let us see **which Data structure to use to store searchResult?**

If we use an Array, we need Linear search to find any element in array and it takes  $O(n)$   
Array.indexOf() also takes  $O(n)$

In a map,  $TC = O(1)$  for searching so we can use hashmap

So we can use map, set

Let us use object for now whose initialState is empty object

```
src > utils > JS searchSlice.js > ↗ searchSlice > ↗ reducers > ↗ cacheResults
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const searchSlice = createSlice({
4   name: "search",
5   initialState: {},
6   reducers: {
7     cacheResults: (state, action) => [
8       state = Object.assign(state, action.payload);
9     ],
10   },
11 });
12
13 export const { cacheResults } = searchSlice.actions;
14
15 export default searchSlice.reducer;
16
```

Let us put this slice in our store

```
src > utils > JS store.js > ↗ store > ↗ reducer > ↗ search
...
1 import { configureStore } from "@reduxjs/toolkit";
2 import appSlice from "./appSlice";
3 import searchSlice from "./searchSlice";
4
5 const store = configureStore({
6   reducer: [
7     app: appSlice,
8     search: searchSlice
9   ],
10 });
11
12 export default store;
13
```

We are making API call in Head.js

```

const searchCache = useSelector((store) => store.search);

/**
 *  searchCache = {
 *    "iphone": ["iphone-11", "iphone-14"] You, 8 sec
 *  }
 *  searchQuery = iphone
 */

useEffect(() => {
  const timer = setTimeout(() => {
    if (searchCache[searchQuery]) {
      setSuggestions(searchCache[searchQuery]);
    } else {
      getSearchSugsestions();
    }
  }, 200);

  return () => {
    clearTimeout(timer);
  };
}, [searchQuery]);

```

If its not present in Cache, make an API call and dispatch an action of cacheResults

```

const searchCache = useSelector((store) => store.search);
const dispatch = useDispatch();

```

Make an API call and store the result in cache. We do [searchQuery] because we cannot store it any other way.

```

const getSearchSugsestions = async () => [
  console.log("API CALL - " + searchQuery),
  const data = await fetch(YOUTUBE_SEARCH_API + searchQuery);
  const json = await data.json();
  //console.log(json[1]);
  setSuggestions(json[1]);

  // update cache
  dispatch(
    cacheResults({
      [searchQuery]: json[1],
    })
  );
]; You, 1 second ago • Uncommitted changes
];

```

Now it will make API call only for things it does not have cached.

We can think that our Store will be flooded with the data if we store results this way so to

avoid it we can also use LRU cache (Least Recently used)

We can do something like "remove result if number of results>100"

## Let us make Comment Section now

We go in our watch page

In youtube our comment section is just 2 level deep, comment and reply

OGS 1 year ago (edited)  
Four things I like about Akshay.  
1) Doesn't try and speak in an American accent, like most Indian Youtube programmers.  
2) Passionate  
3) Someone with real knowledge...  
Read more

152 Reply

Akshay Saini 1 year ago  
Love you!

16 Reply

Isaac Prosper 10 months ago  
hes very authentic,first time i thought he was quite weird owing to the fact that im not indian but now i cant get enough of his content

13 Reply

Prajwal 5 months ago  
That's what make him unique!

1 Reply

We will build it n-level nested comments, it will go n level deep in comment

In reddit, it has this feature

The screenshot shows a nested comment structure. At the top, a comment from 'pikachootrain' is timestamped 'less than a minute ago'. Below it, a reply from 'kijnkevin' says 'Okay'. This is followed by a reply from 'pikachootrain' saying 'NICE NESTED COMMENT'. Another reply from 'pikachootrain' follows, saying 'nice.' A reply from 'kijnkevin' then says 'asdkljzxclnxzm,caskldjsakldjklas'. Finally, another reply from 'pikachootrain' ends with '???'.

To build this structure, we need UI, Data structure knowledge  
Youtube API does not give n level nested comments, we need to make dummy data.

We make CommentContainer.js



And load it inside watchPage component

```
</div>
<CommentsContainer />
```

#### How our data is structured??

It is just Array of objects with name and text and replies which can be an array of object which will again have name, text and replies.

```
const commentsData = [
  {
    name: "Akshay Saini",
    text: "Lorem ipsum dolor sit amet, consectetur adip",
    replies: [
    ]
  },
  {
    name: "Akshay Saini",
    text: "Lorem ipsum dolor sit amet, consectetur adip",
    replies: [
    ]
  }
]
[{
  name: "Akshay Saini",
  text: "Lorem ipsum dolor sit amet, consectetur adip",
  replies: [
  ]
}]]
```

Replies itself can have multiple replies and again and again.  
So, comments can be nested in n levels.

```
{
  name: "Akshay Saini",
  text: "Lorem ipsum dolor sit amet, consectetur adip",
  replies: [
    {
      name: "Akshay Saini",
      text: "Lorem ipsum dolor sit amet, consectetur adip",
      replies: []
    },
    {
      name: "Akshay Saini",
      text: "Lorem ipsum dolor sit amet, consectetur adip",
      replies: []
    },
    {
      name: "Akshay Saini",
      text: "Lorem ipsum dolor sit amet, consectetur adip",
      replies: []
    }
  ]
},
```

