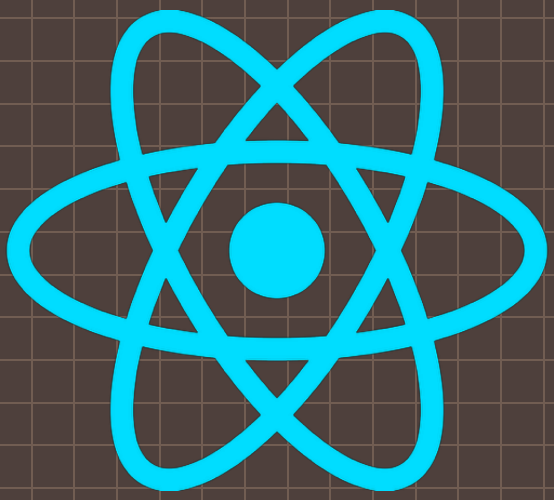
Note Taking

Namaste React



Khushi Johri Notes

Episode 1 - Inception

What is React?

React is an Open-source JavaScript library that is used for creating User Interfaces for Single Page Application (SPA). It is mainly used for handling view layer of any Web or Mobile Application.

Apps Used

Browser
- Chrome

Code Editor
- VS Code

Create an HTML file in VS Code

→ Drag-Drop a folder in VS code
→ Create a file index.html
→ Use Emmet to create a boiler plate :html5

Ways to Add React in our project

1) Using CDN links in Script Tag

CDN → Content Delivery Network, these are websites where react has been hosted and we are pulling react into our project

Why do we use CDN?
To reduce latency (Delay in network communication). It is that annoying delay you experience when trying to access a web page or video stream before it fully loads.

HW What is cross origin in this script tag?
A cross origin request is a request for a resource (eg: style sheets, iframes, images, font or script) from another domain.

Who has written this react code?
Some developers of Facebook

What happened when we got React into our Project?
It gives us alot of different important functions and methods to use in our project.

What are we using 2 CDN links?
First is react.development.js this is the core of react.

Second is the react-dom.development.js to modify the Dom (Document Object Model)

Why there are 2 files and not just 1?
React works on Other devices using React Native in Mobile Phones, React 3D and more.

1st file is the core react and 2nd file is the react-dom file which make a bridge between react and browser.

Creating Elements in React

Create Element in React

```
const heading = React.createElement("h1", {}, "Hello World")
```

Element

id: "heading" to give attribute to tag

Content to display inside Element

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

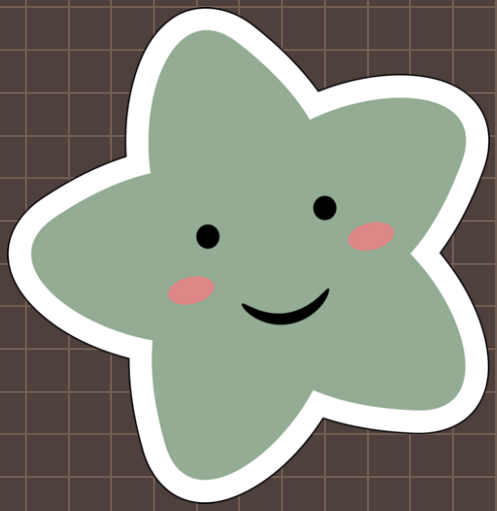
To put the element on the DOM we use React DOM library

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

React was built to manipulate DOM using Js

Most costly thing is change the dom tree on some update in the project

To optimize it React is used to do this using Js.



- Import Js in HTML
  `<script src="./App.js"></script>`
- Import CSS in HTML
  `<link rel="stylesheet" href="./index.css"/>`

HW What is createElement? What does heading return?

```
const heading = React.createElement("h1",
                {id:"heading", xyz:"abc"},
                "Hello World from React!"
                );
console.log(heading)
```

attribute

children

React Element is an object

Props are children + attributes we are passing

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

used to create the <h1> tag and display it in the root

render is used to convert object into tag

How to create nested elements in React ?

```
/*
  <div id="parent">
    <div id="child">
      <h1> I'm h1 tag</h1>
    </div>
  </div>
*/
const parent = React.createElement(
                  "div",
                  {id:"parent",
                  React.createElement(
                      "div"
                      {id:"child"},
                      React.createElement("h1",
                      {},
                      "I'm h1 tag"
                  ));
root.render(parent);
```

ReactElement (Object) ⇒ HTML (Browser Understands)

How to create sibling elements in React ?

To pass siblings we need to pass it in an array in the children

```
<h1>.... </h1>
<h2>..... </h2>
```

```
[React.createElement("h1", {}, "I'm an h1 tag"),
 React.createElement("h2", {}, "I'm an h2 tag"),
]
```

Will the order of files in HTML file matter?

It will throw an error. The order needs to be in sequence.

What happen if something is already inside

```
<div id="root">
    <h1> Khushi </h1>
</div>
```

When we try to do root.render(....) and if something is already present inside the div tag then root.render will replace the child tag inside the div.

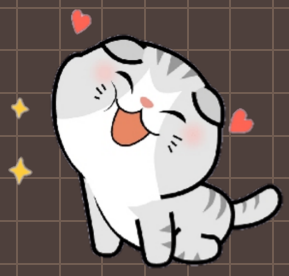
It is loaded but within few milliseconds replaced by root.render(parent)

What is the difference between a library and a framework ?

Library can work in a small part of the app too unlike a framework.

Framework need to be used in a fully fleged app to use them. They can't be used in already existing project.

But react can be used inside your existing project as well.



Episode 2 - Ignite Our App

1) Create a Repo on GitHub

2) In project add these commands

- git init
- git add README.md
- git commit -M main
- git remote add origin ....
- git push -u origin main

Can only React make our fast?
React can only make an app fast to an extend

NPM ⇒ It does not have a full form
It stands for anything but no node package manager.

It manages packages.
All the packages and utilities that we need in a project come from npm.

How to include npm in project?
In terminal
- npm init
  package name: (namaste-react)
  version: (1.0.0)
  description: This is Namaste React
  entry point: (App.js)
  test command: jest
  git repository:
  Keywords: react, namaste react, Js
  author: Khushi Johri
  license: (ISC)

package.json has been added to the project

package.json file is a configuration for our npm

packages are also known as dependencies

npm will take care of the version of the package

What important package/dependancies do we need to install?

Bundler → Our whole code needs to be bundled together, it needs to be minified, cached, compressed, cleaned before we send it to production

Webpack, parcel, vite are some Example of bundler

It packages your app so that it can be shift to production

How to install Parcel?
Run the following command:

npm install -D parcel

dev dependencies → It is generally requires during development

normal dependancies → It is used production also.

What is package-lock.json?
It keeps the track of exact version of the dependency

Package.json keeps approx version & package-lock.js keeps exact version.

What is node-modules?
All the code we fetch from npm. It's like database for our dependency.
Dependencies required for a dependency is called Transitive dependencies. Every dependency will have package.json with its own dev & normal dependencies

Should I put all these node modules to git, production?

No, if I have package.json & package-lock.json
It can regenerate our node modules

What is .gitignore?

When we don't want anything to not go on github or production, we use .gitignore

Go to .gitignore
and type `/node_modules`

Should I put package-lock.json & package.json on git?

