Important questions

Thursday, February 24, 2022 6:20 PM

https://api.github.com/users

==========

1	Introduction to react js
2	React feautres
3	Advantages and disadvantages
4	Version 18, add ons
5	JSX and Rules of JSX
6	Useage of createElement
7	Elements
8	Components (FC,CC)
9	Stateful and stateless
10	React JS Working
11	Virtual DOM working
12	Ways to create a react app
13	React project directory structure
14	Props, usage of props
15	States, Usage of states
16	Component composition
17	Props validation
18	React Life cycle methods 1. Mount 2. Updating 3. Unmounting 4. Error boundaries
19	Props drilling
20	Conditional rendering 1. If else 2. Switch 3. Ternary 4. logical
21	React Lists
22	React Keys
23	React Fragments 1. Why fragments? 2. < > </> 3. <Fragment> </Fragment> 4. Keyed Fragment
24	React Refs 1. Callback Ref 2. Forward Ref
25	Routing 1. V5 React router 2. V6 React router 3. Router Hook a. useLocation b. useNavigate c. useHistory d. useParams e. useRouteMatch i. withRouter ii. Navlink
26	React forms

	1. Functional based a. Uncontrolled b. Controlled 2. Class based a. Uncontrolled b. controlled
27	React Hooks 1. Rules 2. useState, useEffect, useContext 3. useReducer,useMemo, useCallback, useRef, useDebugValue, useLayoutEffect
28	Context API 1. createContext 2. Provider,Consumer, displayName
29	Higher Order components 1. withRouter
30	Pure Components 1. Shallow comparision 2. Deep comparision
31	Code Spiliting 1. React.lazy (lazy loading)
32	Axios 1. GET, POST, DELETE, PATCH 2. SOAP & REST API 3. Http interceptors 4. API, working of API 5. Custom hooks 6. slug
33	React state Management 1. MVC archtecture 2. FLUX architecture 3. REDUX 4. REACT-REDUX 5. REDUX-THUNK 6. FLUX- Action, Dispatcher, Store, View 7. Redux - Action, Reducer, Store, View a. combineReducers b. createStore c. applyMiddleware 8. MVC / FLUX / REDUX 9. Why Redux-Saga used?
34	Redux Hooks 1. useSelector( ) 2. useDispatch( )
35	Optional concepts 1. Profiler 2. Portals 3. React Moment
36	React Events 1. Event, event handler 2. Synthetic event
37	Render props
38	React styling 1. Inline 2. Using JS object/ CSS modules 3. Using stylesheet
39	React Modules 1. Import 2. export

==========

	Functional components	Class-based components	

props	props	this.props	How do we call
	Props.chidren	This.props.children	
state	useState( )	this.state( )= { }	
	Can create multiple states	Can create only one state	

Important questions:

1. What is react?
2. Features of react?

   Features of react JS
   - It uses virtual DOM
   - It uses server side rendering (first loads html then loads js)
   - It follows uni-directional data flow or data binding
   - It follows component based approach, allows reuseability
3. What are the main advantages of reactjs?
   - It increases the application performance => virtual DOM
   - It can be easily used on client as well as on server side.
   - Because of JSX code redability is increased
   - React is easy to integrate with other frameworks
   - With react writing the UI test cases become extremely easy. (can be done using JEST, Enzyme ,etc)
4. Difference between single page application and MPA?
5. Diff BW library and framework?
6. Diff BW Virtual Dom and real DOM?
7. Diff BW Functional comp and class based comp (before 16.8 version)?
8. Diff BW Functional comp and class based comp (after 16.8 version)?
9. Diff BW stateless and stateful components?
10. Diff BW props and state?
11. How do we build UI?
    - UI is built using components
12. What does JSX return? An Object
13. Virtual dom is made up of _____ elements?
14. Real dom is made up of _____ elements?
15. Define props define state?
16. Define JSX?
17. List the Rules of JSX?
18. What is the development server name which is installed during the react app installation?
19. What does the installation of react JS library adds? Third party modules, babel, webpacks, development server
20. Difference between React and ReactDOM library?
21. Why cannot the browsers read JSX?
22. Define ReactDOM.render( ) method?
23. What does the JSX return _____? Ans Object
24. What are the different ways are there in react to create the virtual DOM elements? JSX and ReactDOM.createElement( )
25. What is the Base Class in React? Component
26. Define react.component and explain what is the use of it?
27. Diff BW NPM and NPX?
28. Define Props drilling?
29. How do you overcome Props drilling?
30. Diff BW element and component?
31. Explain the React work flow? How does react work?
32. Explain how virtual DOM works?
33. How Virtual Dom is different from Real DOM?
34. Diff BW React and angular?
35. What makes virtual DOM faster?
36. Define Diffing algorithm?
37. What is reconcilation?
38. What is current stable version of React?
39. What are fragments and why are they used?
40. How many outermost elements are there in JSX expression? One
41. What is the one of the core types in React?
42. In reactjS why there is a need to capitalize the components? If you write small letter it considers it as a tag.
43. Is it possible to nest JSX elements into other JSX elements?
44. Diff bw import and export?
45. Diff bw regualr component and pure component?