Let us display our 1 comment inside comment component which we show here only

```

src > components > JS CommentsContainer.js > [e] Comment
  1 import React from "react";
  2
  3 > const commentsData = [...];
  4
  5
  6 const Comment = ({ data }) => {
  7   const { name, text, replies } = data;
  8   return (
  9     <div className="flex">
 10       
 16         <p className="font-bold">{name}</p>
 17         <p>{text}</p>
 18       </div>
 19     </div>
 20   );
 21 };
 22
 23 const CommentsContainer = () => {
 24   return (
 25     <div className="m-5 p-2">
 26       <h1 className="text-2xl font-bold">Comments: </h1>
 27       <Comment data={commentsData[0]} />
 28     </div>
 29   );
 30 };
 31
 32 export default CommentsContainer;

```

Now we use Recursion to list down nested comments

We make component CommentList which takes whole comment data and maps over

```

> const commentsData = [...];
>
> const Comment = ({ data }) => {...};
>
const CommentsList = (comments) => {
  return (
    <div>
      {comments.map((comment) => {
        return <Comment data={comment} />;
      })}
    </div>
  );
}

```

```

src > components > JS CommentsContainer.js > [x] CommentsContainer
  1 import React from "react";
  2
  3 > const commentsData = [...];
  4
  5
  6 > const Comment = ({ data }) => {
  7   ...
  8 };
  9
 10
 11 const CommentsList = ({ comments }) => {
 12   return comments.map((comment) => <Comment data={comment} />);
 13 };
 14
 15
 16 const CommentsContainer = () => {
 17   return [
 18     <div className="m-5 p-2">
 19       <h1 className="text-2xl font-bold">Comments: </h1>
 20       <CommentsList comments={commentsData} />
 21     </div>
 22   ];
 23 };
 24
 25
 26 export default CommentsContainer;
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107

```

## Output

We have our first level comments from data, now how to display replies array also



### Comments:

- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip
- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip
- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip
- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip
- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip
- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip
- Akshay Saini**  
 Lorem ipsum dolor sit amet, consectetur adip

How to show these replies??

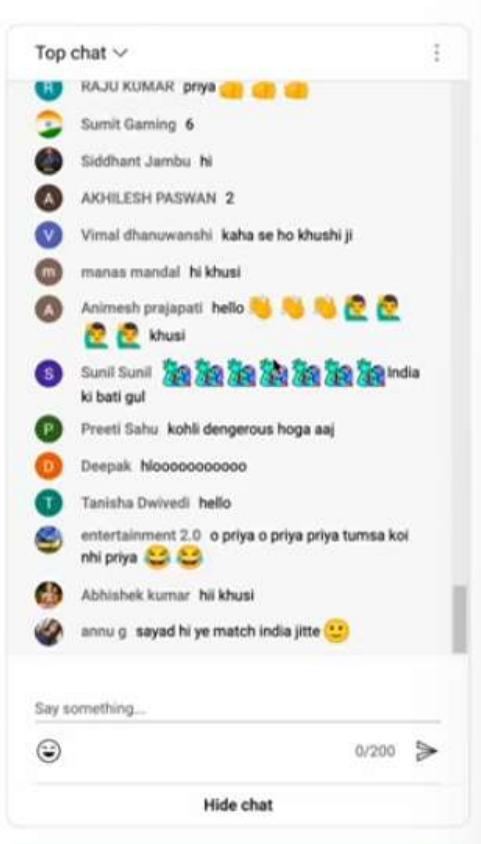
They are just another CommentList or list of comments. So we again call CommentList inside itself and pass replies recursively.

```
const CommentsList = ({ comments }) => {
  // Disclaimer: Don't use indexes as keys
  return comments.map((comment, index) => (
    <div>
      <Comment key={index} data={comment} />
      <div className="pl-5 border border-l-black ml-5">
        <CommentsList comments={comment.replies} />
      </div>
    </div>
  )));
};
```



## Session 18

Let us build [Live chat in youtube](#)



### How are these new chat popping up live and how is it loading these new chats quickly?

Streaming applications has live data, Trading app like Zerodha where UI is changing again and again. Cricbuzz where live data updates automatically.

There are 2 types of applications:

1. Which has live streaming
2. Which does not have live streaming

### Challenges of live chat

1. Get data live (Data Layer problem)
2. How do we update the data on UI (UI layer problem)

If we make a div for each chat and we keep on pushing div in our DOM, our page size will increase a lot, it might make our app very slow.

In live streaming apps, we need to update UI in an efficient way and give good user experience.

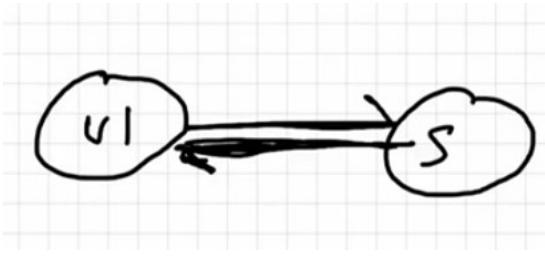
### How to handle live data??