Yes, maintains the note of what all dependencies our project needs

Ignite the app

npx parcel index.html

parcel has created a server
npm → command of npm
npx → executing a package

Another way of getting react in app

① npm
② CDN links → not prefered way of using react in a project

→ We can already have React in our node module we don't have to make a network called externally through CDN

→ Using CDN will not update it to the newest version automatically

In terminal,
npm install react / npm i react
npm install react-dom / npm i react-dom

How to use react in file after modules installation?

```
import React from "react"
import ReactDOM from "react-dom"
```

Error:
Browser Scripts can't have imports and Exports

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

↓ Browser treats it like normal Js file

Normal Js files can't have imports & exports

We have to tell the browser that its a module

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

Now react is coming from module not from CDN links

```
import ReactDOM from "react-dom/client"
```
To remove warning

What does Parcel do?

- Dev Build
- Local Server
- HMR → Hot Module Replacement
- File Watching Algo → written in C++ (To keep a track of every change)
- Caching → Faster Builds present in .parcel-cache
- Image Optimization
- Minification
- Bundling
- Compress files
- Consistent Hashing
- Code Splitting
- Differential Bundling → To support older & newer browsers along with devices
- Diagnostic
- Error Handling
- HTTPs
- Tree shaking Algo → Remove unused code
- Different dev and prod. bundlers

How to create production build?

In terminal,

npx parcel build index.html

// Error

Because here we are giving entry point as index.html but in package.json entry point is ./App.js

This conflict will lead to error

→ so "main": "App.js" needs to be removed from package.js

After creating production ready code, it will put it in dist folder in form of development build

Don't need to put dist & .parcel-cache on github as they can be created again

How to make a project compatible for older versions of browsers?

By using Browsers list in node-modules

Add this in package

Eg

```
"browserlist": {
    "last 2 Chrome version",
    "last 2 Firefox version"
}
```

You can search it on browserlist website

```
"browserlist": [
    "last 2 versions"
]
```

Episode 3 - Jaying the Foundation

Create script to start build of npx parcel index.html

In package.json, in "script" write

```
"start": "parcel index.html",
"build": "parcel build index.html"
```

"start" → In Dev Mode
"build" → Production Mode

To start it:
In terminal,
→ npm run start / npm start

To build it:
In terminal
→ npm run build

Remove React code from App.js

React.createElement is an object
render() will convert Object
into HTML

What is Jsx?

It is a JavaScript syntax which is easier to create React elements.
It makes developer life easy.

** Jsx is not a part of React, is not HTML inside Js. Jsx is HTML like Syntax or XML like syntax
JS Engine will not understand Jsx cuz it understands ECMA script.

Eg Using Jsx

```
const jsxHeading = <h1> Namaste React </h1>
```

React Element — Jsx

This is not pure HTML or Pure Js. This is Jsx.

Eg Using Pure React

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

*** But still this is running on the screen.
This is done by Parcel.
Parcel transpiler all this code before it goes to JS Engine. Then JS Engine recieves the code that browsers can understands

Transpile → Converted such that browser can understand

Parcel gives the responsiblity of transpilation to babel.

Babel is a package which is compiler / transpiler of Js. It takes Jsx and convert it into the code which browsers understands. It is a library not created by Facebook.

JSX ⇒ React.createElement ⇒ ReactElement ⇒ JS Object ⇒ HTML Element (render)
Transpiled by Babel

render() will replace everything present in HTML file (if any)

If writing JSX in multiple lines then () are mandatory. To tell Babel where is JSX starting & ending.

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

//React Element
const jsxHeading = (
        <h1 className="head">
            Namaste React
        </h1>

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

In JSX, we use className instead of class, we use camelCase, we don't use hyphen '-'

What is React functional Component?
Just a JavaScript function with return some JSX

Eg
```
const HeadingComp = () => (
    <h1> Namaste</h1>

const HeadingComponent = () => {
    returning <h1> Namaste React </h1>
};
```
Both are Same

A functional component returns a React Element

What are Component Composition?
A component inside a component

Inside React Component when {} is present we can write any JavaScript expression inside it

If you put one component inside other and other inside one then it will go into infinite loop.

Attacks

If someone gets access to your. Js code and sends some malacious data which will get displayed on the screen that attack is called cross-side scripting

It can read cookies, local storage, session storage, get cookie & read data, get info about laptop

Jsx takes care of your data

If some api passes some malacious data Jsx will escape it. It prevents cross-site scripting. It sanitizes the data before running.

```
<Title />
<Title></Title>
{Title()}
```

All are same

Components can be called like this in side Jsx

React Fragments

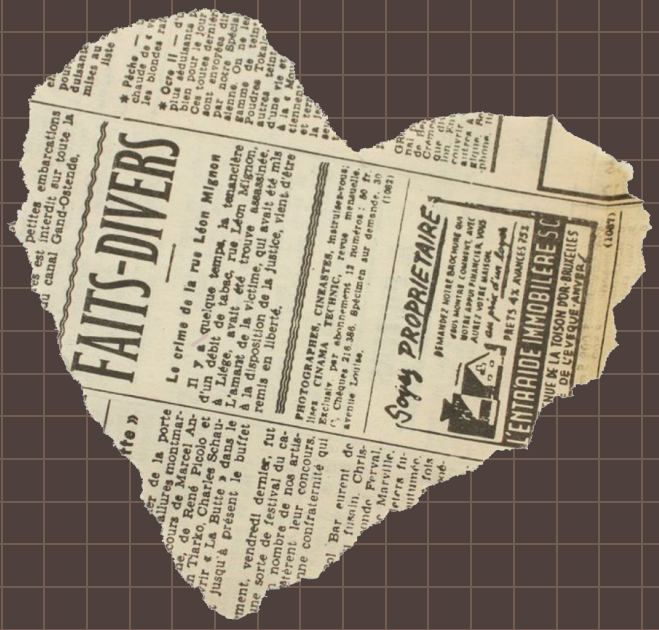
It allows you to retuen multiple elements from a React Component by allowing you to group a list of children without adding extra nodes to the DOM.

It behaves like an empty tag

```
<React.Fragment>
</React.Fragment>
```

or

```
<>
</>
```



Episode 4 - Talk is Cheap, Show me the code

Project → Food Ordering App

Planning

→ Exactly know what to build
→ User Interface

Name of your Project → Romato

Low Level Design

```
/*
  Header
    - Logo
    - Nav Items

  Body
    - Search
    - Restaurant Container
      - Restaurant Card

  Footer
    - Copyright
    - Route Links
    - Social Links
    - Address

  (Add more components
   as you like)
*/
```

Props

Short form for properties
They are used to dynamically send data to a component

Passing a prop to a function is like passing an argument to a function

Eg:-

```
Props [ <RestaurantCard
        resName = "Meghana Foods"
        cuisine = "Biryani, North Indian"
        />
```

Passing props to component

Config Driven UI
It is a user interface that is built and configured using a declarative configuration file or data structure, rather than being hardcoded.

Config is the data coming from the api which keeps on changing according to different factors like user, location etc.

To add something after each element of an array

Eg If you want to add (,) after every element

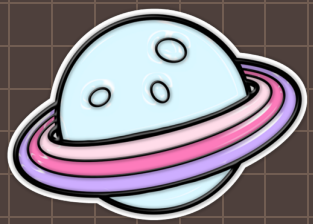
```
resData.data.cuisines.join(",")
```

Good Practices
1) Destructure Props:
   ↳ Optional Chaining

Eg:

```
const { name, avgRating } = resData?.data
```

destructuring — optional Chaining



2) Repeating ourselves (using a component again & again)

Instead use Dynamic Component Listing using JS map() function

3) Unique Key id while using map

Each item in the list must be uniquely identified

Why?