Hot module- can update modify the content on the page when it is running without reloading the whole page

Props can be sent from parent to child
State cannot be sent from one component to another but it can be sent in the form of props.
States are like local variables defined for that function (for understanding only)

React.component is an abstract class

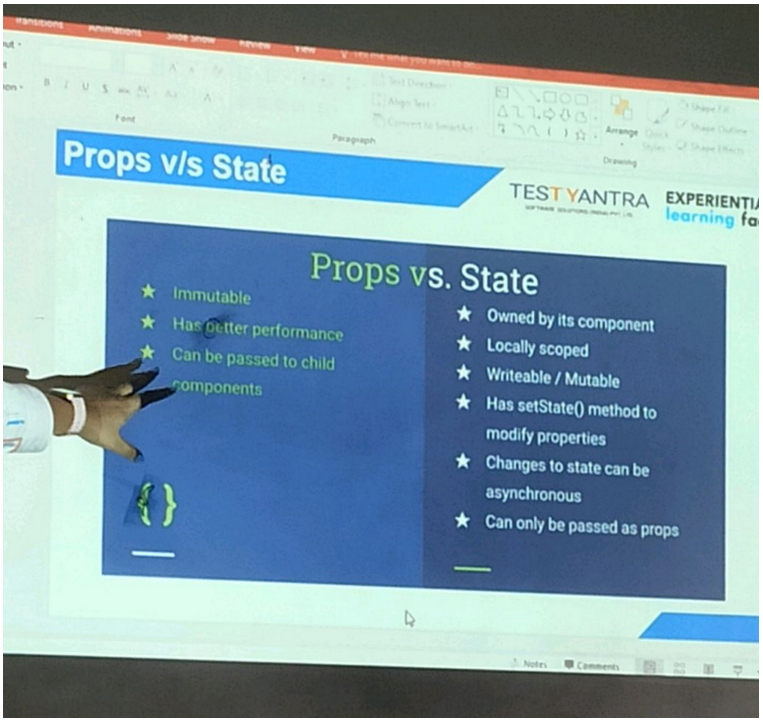
State

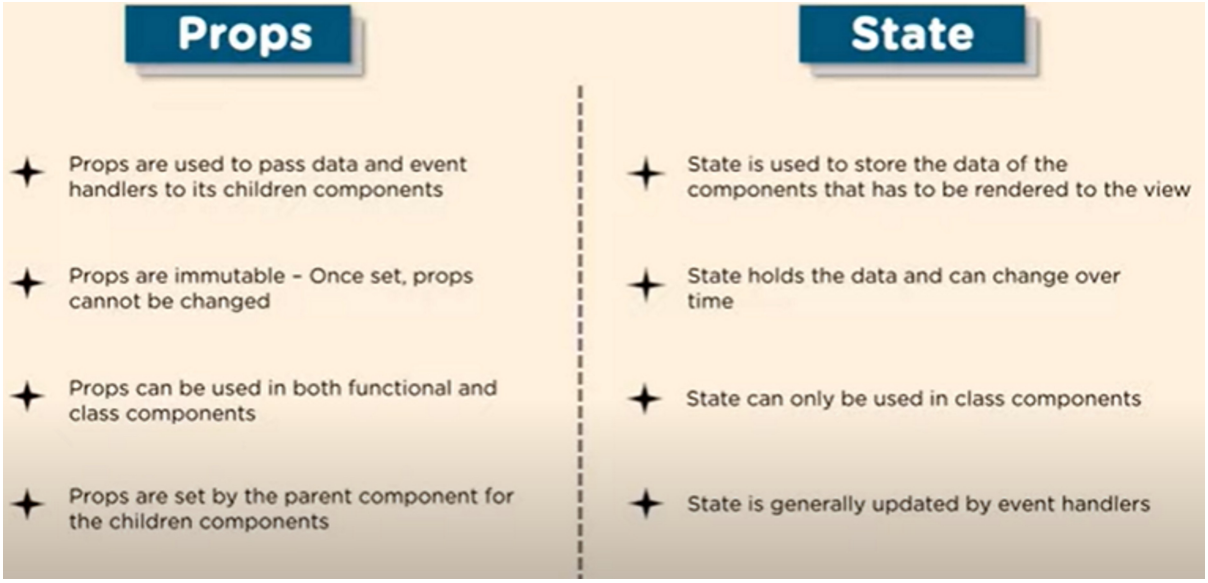
- A component needs state when some data associated with it changes over time.
- React components are built using the state objects.
- In state object:
  - ***We can store the property calues (props) that belongs to component when state object changes the component-rerenders.***
- The most important difference between state and props is that props are passed from a parent component, but state is managed by the component itself.