There are 2 ways to handle live data one of it is **Web sockets**.

Web sockets is a 2 way handshake between UI and server.

Once handshake is made, we can quickly send data from either sides. So it is bi-directional live data. We can send from UI to server and vice-versa.

In web-sockets initial connection takes time but there is no regular interval, data might come in any random time.



Another approach is **API Polling**

UI requests the server and data flows from server to UI (its uni-directional).

There is an time Interval and UI keep polling data after certain time interval from server.

**What should be use if we are building Gmail?**

**Do we need email realtime??**

No, if I get my email after 10sec it is also fine so we do not need to use web sockets for Gmail, we can work in API polling.

**For Applications like Zerodha or any other stock trading platform?**

We need realtime precise data and we need every milisecond information so we use web-sockets.

**For whatsapp or any chat application?**

API polling cannot work for whatapp, it needs realtime data otherwise chats might be missed.

There are applications which have time stamps like in whatsapp we have time with each message so this is a sign that this app uses Web sockets.

**What does cricbuzz do??**

We see in cricbuzz, after every 25 seconds an API call is made for live commentary means API polling is happenning.

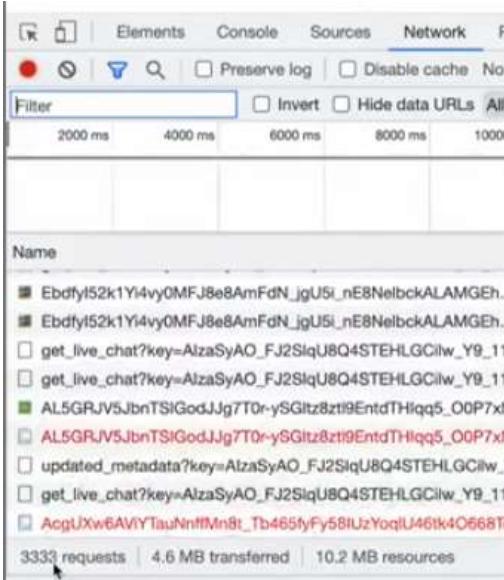
**Why 25 seconds?**

It might happen that average time it takes to bowl one bowl is 25 seconds. This could be a possible reason.

**Live section of Youtube uses??**

It uses **API polling**, it has made 3333 and continued requests for just one video where youtube might have 100's such live videos.

So, we can say API calling is not that expensive.



We observe after every 1.5seconds, youtube is making API call.

Youtube can make it 5 seconds also but it does not because it wants to give good UX, it does not show timestamps because we do not need order in live chat. Order of chat might be random.

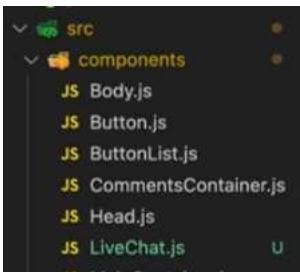
For every 1.5second, it takes all message in that time stamp and populate in the UI

#### How our page/UI is handling so much??

Because as soon as messages exceed a certain number, youtube automatically removes the messages from the top, say after 200 messages it will remove messages so instead of putting all messages on DOM, it removes some messages otherwise our DOM will explode

### Let us build it

We make a liveChat.js component.

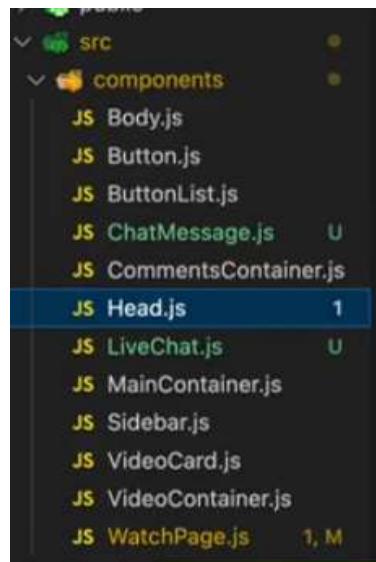


Import it inside watchPage component

```
</div>
<div>
  <LiveChat />
</div>
</div>
<CommentsContainer />

```

Let us make an component for chatMessage also like



LiveChat.js

```
src > components > JS LiveChat.js > [e] LiveChat
1 import ChatMessage from "./ChatMessage";
2
3 const LiveChat = () => {
4   return [
5     <div className="w-full h-[600px] ml-2 p-2 border border-black bg-slate-100 rounded-lg">
6       <ChatMessage
7         name="Akshay Saini"
8         message="This is Namaste React Live! 🙏"
9       />
10      </div>
11    ];
12  };
13 export default LiveChat;
14
```

chatMessage.js

```

src > components > JS ChatMessage.js > ChatMessage
1  const ChatMessage = ({ name, message }) => {
2    return [
3      <div className="flex items-center">
4        
9        <span className="font-bold px-2">{name}</span>
10       <span>{message}</span>
11     </div>
12   ];
13 };
14 export default ChatMessage;
15

```

Never just directly use map function.