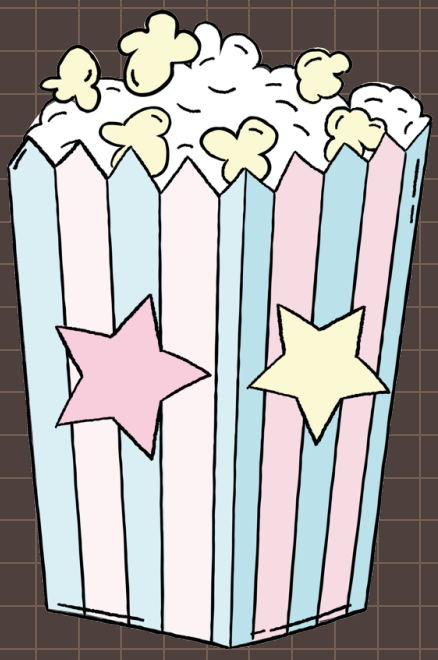
When we have components at same level and if new component comes on the first place (without id of any component)

DOM is going to re-render all the components again. As DOM can't identify where to put it

But if we give each of them a unique id then react knows where to put that component to according to id

It is a good optimization & performance thing

** Never use index as keys in map. It is not recommended.



Episode 5 - Jets Get Hooked

Best Practices

1) Keep code in seperate files
2) Don't write all the code in one file
3) All the code of a project is kept in src folder.
4) Whenever you have a hardcoded data don't keep it in the same file as a component
5) Don't keep the image src link in the component file. Create a seperate file in utils/common folder

```
const CDN_URL = "https:/....."
const LOGO_URL = "https://...."
```

Snake Case

Types of Exports

1) Default Export → In a file it can be written just once to export one main function

   Eg :- export default Button;

2) Named Export → For multiple exports Named export is used

   Eg :- export { PrimaryButton, SecondaryButton };

More Example

Default Export/Import :

```
export default component;
import Component from "path";
```

Named Export / Import :

```
export const Component
import {Component} from "path";
```

React Hooks

React Hooks is a normal javascript function which is given to us by react which comes with some superpower

useState() → Superpowerful state variable

```
const [list, setList] = useState([])
```

↑ state variable ↑ state function

Filter on Button Click

Data from api or Hardcoded Array of objects

```
const [listOfRestaurant, setListOfRestaurant] = useState([{...}, {....}])

<button onClick={() => {
                    const filteredList = listOfRestaurants.filter(
                        (res) => res.data.avgRating > 4
                    );
                    setListOfRestaurant(filteredList);
                }}
```

A normal JS variable will not update the UI but a JS state variable does. It will keep the UI in sync with the variable. It will re-render the dom very fast with that component

React Algorithm

Reconciliation Algorithm → It will be used whenever there is some change in UI

Whenever you have something in UI react create a Virtual UI for it.

Virtual DOM is a representation of actual DOM in the form of a JS Object

The overall React Element gets converted into an Obj which is the Virtual DOM

Diff Algorithm → It finds out the difference between 2 Virtual doms - The updated Virtual DOM and the previous Virtual Dom Then re-renders it & update the actual dom

Incremental Rendering
The ability to split rendering work into chunks & spread it out over multiple frames

Why React is fast?
Because it is doing alot of efficient dom manipulation because it has a virtual dom. It can efficiently find the difference between 2 virtual dom and update only the changed portion in the actual dom using Diff and Reconciliation Algorithm.

Other ways of Writing useState Hook

```
Const arr = useState(reslist)

Const [listOfRestaurants, setListOfRestaurant] =arr

// This is array destructuring
```

Or

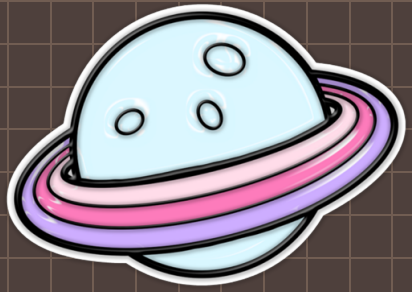
```
Const arr = usestate(reslist);

Const listOfRestaurants = arr[0];
Const setlistOfRestaurants = arr[1];
```

One way data binding
The data can come from or update from one place but not from the other way

Eg Can send a value to input from code but can't update the input from UI when using a Js variable instead of state variable

What is a hook?
Hook is a normal function. Like useState hooks returns an array first element of array is state variable and second element is a state function



Search logic

// Code

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

filterData(searchText, RestaurantList);
```

Shimmer UI

// Code

```
{ Array(20).fill().map ((s, index) => {
        return (
            <div>
            </div>
        )
    }
    )
}
```

Style the Div accordingly

Episode 6 - Exploring the world

If there is any change React can quickly re-render everything in the component.

Microservices

In olden days, there used to be only one big project to handle all the operations through Apis using UI which is known as Monolith Architecture

By now, in newer startups, instead of having one big large project we have small-small different projects

This allows teams to choose best frameworks / libraries for the individual service

Example:
UI - React Js
Backend / Apis - Java
Notification - Python
Authentication - Python
Logs - Golang

Even the databases are different. All the pages are deployed at different ports but same domain name.
This is known as Microservice Architecture.

Advantages of Microservice Architecture

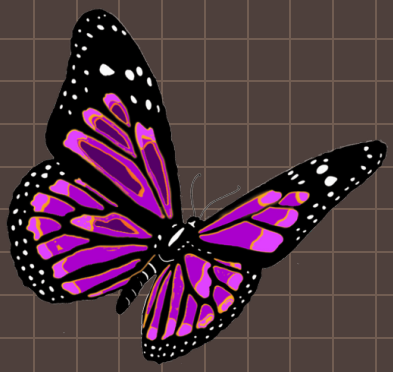
1) Easier to test
2) Develop, Deploy and scale individual component of your application independently
3) Easier to fix bugs
4) Won't crush entire app on one fault like in Monolith Architecture

To make network calls with other or outside

Fetch function is a browser function which is used to make api calls.

As in when my component loads it used to call the Api & fill the data

2 Ways of loading data



1) Loads → API calls → render page
(300 ms) (Total time = 500 ms)

2) Loads → Render UI → API calls → Update
(Total time = 100 ms)

This approach is better for User Experience

useEffect Hook → It has a callback function which will not be called immediately but when the useEffects wants it called. It is called after the component renders.
First it will render everytime the component renders or re-renders.

If you don't want to call it after every render, use a dependancy array.

[] → Dependancy array

When nothing is passed it will be called only one time

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

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

Output:

render
This is useEffect Hook

Anything inside useEffect will be called later as it is a callback function

** Browser doesn't let us talk to local host to some external api directly.
Make the api calls (api fetching) in useEffect.

** When the component is rendered for the first it uses the default data and after the api call when data has been fetched it re-renders the component with new data.

** Whenever the state variable updates, react triggers a reconciliation cycle (re-renders the component)

Optional Chaining

It allows you to access properties of an object without worrying if the object or any of its nested properties are null or undefined.
It is denoted by '?.'

It is useful while dealing with asynchronous data fetching, where you're not sure if some data is present or not

```
// without optional chaining
setRestaurants(json.data.cards[2].data.cards)