- A component cannot change its props, but it can change its state.

Difference bw props and state

Props	State
immutable	Owned by its component
Has better performance	Locally scoped
Can be passed to child components	Writeable/ mutable
	Has setState( ) method to modify properties
	Changes to state can be asynchronous
	Can only be passed as props





==========

MOUNTING PHASE

1. Constructor(props)
    a. Only to initialize state
    b. No business logic
    c. No sideEffects (no api calls like setimeout/setinterval)
    d. No setState( ) is used.
    e. Super(props)
        i. Initialize the parent constructor method.
        ii. It helps to inherit methods from parent (React.Component)
2. Static getDerivedStateFromProps(props, state)
    a. Static method
    b. Called after constructor( )
    c. It returns null/ returns new state
    d. It takes state as an argument and returns object with the change in the state.
    e. We can set the state of the object based on the initial props
3. Render( )
    a. It is used to write UI logic.
    b. No side effects
    c. No business logic
4. componentDidMount( )
    a. Called after the component is rendered.
    b. Can write the Business Logic
    c. Can have sideEffects
    d. When the component is mounted to the Real DOM this method will be executed.
    e. Can use the setState( )

UPDATING PHASE

1. getSnapShotBeforeUpdate( ) :
    a. It has access for props and state before the update.
    b. When using the getSnapShotBeforeUpdate( ) always need to include/call componentDidUpdate( ) or else Errors.

2. ComponetDidUpdate( ) :
    a. It called after the component is updated in the DOM.
    b. We can use this method individually.

UNMOUNTING PHASE:

==========

Fragments:

1. Adding list to the group of elements adding without extra node.
2. It helps in dealing with the issue of json components.
3. React introduced Fragments from the 16.2 and above versions.
4. Fragments allow you to group a list of children without adding extra nodes to the DOM.

5. It helps in dealing with the issue of JSON components

Use of fragments
1. It makes the execution of code faster as compared to the div tag
2. It takes less memory

==========

How to pass parameters to a route?
This hook helps us get the parameter passed on the URL without using any props object.
`useParams` hook here to access the dynamic pieces of the URL.
useParams returns an object of key/value pairs of the URL parameters. Use it to access match.params of the current <Route>

We use a hook called useParams it works like slug in express

To nagivate programmatically we use useNavigate hook

React router hooks
1. useRoutes
2. useparams
3. useLocation
4. useNavigate

React Hooks
1. useState( )
2. useEffect( )
3. useRef( )
4. useContext( )
5. useMemo( )
6. useReducer( )

==========

React Hooks are JS functions.
It's a collection of functions which is used by the React for its functional components.
Or
Hooks is a function that allows you to add the functionality for react component.

Using react hooks in functional components:
1. No state
2. No use of render
3. Use of arrow functions
4. All hooks start with keyword "use"

Usestate hook helps to manage the state in functional components

==========

What is the importance of NPX?

NPX is also a CLI application intended to enhance the experience of using the npm registry packages (npm version 5.2.0 NPXis pre-bundled with npm – tnx StefanTl23). It is simple to run any kind of Node.JS-based executable with NPX.

What is the importance of NPM?

NPM is short for Node package manager and is known to be the largest software registry in the world. It is the default Node.js package manager.

What is the prime comparision between NPX and NPM?

While If you want to make use of create-react-app in npm, the commands are npm install create-react-app then create-react-app myApp. However, in npx, you can use this command only once in the life cycle of each app, before installing it like npx create-react-app myApp.

From <https://codersera.com/blog/npx-vs-npm-a-comparison/>

==========

1. Make social media plugins
2. Upload a file and download the file using pdf, image, txt.
3. UUID unique ID
4. API fetching
5. How to use local storage in front end

6. To do app
7. Conditional rendering for login and logout
8. CRUD operations
9. Frontend form validations
10. how to find alphanumeric value from an array js
11. When will pure component render? And its hook useMemo( )?
12. How to use Error boundaries?