Try to render one element first and on successfully rendering it. Then apply map function on whole data with key

Let us bring some data using polling

For calling fetch we use useEffect, to do polling we use setInterval for 2 second and make sure to clearInterval also.

```

useEffect(() => {
  const i = setInterval(() => {
    // API Polling
    console.log("API Polling");
  }, 2000);

  return () => clearInterval(i);
}, []);

```

Now to store chats, either we can make state variable or we can make an action for each API poll and dispatch an action which puts the data in our chatSlice

We make chatSlice.js



We do array.unshift so that we push element in array from front so that latest chat is at last as flow of our chat will be downward to upward

```
src > utils > JS chatSlice.js > [!] chatSlice > ↗ reducers > ⚡ addMessage
1 import { createSlice } from "@reduxjs/toolkit";
2
3 const chatSlice = createSlice({
4   name: "chat",
5   initialState: {
6     messages: [],
7   },
8   reducers: {
9     addMessage: (state, action) => {
10       state.messages.unshift(action.payload);
11     },
12   },
13 });
14
15 export const { addMessage } = chatSlice.actions;
16 export default chatSlice.reducer;
17
```

Let us add this slice in the store now,

```
src > utils > JS store.js > [!] store > ↗ reducer > ⚡ chat
You, now | 1 author (You)
1 import { configureStore } from "@reduxjs/toolkit";
2 import appSlice from "./appSlice";
3 import chatSlice from "./chatSlice";
4 import searchSlice from "./searchSlice";
5
6 const store = configureStore({
7   reducer: {
8     app: appSlice,
9     search: searchSlice,
10    chat: chatSlice,      You, now + Uncommitted ch
11  },
12});
13
14 export default store;
15
```

Now we dispatch some action from API polling using `useDispatch`  
And we subscribe to messages in our store using `useSelector`.

```
6 const chatMessages = useSelector((store) => store.chat.messages);
```

```
const dispatch = useDispatch();

const chatMessages = useSelector((store) => store.chat.messages);

useEffect(() => {
  const i = setInterval(() => {
    // API Polling
    console.log("API Polling");

    dispatch(
      addMessage({
        name: "Akshay Saini",
        message: "Lorem Ipsum Dolor Site Amet ✨",
      })
    );
  }, 2000);

  return () => clearInterval(i);
}, [ ]);
```

Let us now map on our data we got from useSelector

```
return (
  <div className="w-full h-[600px] ml-2 p-2 border border-black bg-slate-100 rounded-lg">
    {
      // Disclaimer: Don't use indexes as keys
      chatMessages.map((c, i) => (
        <ChatMessage key={i} name={c.name} message={c.message} />
      ))
    }
  </div>
);
export default LiveChat;
```

To get random names, let us make helper.js in utils and write our random name generator function there

```
src > utils > JS helper.js > ⚡ generateRandomName
1 > var nameList = [...]
176 ];
177
178 export function generateRandomName() {
179   return nameList[Math.floor(Math.random() * nameList.length)];
180 }
```

Inside dispatch we call this function

```

useEffect(() => {
  const i = setInterval(() => {
    // API Polling
    console.log("API Polling");

    dispatch(
      addMessage({
        name: generateRandomName(),
        message: "Lorem Ipsum Dolor Site Amet ✎",
      })
    );
  }, 2000);

  return () => clearInterval(i);
}, []);

```

Calling this redux#Action  
PayloadAction of type T

Let us generate random text also similarly  
Inside helper.js

```

export function makeRandomMessage(length) {
  let result = "";
  const characters =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  const charactersLength = characters.length;
  let counter = 0;
  while (counter < length) {
    result += characters.charAt(Math.floor(Math.random() * charactersLength));
    counter += 1;
  }
  return result;
}

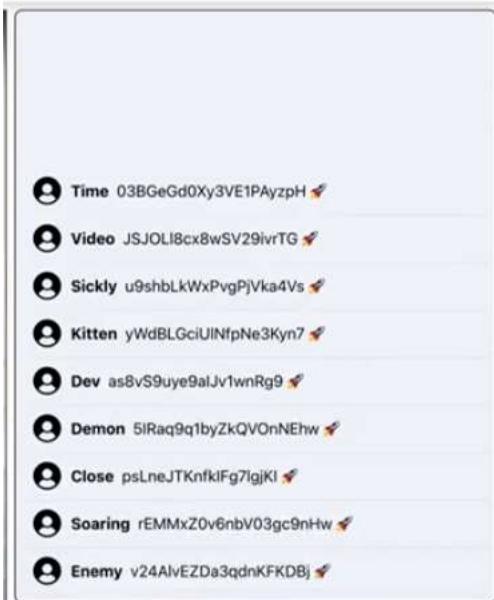
```

```

dispatch(
  addMessage({
    name: generateRandomName(),
    message: makeRandomMessage(20) + " ✎",
  })
);

```

Now we need to reverse the flow of our messages we give flex-col-reverse to make our chat go downward to upwards



But we are putting so many messages in DOM, so our screen will freeze so we need to remove the old messages after some limit.