// with optional chaining
setRestaurants(json?.data?.cards[2]?.data?.cards)
```

Conditional Rendering

It is rendering a peice of code / component based on some condition.
It is like 'if else' but using Ternary Operator (Short Hand if else)

Example:

```
{
  restaurant.length === 0 ? <Shimmer /> : <div>hello</div>
}
```

Early Return

if (!allRestaurants) return null

Only javaScript expression & statement can be rendered inside {} inside a component

Episode 7 - Finding Your Path

UseEffect is a hook which is called after the component is rendered

It take a callback function & a dependancy array.

(() → { },) ⇒ If we don't have a dependancy array it will be called everytime the component renders

(() → { }, [ ]) ⇒ It will be called only once after intial render

(() → { }, [count]) ⇒ It will be called after intial render once and will be called everytime count changes

** Never create a component inside a component. As it will be called so many times.

Eg

```
const AppLayout = () → {
        const Food = () → { }
}
```

What should not be done?
1) Never use a hook inside if statement, for loop
Eg

```
if () {
  const [value, setvalue] = useState()
  }
```

React gives you a hook which let's you create local state variable inside a component = useState()

2) Never use/create useState outside a function component

Can I use more than one useEffect?
Yes, you can use as many as you want depending on different usecase

Why is CDN a great place to store image?
Because CDN is faster

Use NPM Package Formik

CDN is faster, caches your images, serves it very fast, has 100% uptime, optimize the app

React Router

```
import {createBrowserRouter, RouterProvider} from "react-router-dom"

const appRouter = createBrowserRouter([
    {
      path: "/",
      element: <AppLayout />
    },
    {
      path: "/",
      element: <About />
    }
]);

root.render(<RouterProvider router = {appRouter} />
```

configuration

React Router Dom also provides a better UI for errors in DOM.

→ Error element in Router ⇒ errorElement: <Error/>
which we write in createBrowserRouter objects

It takes a list of paths

```
const appRouter = createBrowserRouter([
                    {
                      path: "/",
                      element: <AppLayout />,
                      errorElement: <Error/>
                    }
                ]);
```

Configuration means some information which specifies what will happen in a specific route/path

→ useRouteError is a hook which react Router Dom provides to read & get the information about the type of error we are facing. It can catch error & display it.

```
import {useRouteError} from 'react-router-dom'

const Error = () => {
    const err = useRouteError();
     <h2> {err.status + ":" + err.status + text </h2>
```

Should you use <a> tag or <link> tag to navigate to a different page?
We should use <Link> tag provided by React Router Dom as <a> will reload the site as we are creating a Single Page Application.

Router provider will provide the routing configuration to our App

<Link> tag is <a> tag in the end so that browser can understand it.

Creating children in createBrowserRoute

```
const AppLayout = () => {
    return (
        <>
            <Header />
            <About />    // if path is /about
            <Body />     // if path is /
            <Contact/>   // if path is /contact
            <Footer />
        </>
    )
}

const appRoute = createBrowserRouter ([
        {
            path : "/",
            element : <AppLayout />,
            errorElement : <Error />,
            children : [
                {
                    path : "/about",
                    element : <About />,
                }
            ]
        }
    ])
```

This won't work as it will load everything together so we are going to use Outlet

Types of Routing

1) Client side Routing → All the page info are already there in the app so there is no need of network

2) Server side Routing → All the pages while loading comes from server with network call and it reloads the web pages

Outlet → According to the route, all the children will go to Outlet.
It is a named export component

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

const appRouter = createBrowserRouter ([
    {
      path: "/",
      element: <AppLayout />,
      errorElement: <Error />,
      children: [
        {
          path: "/",
          element: <Body />,
        },
        {
          path: "/about",
          element: <About />,
        },
        {
          path: "/contact",
          element: <Contact />,
        }
      ]
    }
])
```

Dynamic Routing
When there is a dynamic link the router should take us to that page according to Id

```
Eg {                                    → : means can be changed
    path: "/restaurant/:id",              This will come in useParams
    element: <Restaurant />
```

useParams → It is a hook provided by React Router Dom which gives url Id

How to read a dynamic URL params ?

```
const { id } = useParams();
```

→ Object.values convert objects into array of value

```
<Link to={"/restaurant/"+restaurant.data.id}>
</Link>
```

Episode 8 - Let's Get Classy

Earlier, Class-based components were used

Children of Children in Routing

```
children: [
    {
        path: "/about",
        element: <About />,
        children: [{
            path: "profile",
            element: <Profile />,
        }]
    }
]
```

Parent Path

Relative Path

/about/profile

No need to write /profile as it will not consider it as a children of about It will become /profile not /about/profile

"/" means from the root

To render this we need to create an outlet in the parent element means inside <About/> or we can directly import Profile Component inside parent

→ Functional Component in the end is a JS function

→ Class based Component in the end is a JS class

Class Based Components

→ Class Profile extends React component {}

extends → to tell its a class Based components

Class based components can't be created without a render method. It's mandatory

render() returns some JSX which will be injected in DOM

#

```
render() {
    return <h1> Profile Class Component </h1>
}
```

we use this.props in Class Based Components

```
constructor (props) {
    super (props);
}
```

We have to use constructor when we call/use props. While creating constructor we have to write this always. If we don't do this React will complain.

Constructor is a place where initialization happens. When the class is rendered constructor is the method to call state

Instead of useState we use this.state

```
→ this.state = {
    count: 0
  }
```

```
→ this.state = {
    count: 0,
    count2: 0
  }
```

```
→ this.state = {
    count: 0
  }

onClick {() => {
        this.setState ({
            count: 1
        })
    }}
```

```
Const [count] = useState (0);
```

```
const [count] = useState (0);
const [count2] = useState (0);
```

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

onClick {() => setCount (1) }
```

We do not mutate state directly
Never do this.state = something! In class-based component, all the states can be updated at once

Functional Component has exact states to update

If there are multiple state variables React will update only the newly updated state variable, keeping the other variables same/untouched.

React Life Cycle

Constructor is called then Render is called then componentDidMount is called.

```
componentDidMount() {
    // API Call
}
```

This method is called after render like useEffect

→ If Parent & 1 Child

Parent - constructor
Parent - render
Child - constructor
Child - render
Child - componentDidMount
Parent - componentDidMount

→ If Parent & 2 Children

Parent - constructor
Parent - render
First Child - constructor
First Child - render
Second Child - constructor
Second Child - render
First Child - componentDidMount
Second Child - componentDidMount
Parent - componentDidMount

React batches up the render phases of child components as commit phase takes time as updating/manipulating DOM is expensive

DOM is updated in a single batch to prevent multiple DOM updation

React Lifecycle Methods

There are 2 phases Render Phase and Commit Phase in Reconciliation

Render Phase → constructor()
Render()

Commit Phase → componentDidMount()

Here, React is modifying the DOM

First, React tries to batch up render phase then go to commit phase
Render Phase is very fast

In parallel, rendering of 2 children / life cycle, Render Phase is completed for both then Commit Phase starts & completes for updating the DOM

Fetching in Class components