==========

P

Priyanka K M 5:48 PM

INTERVIEW QUESTIONS:How is React different from Angular?
What is React?
What are the features of React?
Differentiate between Real DOM and Virtual DOM.
How does the Real DOM differ from the Virtual DOM?
List some of the major advantages of React
What are the limitations of React?
Why React is used?
How React works?
Explain the Virtual DOM Working
Ø **Why can't browsers read JSX?**
Ø What is the difference between Element and Component?
When to use a Class Component over a Function Component?
What are props in React?
What is state in React?
Differentiate between states and props.
Why should we not update the state directly?
Ø What is the purpose of callback function as an argument of setState()?
Ø What is the difference between Shadow DOM and Virtual DOM?
What Makes Virtual DOM Faster?
What is the difference between createElement and cloneElement?
Ø **What are the different phases of React component's lifecycle?**
Explain the Error Boundaries
Difference Between class and Functional Components
Why do we need a Router in React?
Explain about the React LifeCycle phases
Explain the Mounting Phase with its Methods
Explain the Updating Phase with its Methods
Explain the Unmounting Phase with its Methods
At What Situations the use of setState() should be avoided
Explain the Handling Events in React JS
What is the use of Bind in Events?
Explain in Brief on Conditional Rendering
How to Render a List?
Explain the importance of Keys in React
What are Controlled Components?
What are Uncontrolled Components?
When to use the refs?
Explain the Routing Concept in React JS
How is React Router different from conventional routing?
What are Higher Order Components?
How to create components in React?
What are Pure Components?
What is the difference between HTML and React event handling?
How to bind methods or event handlers in JSX call backs?
How to pass a parameter to an event handler or callback?
What are synthetic events in React?
What is inline conditional expressions?
How to create refs?
What are forward refs?

List some of the cases when you should use Refs.
What can you do with HOC?
What is the second argument that can optionally be passed to setState() and what is its purpose?
How Virtual-DOM is more efficient than Dirty checking?
What is render() in React? Explain its purpose.
What is Flux in JavaScript?
Ø **What is the use of the arrow function in React?**
Ø **Describe how events are handled in React.**
Ø **Is setState() async? Why?**
Ø What is React Context?
Ø Explain the Context –API in brief
Ø Explain mixin or higher order components (HOC) in React.
Ø **What is the current stable version of ReactJS?**
Ø How the parent and child components exchange information?
Ø Give one basic difference between pros and state?
Ø In which lifecycle event do you make AJAX requests?
Ø How many outermost elements can be there in a JSX expression?
Ø What happens during the lifecycle of a React component?
Ø **What is one of the core types in React?**
Is it possible to display props on a parent component?
In ReactJS, why there is a need to capitalize on the components?
Is it possible to nest JSX elements into other JSX elements?
Ø What is the purpose of using super constructor with props argument?
Ø What are advantages of using React Hooks?
Ø What is useState() in React?
Ø What is StrictMode in React
Ø What is prop drilling and how can you avoid it?
Ø Do Hooks replace render props and higher-order components?
Ø How would you go about investigating slow React application rendering?
Ø How does React renderer work exactly when we call setState?
What are Fragments? When it is used
Explain the Lazy Loading.

React Redux – React Interview Questions What were the major problems with MVC framework?
Explain Flux.
What is Redux?
What are the three principles that Redux follows?
List down the components of Redux.
Ø **How is Redux different from Flux?**
Ø What are the advantages of Redux?
Ø Compare MVC with Flux?
Ø Where can Redux be used?
Ø Explain Reducers in Redux?
Ø Redux workflow features?
Ø Explain action's in Redux?
Ø What is the typical flow of data in a React + Redux app?
Ø Explain the Redux change of state?
Ø Explain Functional programming concepts?

From <https://app.slack.com/client/T03AB8H7KDH/C03A8CCJ56Z>

introduction

Monday, February 21, 2022 10:15 AM

- It is used to build a single page application
- No reload or no refresh required
- Ember.js angular.js reactjs and vuejs are alternatives for single page applications

Jordan Walke is the creator of react library
In MVC, react is used only in view

CURRENT VERSION of 17.0.2

Features of react

- It uses virtual DOM
- It uses server side rendering (first loads html then loads js)
- It follows uni-directional data flow or data binding
- It follows component based approach

Babbel is a JS compiler

Server name run on port 3000 ===> webpack Dev server

AJAX

- AJAX- asynchronous JS and XML
- AJAX was introduced in 2001
- We need AJAX to asynchronously update content without reload or refresh
- XML HTTP Request (or XHR) it means AJAX request ---> no page reload and no page refresh

XMLHttpRequest
XMLHttpRequest (XHR) objects are used to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing.