Inside our chatSlice we do array.splice(10,1)

What it does is, if our array length>10, remove 1 element from top

Put this 10 inside constants.js

```
src > utils > JS contants.js > (e) OFFSET_LIVE_CHAT
...
1 | const GOOGLE_API_KEY = "AIzaSyAtIibgU4boIyzkgbabBCe8BJJsBSfNLWA";
2 |
3 | const OFFSET_LIVE_CHAT = 10
4 |
```

```
src > utils > JS chatSlice.js > (e) chatSlice > (p) reducers > (p) addMessage
1 | import { createSlice } from "@reduxjs/toolkit";
2 | import { OFFSET_LIVE_CHAT } from "./contants";
3 |
4 | const chatSlice = createSlice({
5 |   name: "chat",
6 |   initialState: {
7 |     messages: [],
8 |   },
9 |   reducers: {
10 |     addMessage: (state, action) => {
11 |       state.messages.splice(OFFSET_LIVE_CHAT, 1);
12 |       state.messages.unshift(action.payload);
13 |     },
14 |   },
15 | });
16 |
17 | export const { addMessage } = chatSlice.actions;
18 | export default chatSlice.reducer;
19 |
```

Let us now build an feature where user can write his own message and send it in the live chat.

Let us make an input box and make state for it

```
<div className="w-full p-2 ml-2 border border-black">
  <input
    className="w-96"
    type="text"
    value={liveMessage}
    onChange={(e) => [
      setLiveMessage(e.target.value),
    ]}
  />
  <button className="px-2 mx-2 bg-green-100">Send</button>
</div>
```

We want to send the message on press of "ENTER" also so we make it a form

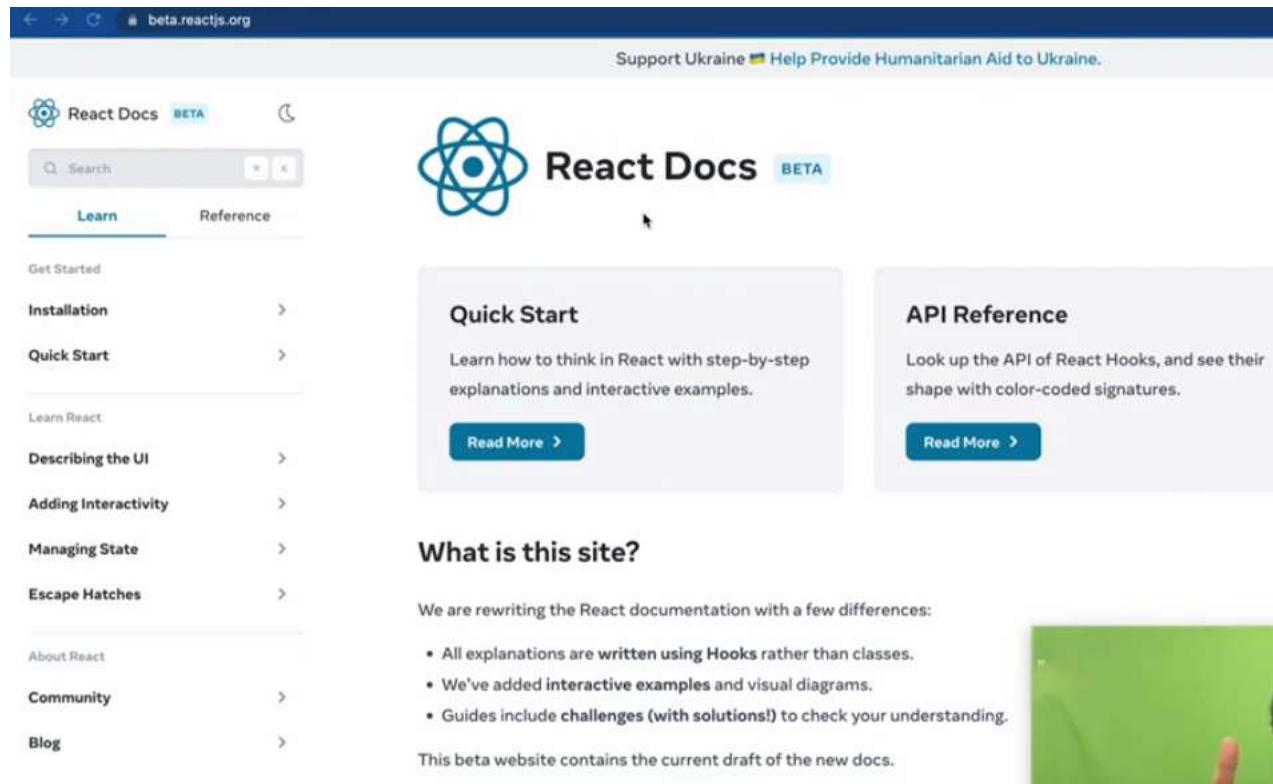
```
<form
  className="w-full p-2 ml-2 border border-black"
  onSubmit={(e) => {
    e.preventDefault();
    console.log("ON Form Submit", liveMessage);
  }}
>
  <input
    className="px-2 w-96"
    type="text"
    value={liveMessage}
    onChange={(e) => [
      setLiveMessage(e.target.value),
    ]}
  />
  <button className="px-2 mx-2 bg-green-100">Send</button>
</form>
</>
```