```
constructor(props) {
        super(props) {
          this.state = {
              userInfo : {
                  name : "Dummy Name",
                  location : "Dummy Location"
              }
          }
      }
  }

async componentDidMount() {
    const data = await fetch(URL);
    const json = await data.json();

    this.setState({
        userInfo : json,
    })
}

render() {

return (
   <h2> Name: {this.state.userInfo.name} </h2>
   <h2> Location : {this.state.userInfo.location} </h2>
  )
}
```

Output

Parent - constructor
Parent - render (with default value)
Child - constructor First Child
Child - render First Child
Parent - componentDidMount
{login: 'akshaymarch7', id: '28242231'.... }
Child - componentDidMount First Child
Child - render First Child

Mounting Cycle

- Constructor (dummy data)
- Render (dummy data)
  <HTML Dummy>
- ComponentDidMount
  <API call>
  <this.setState>
  └> State variable
  └> userInfo

Updating Cycle

render (API data)
<HTML (new API data)>
- ComponentDidUpdate

Parent - componentDidMount
This is called before making API call in child.
Async componentDidMount() will be called later after parent's componentDidMount() as it is async in Commit phase it will take some data to load that's why it is taking time

Child - render First Child
As async componentDidMount() is setting the state it will call the render phase again so child render is printed

Re-render cycle is known as Updating. For updating the dom componentDidUpdate is used

Difference between ComponentDidMount and ComponentDidUpdate ?

ComponentDidMount() is called after first render and componentDidUpdate is called after every render just like dependancy array

ComponentWillUnmount() will be called when we go to other page before that this method will be called to unmount rendering process.

** Never compare lifecycle methods to React Hooks

```
componentDidMount(prevProps, prevState) {
    if (this.state.count ≠ prevState.count) {
        // code
    }

    if (this.state.count2 ≠ prevState.count2) {
        // code
    }
}
```

This is how it is written in class components

```
useEffect (() => {
    // code
}, [count , count2])
```

This is how it is written in React Hooks

Unmounting in Functional Component

```
useEffect (() => {
    setInterval (() => {
        console.log ("Namaste React");
    }, 1000)

return () => {
    console.log ("useEffect Return");
    clearInterval (timer)
}, []);
```

Unmouting means Removing from the page.

Episode 9 - Optimizing Our App

Why should we build hooks or use Single Responsibility Principal ?

→ Modularity
→ Reusability
→ Readability
→ Maintainability
→ Testable
→ Easy to Debug

→ Single Responsibility : Every function or class in React should have single responsibility. It is easy to test bugs

Components are functions & we can wrap up a small logic inside a function & use it anywhere. To keep reusable functions we can create a folder utils & import it from there

Modularity → we have broken down code into meaningful pieces to make it reusable & testable

How to create a Custom Hook ?

Use 'use' before the Hook name to identify it easily as Hooks.

→ Using Custom Hooks

```
const restaurant = useRestaurant (resId);
```

Hook is a utility function or helper functions

→ Create Custom Hooks

```
const useRestaurant = (resId) => {
        const [restaurant, setRestaurant] = useState(null)

    // Get data from API
        useEffect (() => {
            getRestaurant ();
        }, [])

    ... Rest of the code

    return restaurant;
```

A functional component is a function that returns JSX

Custom Hooks has its own reconciliation going on then it will update the DOM automatically

A normal function can't have reconciliation, it can't create a state variable that's why we use a Custom Hook

→ Creating a Custom Hook "Are you online or not"

```
const useOnline = () => {
    Const [isOnline, setIsOnline] = useState (true);

    useEffect (() => {
        window.addEventListener ("online", () => {
            setIsOnline (true);
        });

        window.addEventListener ("offline", () => {
            setIsOnline (false);
        })
    }, []);

    return isOnline;
}

export default useOnline;
```

→ Using a Custom Hook

```
const isOnline = useOffline ();

if (isOnline) {
    return <h1> Offline, please check your network </h1>
}
```

How to fake offline in browser ?
Chrome Dev Tools < Network Tab < Offline

You need to clean up event listener after calling when we go out of our event listener otherwise it will create a new eventlistener (Unmouting phase)

```
return () => {
    window. removeEventListener ()
}
```

Code

```
useEffect(() => {
    const handleOnline = () => {
        setIsOnline(true);
    }

    window.addEventListener("online", handleOnline);

    const handleOffline = () => {
        setIsOnline(false)
    }

    window.addEventListener("offline", handleOffline)

    return () => {
        window.removeEventListener("online", handleOnline);

        window.removeEventListener("offline", handleOffline);
    }
})
```

Parcel bundles up all the code and store it in 1 JavaScript file. In development, size of this file will be large but in production bundle the size will be small.

But large scale production apps cannot have all the code in 1 JS file as it will make the app slow.

What is Chunking / Code Splitting / Dynamic Bundling / Lazy Loading / On demand loading / Dynamic Import ?

Even if we have bundled our whole code in one file we can't load all of them at once as it will make the app slow so we have to split the code into smaller chunks. As only that part will load in the app while is currently being visited

For Lazy Loading a component:

```
import {lazy} from 'react';
const Instamart = lazy(() => import("./components/Instamart"));
```

It's a promise

This will not include the code of this file in the bundle. That file will load only on visiting that component

→ When load the component "on Demand" React tries to suspend it. For the first time, on visiting that component. It will suspend as it is not loaded yet due to lazy loading & react tries to render it.

To handle it we use Suspense

```
<Suspense></Suspense>
```

There will be a slight delay when the code split component is fetched, so it is important to display a loading state otherwise it will not render the page & show error

In routing,

```
{
  path: "/instamart",
  element: (
          <suspense>
              < Instamart/ >
          </suspense>
      )
}
```

→ Fallback
It is an attribute of suspense tag which will load a component until the lazy loads completes

```
<Suspense fallback = { <shimmer/> } >
      < Instamart/ >
</suspense >
```

Is chunking necessary for smaller apps?
No, it is important for big apps

Never ever lazy load inside a component because it will be lazy loaded after every render

Episode 10 - Joh Dikhta Hai Wahi Bikta Hai

Tailwind CSS

Why should we use styling frameworks in our project?
- To write optimized CSS
- It saves time
- Consistent UI
- Reusability
- Takes care of responsiveness

Ways of writing CSS
- Normal CSS in CSS file
- SCSS → Modern ways of writing CSS — not used in company ready Apps
- Inline CSS
- Using Libraries → Material UI, Ant, Base UI, Chakra UI
- Styled Components
- Tailwind CSS] Recommended ways in Companies
- Bootstrap CSS]

```
style = {{ }}
```
{{ }} → This is for a JS object
{{ }} → This is for style to tell JSX is coming inside

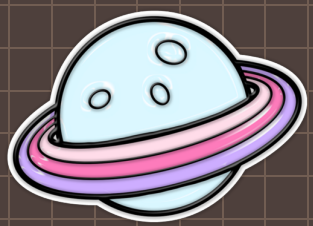
Tailwind CSS is a CSS framework

Can you use 2 styling libraries in a project?
Can be used but not preferred. It's not a good practice. As it will be inconsistent.

Why do companies focus on style & UI of app?
Joh dikhta hai woh bikta hai

What are the cons of using styling libraries & framework?
1) Makes our bundles size heavy
2) Lose control over the designing, have to use only that which is in that framework or library
3) Some styling doesn't work or work abnormally in different devices like older phones

They restrict you in some way



Tailwind CSS

It is an open source CSS framework. It change the default behavior of many tags

Features :-

1) CSS on the go (in the same file), don't have to toggle b/w files
2) Reusability → comes with a lot of pre-built classes
3) Less Bundles size → Minimal CSS
4) Flexible UI → Customizable easily

How to inject Tailwind CSS in a project ?

1) CDN Links → Tailwindcss is a package which can be installed directly

In tailwind, we have to add classname to the tags for styling

For bolding text → font-bold
For increasing size → x1, 2x1, 3x1
For text small → sm
For text medium → md
For text large → lg

There are many more which can be seen on the Tailwind website

2) Install Tailwind CSS package via parcel in terminal and Install postCSS

What is postCSS ?

- A tool transforming CSS into JS
- Compilation of Tailwind in project
- We need to tell parcel that we are using Tailwind CSS in which classes needs to be converted into CSS so that browser can understand it.

Code
→ npx tailwindcss init
It will create a config file for tailwind

Content → what files will be used in tailwind

Code
→ .postcssrc
To tell bundler compile our CSS

CSS file will be empty when we are using Tailwind CSS

Code