XMLHttpRequest is used heavily in AJAX programming.
The XMLHttpRequest object can be used to request data from a web server.

The XMLHttpRequest object is a developers dream, because you can:

- Update a web page without reloading the page
- Request data from a server - after the page has loaded
- Receive data from a server - after the page has loaded
- Send data to a server - in the background

AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

- Ajax uses XHTML for content, CSS for presentation, along with Document Object Model and JavaScript for dynamic content display.
- Conventional web applications transmit information to and from the sever using synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server.
- With AJAX, when you hit submit, JavaScript will make a request to the server, interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server.
- XML is commonly used as the format for receiving server data, although any format, including plain text, can be used.
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background.
- Intuitive and natural user interaction. Clicking is not required, mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven.

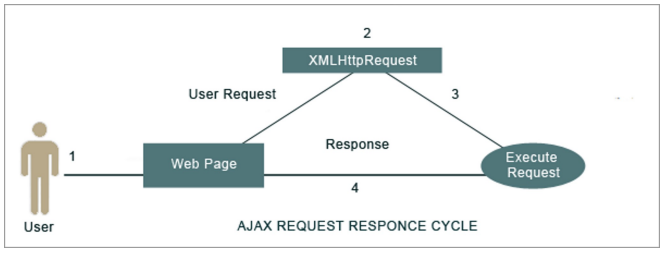
From <https://www.tutorialspoint.com/ajax/what_is_ajax.htm>

AJAX is Based on Open Standards

AJAX is based on the following open standards −

- Browser-based presentation using HTML and Cascading Style Sheets (CSS).
- Data is stored in XML format and fetched from the server.
- Behind-the-scenes data fetches using XMLHttpRequest objects in the browser.
- JavaScript to make everything happen.

From <https://www.tutorialspoint.com/ajax/what_is_ajax.htm>



From 2009
Google developers created a framework called angular.js to create SPA single page applications.
Based on MVW pattern model view whatever
2010 it became open source
Till 2013 it was at its peak
AJAX + JQUERY = asynchronous operations before angular.js

2013 a new era started facebook dev's invented reactjs
Reactjs is not a framework it is small UI library
It is component based architecture.

2015 vuejs was introduced it is community based

Angular is a huge framework it is used for enterprise huge applications

REACTJS is a JS UI library to build single page application
React can be used for server side rendering also.
By using react native we can create cross platform andriod application
Learn once execute anywhere

NPM way
React is a JS library or TS library
React is a declarative approach object and not imparative
React is component based architecture

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Declarative views make your code more predictable and easier to debug.

Component-Based

Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript instead of templates, you can easily pass rich data through your app and keep state out of the DOM.

Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

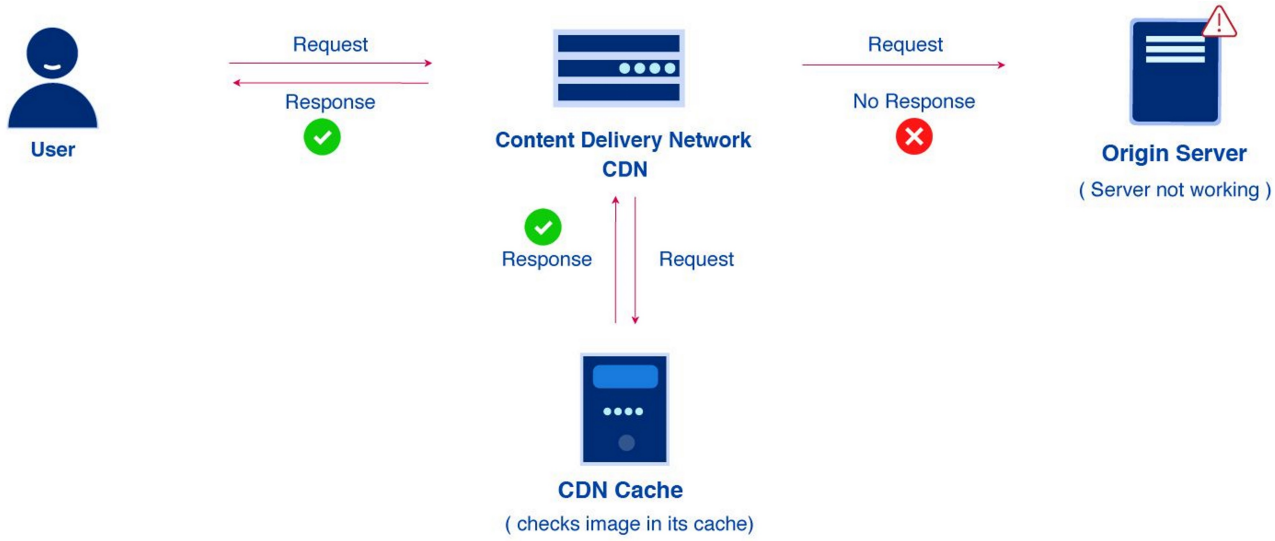
React can also render on the server using Node and power mobile apps using React Native.