Now we need to dispatch an action to add message to the live chat data

```
<form
  className="w-full p-2 ml-2 border border-black"
  onSubmit={(e) => {
    e.preventDefault();
    dispatch(
      addMessage({
        name: "Akshay Saini",
        message: liveMessage,
      })
    );
  }}
>
  <input
    className="px-2 w-96"
    type="text"
    value={liveMessage}
    onChange={(e) => [
      setLiveMessage(e.target.value),
    ]}
  />
  <button className="px-2 mx-2 bg-green-100">Send</button>
</form>
```

# useCallback, useMemo, useRef hook

New react documentation, always refer it.



The screenshot shows the beta version of the React Docs website at [beta.reactjs.org](https://beta.reactjs.org). The page features a navigation bar with a search bar and tabs for 'Learn' and 'Reference'. Below the navigation is a sidebar with links to 'Get Started', 'Installation', 'Quick Start', 'Learn React', 'Describing the UI', 'Adding Interactivity', 'Managing State', 'Escape Hatches', 'About React', 'Community', and 'Blog'. The main content area has two columns: 'Quick Start' (describing step-by-step explanations and interactive examples) and 'API Reference' (for looking up the API of React Hooks). A central section titled 'What is this site?' explains the rewrite with bullet points about hooks, interactivity, and challenges. A note at the bottom states 'This beta website contains the current draft of the new docs.'

One hook that will increase performance of our app is **useMemo**.

## useMemo

`useMemo` is a React Hook that lets you cache the result of a calculation between re-renders.

```
const cachedValue = useMemo(calculateValue, dependencies)
```

Cache the results between re-renders means

When does our component re-render?

When we change our state so every time our state changes, it caches the result so that react do not have to do some expensive operation again and again.

Let us see some example

Let us create a **Demo.js** in folder components inside src

Let us setup route for it also

```
const appRouter = createBrowserRouter([
  {
    path: "/",
    element: <Body />,
    children: [
      {
        path: "/",
        element: <MainContainer />,
      },
      {
        path: "watch",
        element: <WatchPage />,
      },
      [
        {
          path: "demo",
          element: <Demo />,
        },
      ],
    ],
  },
]);
```

We need to change the state again and again so let us make a input field so that everytime we input something, our state changes.

**Remove StrictMode from index.js of react app as StrictMode calls our component twice, although in production it won't happen so we can keep StrictMode in production. It only happens in browser.**

```
src > JS index.js > ...
You, last week | 1 author (You)
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6
7 const root = ReactDOM.createRoot(document.getElementById('root'));
8 , root.render([
9   <App />
10   </React.StrictMode>
11 ]);
12
13 // If you want to start measuring performance in your app, pass a f
14 // to log results (for example: reportWebVitals(console.log))
15 // or send to an analytics endpoint. Learn more: https://bit.ly/CRA
16 reportWebVitals();
17
```

Let us see demo.js where our component renders again and again everytime we give input, now everytime our state changes, Let say we are some heavy operations inside our demo.js

so everytime it re-renders that heavy operation happens which can decrease performance of our app so we use useMemo hook

Let us say we calculate a prime number

We make prime number calculation function in helper.js

```
src > utils > JS helper.js > ...
189     result += characters.charAt(Math.floor(counter / 2));
190     counter += 1;
191 }
192 return result;
193 }
194
195 export const findPrime = (num) => {
196     let i,
197         primes = [2, 3],
198         n = 5;
199     const isPrime = (n) => {
200         let i = 1,
201             p = primes[i],
202             limit = Math.ceil(Math.sqrt(n));
203         while (p <= limit) {
204             if (n % p === 0) {
205                 return false;
206             }
207             i += 1;
208             p = primes[i];
209         }
210         return true;
211     };
212     for (i = 2; i < num; i += 1) {
213         while (!isPrime(n)) {
214             n += 2;
215         }
216         primes.push(n);
217         n += 2;
218     }
219     return primes[num - 1];
220 }
```

```

const Demo = () => {
  const [text, setText] = useState("");
  console.log("Rendering..."); // heavy operation
  const prime = findPrime(123);
  return [
    <div className="m-4 p-2 w-96 h-96 border border-black">
      <div>
        <input
          className="border border-black w-72 px-2"
          type="text"
          value={text}
          onChange={(e) => setText(e.target.value)}>
      </div>
      <div>
        <h1>nth Prime : {prime}</h1>
      </div>
    </div>
  ];
};

export default Demo;

```

Let us now, calculate prime number everytime for our input text

```
const prime = findPrime(text);
```