```
@ tailwind base;
@ tailwind components;
@ tailwind utilities;
```

It is like import inside CSS

Import Tailwind CSS plugin in VS Code

Tailwind CSS IntelliSence

If there is no suggestion press:
Ctrl + Spacebar

To get exact values

" w-[200px] "

In tailwind CSS, it will only included those CSS classes in production which we have used in project

Media Query in Tailwind CSS

sm: bg-pink-50 → Small devices
md: → Medium devices

What are the pros of Tailwind

1. Easy to debug
2. Less Code
3. No duplicate CSS
4. Bundle size is small
5. Faster Development
6. Gives more control - More Customizable
7. Saves alot of time
8. Very light weight

Sync in code when there are multiple developers

What are the cons of Tailwind CSS ?

1. To much classes
2. Initial high learning curve
3. It makes the classnames looks ugly. Makes code little less readable.

Episode 11 - Data is the New Oil

Layers of Web App :-

1. UI layers → Everything you see on the screen (the view)
2. Data layers → Maintained by states & props

JSX → JS → Object → Reconciliation
via Babel — visible in Virtual DOM

The updation b/w previous Virtual Dom & new Virtual DOM is diff

Difference b/w State & props ?

State is a local variable in a component props is the value passed from one component to other component.

⇒ Props drilling

```
App Layout (state = user)
  - <Body user = {user} />
  - <RestaurantContainer user = {user} />
    - <RestaurantCard user = {user} />
      - <h4> {user} </h4>
```

Sending data from higher heirarchy to down the chain to children
But the data flows one way

How to send data from child to parent components ?

1. Custom Hooks
2. Redux

Is prop drilling a good way ?

1. No, it makes the code cluttered as we pass the data down the line but sometimes in some components we don't even need that information but needs to be passed to the child below

2. It has unnecessary variables which will be rendered again & again, it makes it difficult to debug in React dev tools

-> Lifting the State up

We can't change / control a sibling component so we have to lift the state to the parent so that all the children can be controlled through parent component

React Devtools

Profiler < Flamegraph

You can rank the components based on their loading time

React content

If you want a data all across your app, then props drilling is not a good way instead you should use Context Api on Redux, MobX, NGRX, Flux

Why you should not make a global variable to pass it in any component ?

Because React will not track its state. No reconciliation

How to create context?

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

Context is a function at the end of the day.

Difference between useState/props and context

useState / props are tied to a component but context is not. Context can be stored in a central store which is not specific to any component. Context is like useState for whole app

Context in Class Component

```
<UserContext.Consumer>
  {({user}) => (
      <h4> {user.name} - {user.email} </h4>
  )}
</UserContext.Consumer>
```

How to override default value of context

```
const [user, setUser] = useState ( )
<UserContext.Provider
      value = {{ user : user }}
>
  <Header />
  <Outlet />
  <Footer />
</UserContext.Provider>
```

When creating a context you can give names to debug it in React DevTool?

Even when the code is not there (Lazy Loading). The Virtual Dom will be updated & the data will be displayed on the DOM.

Episode 12 - Let's Build Our Store

Managing the data efficiently

When the app is small you don't have to manage the data that efficiently its not crucial but for large production apps you need to manage the data its necessary.

Why do we need Redux ?

We need Redux to manage & handle data layer. We need context to avoid prop drilling. Instead of creating many contexts, we can create one store of Redux.

Cons of Redux

It is hard to set up.
It has a huge learning curve

But now it has come up with Redux Toolkit which is much more easier than Redux.

Difference between Redux & Redux Toolkit

Problems with Redux that Redux Toolkit resolves

1. Configuring a Redux store is too complicated
2. I have to add a lot of packages to get Redux to do anything useful
3. Redux requires too much boiler plate code

Architecture of Redux Toolkit

What is Redux Store ?

It's like a big object where we store data which all the component can access. There will be only one store in one app we will have logic seperations in a store means slices like User Slice, Authentication Slice, Theme Slice, Cart Slice

What is a Slice ?

A slice is a smaller sections of a store when can be used for different purposes

Our components cannot directly modify the store. We have to dispatch and action

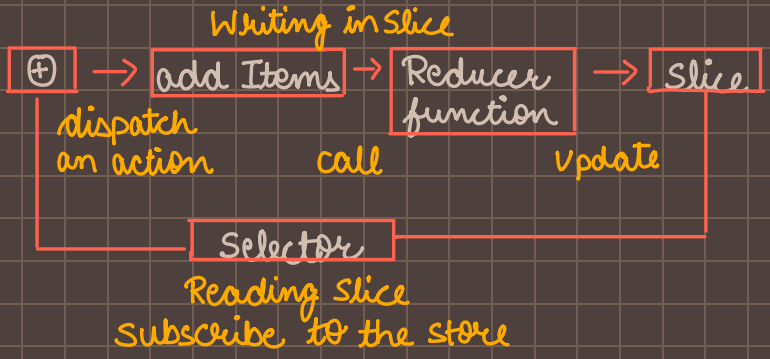
⇒ On clicking a button it will dispatch an action like addItems which will call a function. Then this function will modify the Cart Slice

** The function is reducer

Why?
We need to keep a track of everything. We don't want random components to modify our cart

+button → Dispatches action → Calls reducer function → Update Slice of Redux store

How to read the Slice?
Selector is to be called in order to read the slice



What is Selector?
Selecting the portion of that Slice. It is used to read the data of Slice. Selector is a hook which is a function at the end of the day.

Subscribe to the Store
Means reading from the store. In sync with the store when store will modify the UI will automatically modify

Redux in Project
Install Redux toolkit in project

→ npm i @reduxjs/toolkit Core of redux

→ npm i react-redux Bridge between React & Redux

Creating Store
In Store.js

```
import {configureStore} from "@reduxjs/toolkit"

const store = configureStore({})

export default store
```

Providing store to whole app

```
import {Provider} from "react-redux"
import store from "./utils/store";

return <Provider store={store}>
          <App/>
       </Provider>
```

To provider store to our app. It is a bridge between react and redux so it comes from react-redux

Actions are like small apis to communicate with redux store for add, delete, update states

Creating a Slices

In cartSlice.js,

```
import {createSlice} from "@reduxjs/toolkit";

const cartSlice = createSlice({
    name: "cart",
    initialState: {
        items: ["Banana", "Apple"]
    },
    reducers: {
        addItem: (state, action) => {
            state.items.push(action.payload);
        },
        removeItem: (state, action) => {
            state.items.pop();
        },
        clearCart: (state) => {
            state.items = [];
        }
    }
})

export const {addItems, removeItem, clearCart} = cartSlice.actions;

export default cartSlice.reducer;
```

```
cartSlice = {
    actions: {addItems, removeItems, clearCart},
    reducer: reducer
}
```

Adding Slice to the store