Install react app

To use react we have two ways
1. CDN way content delivery network
2. Node JS way or CLI way

CDN provider
For production purpose CDN is a recommended way

A CDN is a network of servers that distributes content from an "origin" server throughout the world by caching content close to where each end user is accessing the internet via a web-enabled device. The content they request is first stored on the origin server and is then replicated and stored elsewhere as needed.



CDN providers list
1. Cloudflare · 2. CloudFront · 3. Fastly · 4. CDNetworks · 5. Incapsula · 6. Akamai · 7. StackPath.

CLI way

Npm install -g create-react-app

To check cli installed or not
Create-react-app --version

To create a new project
create-react-app simple-react

create-react-app is nothing but react CLI

Npx - node package execute
NPX: The npx stands for Node Package Execute and it comes with the npm, when you installed npm above 5.2.0 version then automatically npx will installed. It is an npm package runner that can execute any package that you want from the npm registry without even installing that package

Do not use capital names for app-name
You will get error called naming restrictions
Do not use react name only in the app name

To run react application
We need three libraries especially CLI way

1. React
2. React DOM
3. React Scripts

CDN way we need
React,
In index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <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>adding react in html</title>
    <!-- step 2 adding scripts -->
    <script
      crossorigin
      src="https://unpkg.com/react@17/umd/react.development.js"
    ></script>
    <script
      crossorigin
      src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"
    ></script>
    <!-- step 3 adding babbel script -->
    <script
      src="https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.26.0/babel.min.js"
      integrity="sha512-kp7YHLxuJDJcOzStgd6vtpxr4ZU9kjn77e6dBsivSz+pUuAuMlE2UTdKB7jjsWT84qbS8kdCWHPETnP/ctrFsA=="
      crossorigin="anonymous"
      referrerpolicy="no-referrer"
    ></script>
  </head>
  <body>
    <!-- step 1 create a container -->
```

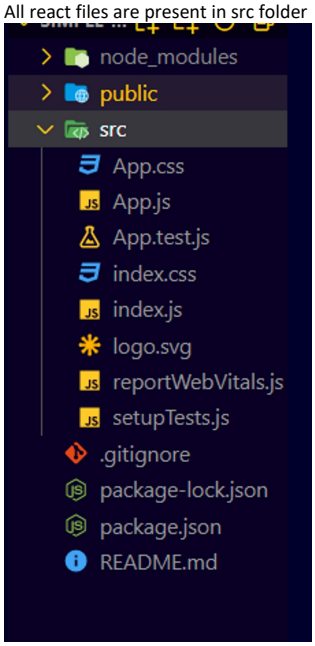
```
    <div id="root"></div>
    <script src="./index.js" type="text/jsx"></script>
  </body>
</html>
```

In index.js