Let us build a dark/light theme for that demo.js and we need to make another state for it.

```

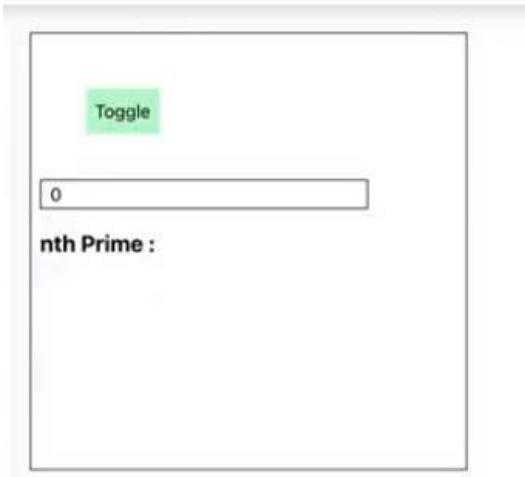
const [isDarkTheme, setIsDarkTheme] = useState(false);

<div
  className={[
    "m-4 p-2 w-96 h-96 border border-black",
    isDarkTheme ? "bg-gray-900 text-white"
  ].join(" ")}
>
```

Let us make a button to toggle this theme

```

<button
  className="m-10 p-2 bg-green-200"
  onClick={() => setIsDarkTheme(!isDarkTheme)}
>
  Toggle
</button>
```



We are calculating prime number every time our text changes but we observe we are calling PrimeCalculation function even when we are toggling our theme which is un-necessary heavy operation.

We just wanted to change the theme using toggle button why is it calling prime() function again and again.

Although in simple operations, react is very fast but if we are doing any heavy operation we run into such type of issues so we solve such issues using useMemo.

We cache the result between re-renders.

We need to memoise the result of prime()

useMemo takes first parameter as function to be cached,

useMemo caches the result of prime() so it returns a variable.

It takes dependencies as second parameter, dependencies means variable upon changing which our result will be cached, here it is "text"

```
const prime = useMemo(() => findPrime(text), [text]);
```

And we show prime variable in our div

What was the issue is, our react re-renders the component everytime our state or prop changes so we are using 2 states inside demo.js one for darkTheme other for PrimeNumber so when we are changing theme, it re-renders the whole component which makes call to prime() also which is a heavy function that freezes our browser.

So using useMemo we are memoising the heavy operations.

**Chrome allocates memory to each and every tab individually**

**useCallback hook**

## useCallback

`useCallback` is a React Hook that lets you cache a function definition between renders.

```
I  
const cachedFn = useCallback(fn, dependencies)
```

In `useMemo` we cached the result, In `useCallback` we cache the function. Rest everything is same.

## useRef hook

### useRef

`useRef` is a React Hook that lets you reference a value that's not needed for rendering.

```
const ref = useRef(initialValue)
```

#### Why do we use this hook??

If there is a case, when you want to keep some data in our component which we do not want component to re-render.

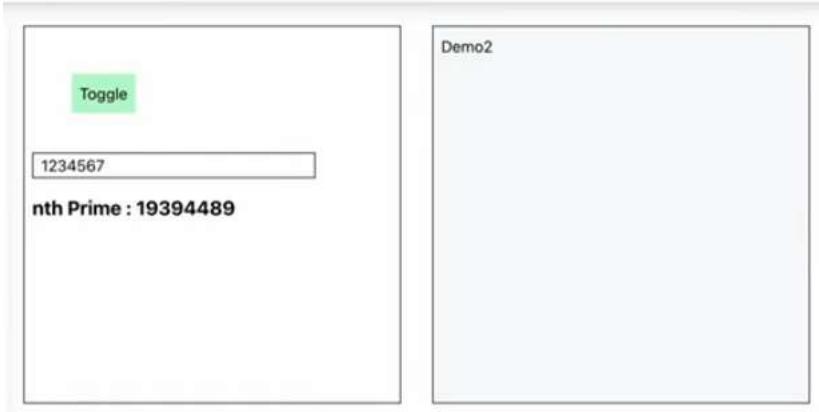
#### Can we use local variable for the same??

```
const l = 10;
```

Yes, we can but we cannot change the value without using `useState` and using `useState` will render our component which does not serve the purpose.

Let us create another `Demo2.js` and render it in the same route with `demo.js`

```
{  
  path: "demo",  
  element: (  
      
      <Demo />  
      <Demo2 />  
    </>  
  ),  
},  
  You, 1 s
```



## Why we use state variable in react if we have "let" in JS?

Suppose we have a let varibale inside demo2.js and we print that varibale inside a div

```
src > components > Js Demo2.js > ...
1  const Demo2 = () => {
2    let x = 10;
3
4    return (
5      <div className="m-4 p-2 bg-slate-50 border border-black w-96 h-96">
6        <div>
7          <h1 className="font-bold text-xl">Let = {x}</h1>
8        </div>
9      </div>
10    );
11  };
12  export default Demo2;
```

Now we want to change it so let us make a button to change this.

```
<button
  className="bg-green-100 p-2 m-4"
  onClick={() => {
    x = x + 1;
  }}
>
  Increase x
</button>
```

### Let us see if it works or not??

No, it will not because react is not able to track that variable so it do not change it in reconciliation.