```
import {configureStore} from "@reduxjs/toolkit"
import cartSlice from "./cartslice";

const store = configureStore ({
        reducer: {
                cart : cartSlice
            }
    })

export default store;
```

Note Revision

-> Create Store
- configureStore () - RTC

-> Provide my store to app
- <Provider store = {store}> - import from react-redux

-> Slice
- RTK - createSlice ({
        name : " "
        initialState : ,
        reducers: {
            addItems : (state, action) => {
                    state = action.payload;
                }
        }
    })

export const {addItem} = cartSlice.actions;
export default cartSlice.reducer;

-> Put that Slice into Store
- {
    reducer : {
            cart : cartSlice,
            user : userSlice
        }
    }

Subscribe to Slice

```
import {useSelector} from "react-redux"

const cartItems = useSelector ( store.cart.items);
```

Update the cart on click

```
import {addItems} from "./utils/cartSlice"
import {useDispatch} from "react-redux"

const dispatch = useDispatch()

const handleAddItem = () => {
      dispatch(addItem("Grapes"));
    };

<button onClick={() => handleAddItem()}>
    Add to Cart
</button>
```

→ Steps
- Install @reduxjs/toolkit and react-redux
- Build our store
- Connect our store to our app
- Slice (cartSlice)
- Dispatch (action)
- Selector

Episode 13 - Time for the Test

It is going to cover how to test your app & write test cases for it

Q Why do we need test cases?
We write test cases to prevent new features to break existing code

Test driven development
We write tests even before we write code. It is good but it makes development very slow

Different type of Testing:

1) Manual Testing → Human testing done on each component

2) Automated Testing → Testing done through softwares like Selenium to automate the testing process

Testing done by Developer

3) E2E Testing → Flow way testing, covers entire user Journey (Q/A Team)
Headless Browser - Kind of like an actual browser without UI
Replacing the manual testing with code

Testing the entire flow when a user lands on a website till it leaves the website. It will test all the actions user is doing after landing the website until it leaves in one go.

Tools - Cypress, Puppeteer, Selenium

4) Unit Testing → Core job of developers. Testing components in Isolation or testing just a part of a large app. Testing 1 unit or component or a small part of app. Eg- Header

5) Integration Testing → Testing integration between components. Where many components are involved
Eg: Search Filter

Libraries / Tools to Write Test Cases

1) **React Testing Library -**
This is built on top of DOM Testing Library. React Testing Library adds more React features in DOM Testing Library.
DOM Testing Library is a base library for all others Testing Libraries like React Testing Library, Native Testing Library, Angular Testing Library etc.

Create React App already provides testing file (RTL) in the app.
RTL uses Jest. It is a JavaScript Testing Framework. Jest uses Babel

Setting up Testing in app:-

→ **Install React Testing Library**
```
npm i -D @testing-library/react
```

→ **Install Jest**
```
npm i -D jest
```

Jest used with Babel needs additional dependencies

→ **Using Babel, Install dependencies (Jest Website)**
```
npm install --save-dev babel-jest @babel/core @babel/preset-env
```

→ **Config Babel**
In babel.config.js,

```
// Code
  module.exports = {
      presets: [[`@babel/preset-env`, { target: { node: `current` } }]],
  };
```

Parcel's Babel Configuration clashing with new Babel configuration. How to fix it?
Parcel is using Babel internally and has its own configuration but to use jest we have created a new Babel configuration. This will create a conflict that which babel configuration should be used.

Parcel gives an option to disable its in-built Babel Configuration.

→ How to disable Babel Configuration of Parcel:

1) Create .parcelrc file

2) Write the following code in that file

```
{
  "extends": "@parcel/config-default",
  "transformers": {
      "*.{js,mjs,jsx,cjs,ts,tsx}": [
        "@parcel/transformer-js",
        "@parcel/transformer-react-refresh-wrap"
      ]
    }
}
```

Or

Copy the code from parceljs.org → JavaScript → Babel → Usage with other tools

Now it will use Babel Config of babel.config.js file

→ Run Test Cases

```
npm run test
```

→ Test Configuration

```
npx jest --init
```

No
jsdom (browser-like)
Yes
Babel
Yes

Now jest.config.js will be formed

When we run test cases there is no browser running they need a runtime so we use jsdom. It is a library that provides the feature of a browser and helps to run test cases.

Js DOM Installation

If Test's version is 28 or higher then js-environment-jsdom must be installed separately

// Code

```
npm install --save-dev jest-environment-jsdom
```

✯ Basic Testing

→ Install VScode-icons extension

→ Create a folder in src
__tests__ [__ __ (2 underscores front & back)] → called Dunder

Or a file
1) filename.test.js (or)
2) filename.test.ts (or)
3) filename.spec.js (or)
4) filename.spect.ts

Eg:-
In sum.js,

```
//Code
export const sum = (a,b) => {
    return a+b;
}
```

In __tests__ → sum.test.js,

```
//Code
import {sum} from "../sum";

test("Sum function should calculate the sum of two numbers", () => {
    const result = sum(3,4);

    //Assertion
    expect(result).toBe(7);
});
```

Run command in terminal to test:
npm run test

✯ React Testing (Unit Testing)

```
//Code
import {render, screen} from "@testing-library/react";
import Contact from "../Contact";
import "@testing-library/jest-dom";

test("Should load contact us component", () => {
    render(<Contact />);

    //Querying
    const heading = screen.getByRole("heading");

    //Assertion
    expect(heading).toBeInTheDocument();
});
```

→ render() in test — to load a UI component on JS dom. To test a UI we have to load it on Jsdom. This method comes from react testing library

→ screen in test - to get the information about the Rendered UI. It give access to different methods to find specific elements, or information.

→ expect() in test - to check if the passed value is present or not. It give access of different methods to check where that value is present. But we need an external library. Eg. InTheDocument using toBeInTheDocument() method.

Errors :-

1) Syntax Error: Jsx

We are passing Jsx but it is not enabled in our test cases

→ To enable js, we have to install @babel/present-react

// Code

```
npm i -D babel/preset-react
```

→ Include it in Babel.config.js as well

```
module.exports = {
    presets: [
        ["@babel/preset-env", {targets: {node: "current"}}],
        ["@babel/preset-react", {runtime: "automatic"}],
    ],
};
```

↓

Not adding this might throw errors as it will take some default value

Babel/present-react is helping our testing library to convert the Jsx code to HTML so that it can read properly

2) TypeError: toBeInTheDocument is not a function

→ Install @testing-library/jest-dom

```
npm install -D @testing-library/jest-dom
```

It will give alot of different methods along with expect method in test cases

☆ More React Examples

1) Button:

```
test("Should load button in contact component", () => {
    render(<Contact />);
    const button = screen.getByRole("button");
    expect(button).toBeInTheDocument();
});
```

2) Through text:

```
test("Should load text in contact component", () => {
    render(<Contact/>);
    const buttonName = screen.getByText("Submit");
    expect(buttonName).toBeInTheDocument();
    },
  );
```

3) Multiple Elements:

```
test("Should load 2 input boxes on the contact component", () => {
    render(<Contact/>);

    // Querying
    const inputBoxes = screen.getAllByRole("textBox");