```
// step 4 is adding the react element
const element = <h1>Home welcome to react</h1>;
ReactDOM.render(element,document.getElementById("root"));
```

React file structure

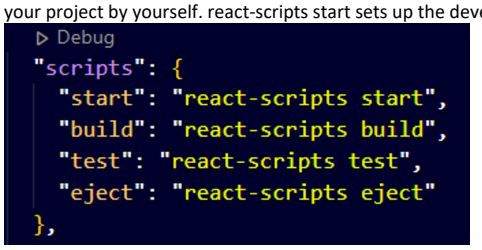
All react files are present in src folder



react-scripts

What is react-scripts?

react-scripts is a set of scripts from the create-react-app starter pack. create-react-app helps you kick off projects without configuring, so you do not have to setup your project by yourself. react-scripts start sets up the development environment and starts a server, as well as hot module reloading.



React-scripts is providing starting development server.

What is hot module reloading?

Hot Module Replacement (HMR) exchanges, adds, or removes modules while an application is running, without a full reload. This can significantly speed up development in a few ways: Retain application state which is lost during a full reload.

Hot Module Replacement (HMR) **exchanges, adds, or removes modules while an application is running, without a full reload**. This can significantly speed up development in a few ways: Retain application state which is lost during a full reload. Save valuable development time by only updating what's changed.

What is the build folder in react?

When we type npm run build , we see output in /build folder. Builds the app for production to the build folder. It correctly bundles React in production mode and optimizes the build for the best performance. The build is minified and the filenames include the hashes.

Command to generate build:- npm run build

Command to test:- npm run test

For unique testing and end to end testing

Jest is a JavaScript testing framework maintained by Facebook

npm run eject

very dangerous command do not use it unless you want to remove default config and set your own

This command will remove the single build dependency from your project. Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc.) into your project as dependencies in package

eslintConfig

ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, with the goal of making code more consistent and avoiding bugs. ESLint uses an AST (Abstract Syntax Tree) to evaluate patterns in code. ESLint is completely pluggable, every single rule is a plugin and you can add more at runtime.
This is mainly used to avoid javascript bug.

Features of React

1. Unidirectional data flow
2. Uses VDOM
3. Easy to integrate with other fraemworks
4. Uses JSX

Day2

Tuesday, February 22, 2022 10:03 AM

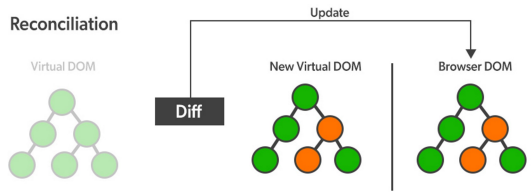
Library
React

1. Npm/yarn
2. Babel
3. Webpack
4. React DOM and react
5. React scripts
6. Virtual DOM

Why react?
It is faster
It has virtual DOM (extact clone of real DOM)
Virtual DOM memory is temprovary in nature

Virtual DOM	Real DOM

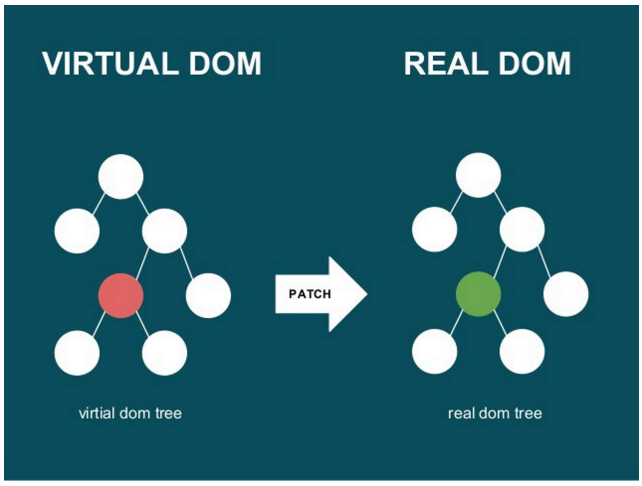
Virtual DOM
The virtual DOM (VDOM) is a programming concept where ***an ideal, or "virtual", representation of a UI is kept in memory and synced*** with the "real" DOM by a library such as ReactDOM. This process is called reconciliation. ... They may also be considered a part of "virtual DOM" implementation in React.

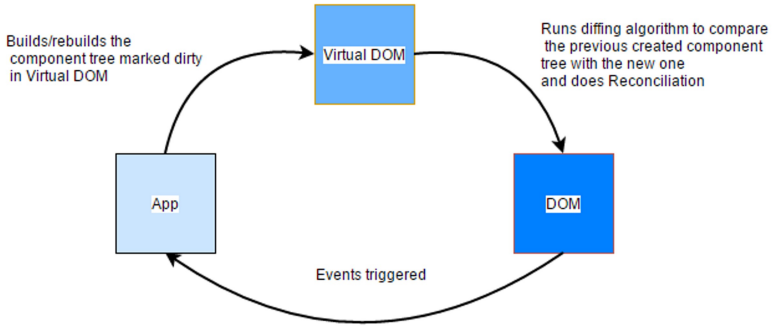


There are two phases for Virtual DOM

1. Render phase is asynchronous in nature
2. Commit phase is synchronous in nature

Reconciliation
The Diffing Algorithm. When diffing two trees, React first compares the two root elements.
The behavior is different depending on the types of the root elements.

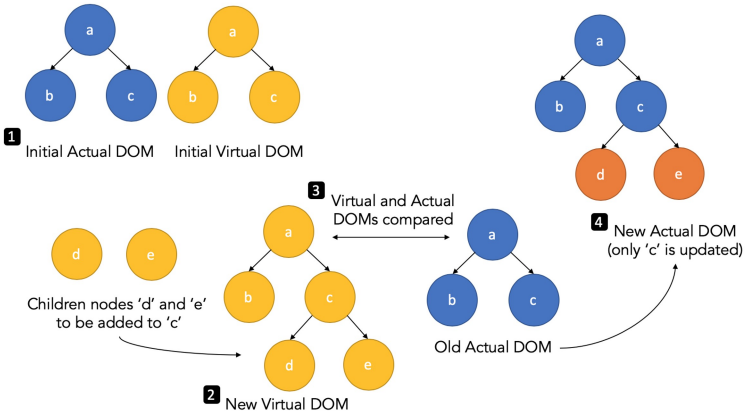




Diffing algorithm works based on keys

What is a Diffing algorithm?
Diffing is a heuristic algorithm based on two assumptions: Two elements of different types will produce different trees. The developer can hint at what elements will remain stable across renders with a key prop.
https://javascript.plainenglish.io/reacts-diffing-algorithm-1a64cfefa4e0

We write code for virtual DOM which inturn goes to diffing alogrithm matches differences and updates the Real or browser DOM



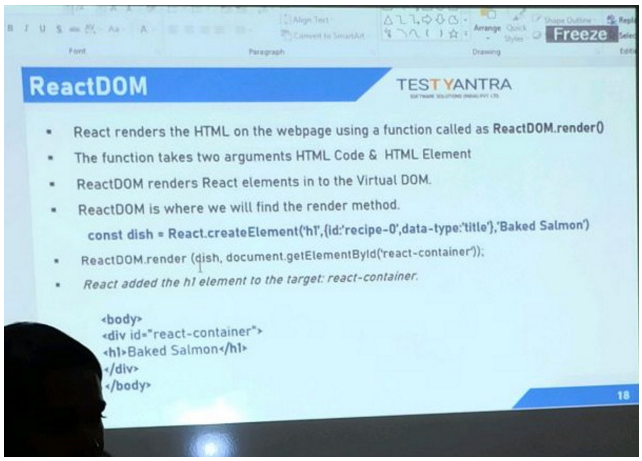