When we click button, it increases but it does not re-render the page and there is no way it can get showed in the UI.

```
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
>
```

Let say we **make variable const**

```
const Demo2 = () =>  
  const x = 10;
```

It gives an **error** as we are assigning it to a constant variable

When we make it a **state variable**, it works perfectly fine.  
Now, it is changing and rendering also fine.

```
<button  
  className="bg-green-100 p-2 m-4"  
  onClick={() => {  
    setY(y + 1);  
  }}>  
  Increase Y  
</button>  
<span className="font-bold text-xl">State = {y}</span>
```

Everytime we re-render component, our value of x becomes same as initial value = 10.  
previous value of x gets deleted and whole new execution context is created and new  
memory space is created. does not matter if we make it 16 in the backend it will become 10  
on re-render of component.

**If we do not want value of x to change between re-renders so we can use useRef  
for it**

**If we increase value of x to a value and we change the state now value of x do not  
gets reset, react holds the value of x.**

```
import { useState, useRef } from "react";  
  
const Demo2 = () => {  
  const [y, setY] = useState(0);  
  let x = 0;  
  
  const ref = useRef(0);
```

ref is not just a normal variable, ref is like an object which react gives us.

ref = useRef(0) does not give us just a variable, it gives us an object.

It has a ref : { current : 0 }

### How to change value of ref??

Just like a normal variable we change it.

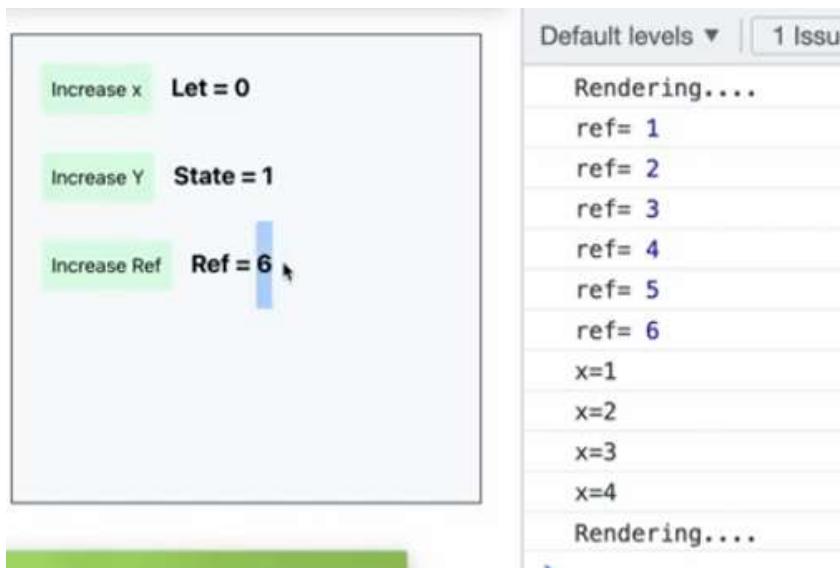
```
<button
  className="bg-green-100 p-2 m-4"
  onClick={() => [
    ref.current = ref.current + 1;
  ]}>
  Increase Ref
</button>
Ref = {ref.current}</span>
```

Our ref value is getting changed on click of button but its not reflected in the UI, but when we re-render the component, its updated value gets rendered in the UI.

Before state change, we click ref 6 times so its value is actually 6 but in UI its 0



Now we change state variable and component re-renders so new value of ref is rendered on screen which is = 6



Let us say we use useEffect to print console and we garbage collect it also inside demo2.js

```
useEffect(() => {
  const i = setInterval(() => {
    console.log("Namaste React", Math.random());
  }, 1000);

  return () => clearInterval(i);
}, [ ]);
```

Now we want to make a button so that when we click button printing stops

We need to clearInterval for i

But i has a block scope, how to access i?? Outside useEffect

```
<button
  className="bg-red-900 p-4 m-4 text-white font-bold rounded-lg"
  onClick={() => {
    clearInterval(i);
  }}
>
  Stop Printing
</button>
```

We can make it

```
let i;
useEffect(() => {
  i = setInterval(() => {
    console.log("Namaste React", Math.random());
  }, 1000);

  return () => clearInterval(i);
}, [ ]);
```

When we press Stop it works but if we render it again, things are lost.  
So we use **useRef**

```
const i = useRef(null);
useEffect(() => {
  i.current = setInterval(() => {
    console.log("Namaste React", Math.random());
  }, 1000);

  return () => clearInterval(i.current);
}, []);
```

## Session 19

**Code slow**

**Learn one thing at a time.**

**Know every word you are writing in your code.**

**While writing code, check if we can optimise it more, Is there any other way of doing it.**

**Be honest with your job, "Itne mai itna hi milega" vali approach is slavery mindset.**

**If you're in top 1% of software engineer, you will eventually earn a lot.**

**Take ownership of your work.**

**Don't attach yourself to the company.**

**Code passionately, Be one of the top performer.**

**Try to push yourself, don't think like "Yeh mera kaam nhi hain"**