    // Assertion
    expect(inputBoxes.length).toBe(2);
  },
);
```

Difference between getByRole & getAllByRole

getByRole() will expect only 1 element if multiple elements are passed then it will throw an error.

Whereas getAllByRole() will expect 1 or more elements. It will get the element in Jsx. It returns an array of objects. These objects are HTML input element. It is virtual dom react element.

Jsx → React Element → Object

☆ Grouping Test cases

Grouping similar working test cases into a single block which is "describe"

Eg

```
describe("Contact Us Page Test Cases", () => {
    test("Should load Contact Us", () => {
      render(<Contact/>);
      const buttonName = screen.getByText("Submit");
      expect(buttonName).toBeInTheDocument();
      },
    );

    test("Should load button in contact component", () => {
      render(<Contact/>);
      const button = screen.getByRole("button");
      expect(button).toBeInTheDocument();
    });
  }
);
```

→ Instead of test() we can use it(). "it" is another name for "test"

→ Add coverage folder to git-ignore

✲ Unit Testing

//Code

```
import {Provider} from 'react-redux';
import {render, screen} from "@testing-library/react";
import appStore from "../../utils/appstore";
import Header from "../Header";
import {BrowserRouter} from "react-router-dom";
import "@testing-library/jest-dom";

it("Should render Header Component with a login button", () => {
    render(
        <BrowserRouter>
          <Provider store={appStore}>
            <Header />
          </Provider>
        </BrowserRouter>
      );

    const loginButton = screen.getByRole("button", {name: "login"});

    expect(loginButton).toBeInTheDocument();
})
```

→ In Header component, useSelector is used which is imported from react-redux. It is not JSX or JavaScript so it will throw error. To prevent it we have to provider the store to Header component in test case.

→ In Header component, <Link> tag is imported from react-router-dom. It is not a part of React so it will throw error. To prevent it we have to pass <Browser Router> to Header component

→ If there are multiple buttons in the component and we only want to select a specific button or element then we can pass additional info in screen method like name of the button (element)

→ You can pass regex in screen.getByText(/..../);

→ Test case of Login Toggle Button :-

```
it("Should change Login Button to Logout on Click", () => {
    render(
      <BrowserRouter>
        <Provider store={appStore}>
          <Header/>
        </Provider>
      </Browser>
    );

    const loginButton = screen.getByRole("button", {name: "Login"});

    fireEvent.click(loginButton);        → Firing an event in testing
    const logoutButton = Screen.getByRole("button", {name: "Logout"});
    expect(logoutButton).toBeInTheDocument();
});
```

→ Test case to render a component with props :-

```
import {render, screen} from "@testing-library/react";
import RestaurantCard from "../RestaurantCard";
import MOCK_DATA from "./mocks/resCardMock.json";
import "@testing-library/jest-dom";

it("Should render RestaurantCard component with props Data", () => {
    render(<RestaurantCard resData={MOCK_DATA} />);

    const name = screen.getByText("Leon's - Burgers & Wings (Leon Grill)");

    expect(name).toBeInTheDocument();
});
```

→ MOCK_DATA = a file where the actual data passed for a single Restaurant Card is present

★ Integration Testing

→ Test cases for Search Restaurants

```
import {render, screen} from "@testing-library/react";
import {act} from "react-dom/test-utils";
import Body from "../Body";
import MOCK_DATA from "../mocks/mockResListData.json";
import {BrowserRouter} from "react-router-dom";
import "@testing-library/jest-dom";

global.fetch = jest.fn(() => {
      return Promise.resolve({
            json: () => {
                return Promise.resolve(MOCK_DATA)
            }
        });
```

```
});

it("Should Search ResList for burger text input", async () => {
    await act(async () =>
        render(
            <BrowserRouter>
                <Body />
            </BrowserRouter>
        )
    );

    const cardsBeforeSearch = Screen.getAllByTestId("resCard");

    expect(cardBeforeSearch.length).toBe(20);

    const searchBtn = screen.getByRole("button", { name: "Search" });

    const searchInput = screen.getByTestId("Search Input");

    fireEvent.change(SearchInput, { target : { value : "burger" } });
                                  ↳ updating the value of event(e) coming from browser

    fireEvent.click(searchBtn);

    const cardsAfterSearch = screen.getAllByTestId("resCard");

    expect(cardsAfterSearch.length).toBe(4);
});
```

→ Faking Fetch (Mock Fetch Function)

Fetch is a browser operation and testing is done on jsdom which has limited features of browser. Testing is not done on actual browser so we can't make a network call during testing. Therefore, we have to make a fetch like function

→ global.fetch → accessing fetch variable in global object

→ jest.fn() → to create a new function that works similar to fetch (mock fetch function)

It takes a callback function where a promise is returned. On fulfilling the promise, it will resolve & return json() where json() again returns a promise & on fulfilling it returns the data. In our case, MOCK_DATA.

Eg: Actual Fetch

```
const res = await fetch("URL")
const data = await res.json()
```

→ MOCK_DATA – Data from the actual API copied & pasted in a file to use it in mock fetch function

→ Automatic running tests (using HMR)

Add In package.json,
"watch-tests : " jest --watch"

npm run watch-test

To automatically run test on saving file

→ act() – comes from "react-dom/test-utils". It is used whenever we try to test a fetch or any state updates. act() returns a promise so we have to use async await outside. It will take a callback function which is async

→ <BrowserRouter> – Link tag is imported from react-router-dom that's why we are wrapping our component in <BrowserRouter>

→ Test 1 – Checking the number of all the cards before updating

→ Updating – Firing changeEvent on searchInput and adding the value "burger" in searchInput.

Firing click Event on search button to search the cards with updating value in input

→ Test 2 - Checking the number of cards after firing events for search

→ Test Id
1) In JSX, data-testid = "Search Input"

2) data-testid = "resCard"
It will give testid to all the cards present. When we get elements by test id it will give an array of objects of all the React Elements with that id and we can get the length of that array & check.

→ Test cases to get Top Rated Restaurants
Similar to previous one

// Code
```
it ("Should filter Top Rated Restaurants", async () => {
    await act (async () =>
        render (
            <BrowserRouter>
                <Body />
            </BrowserRouter>
        )
    );
```

```
const cardBeforeFilter = screen.getAllByTestId("resCard");

expect(cardsBeforeFilter.length).toBe(20);

const topRatedBtn = screen.getByRole("button", { name: "Top Rated Restaurants" });

fireEvent.click(topRatedBtn);

const cardsAfterFilter = screen.getAllByTestId("resCard");

expect(cardsAfterFilter.length).toBe(13);
});
```

✱ Different Methods or Helper Functions :-

1) **beforeAll()** → It will run before running all the test cases. Used for clean up, log, test something before running any test cases

Eg

```
beforeAll(() => {
    console.log("BeforeAll");
});
```

2) **beforeEach()** → It will run before running each test case. Before every test case

Eg

```
beforeEach(() => {
    console.log("Before Each");
```

Used for cleaning up before every test case

3) **AfterAll()** → It will run after completing all the test cases

Eg

```
afterAll(() => {
    console.log("After All");
});
```

4) **AfterEach()** → It will run after every test case

Eg

```
afterEach(() => {
    console.log("After Each");
});
```

☆ Coverage File

Coverage → Icon-report → index.html

Right click on index.html & click open with Live Server

It will give a coverage report where it will tell you exactly where it has tested & where it has not.