```
import ReactDOM from "react-dom";
import React from "react";
let dom = ReactDOM.render("bhuvan", document.getElementById("root"), () => {
  console.log("successfully called vDOM");
});
console.log(dom);
```

==========

React.createElement(a,b,c)
a = element you want to create or tag name
b = props value you want to pass, null or { }
c = content inside the tag

ReactDOM.render(x,y,z)
x = variable or tag name
Y = where you want to insert the new element
Z= call back function

https://learn.co/lessons/react-create-element



Best example

```
const title = react.createElement("h2", null, "what's happening?");
ReactDOM.render(title, document.getElementById("root"), () => {console.log("element added ");});
```

==========

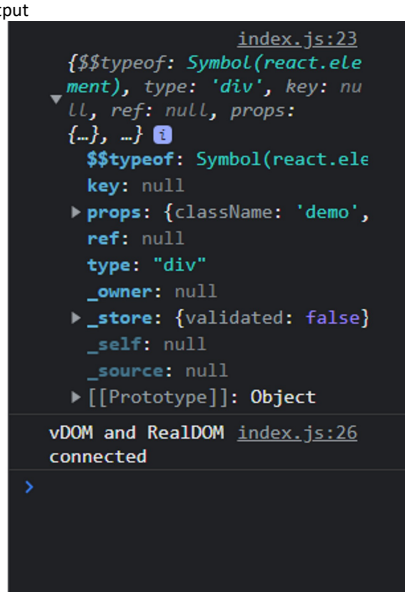
```
//how to write for VDOM using react
let element = React.createElement("section", null, "iam section element");

//how to write in real DOM
let domElement = document.createElement("section");
domElement.textContent = "iam a section element";
document.body.appendChild(domElement);
ReactDOM.render(element, document.getElementById("root"));
```

==========

```
let div = React.createElement(
  "div",
  { className: "demo", id: "demo" },
  "hello world"
);
console.log(div);
ReactDOM.render(div, document.getElementById("root"), () => {
  console.log("vDOM and RealDOM connected");
});
```

Output



Different ways of write code

DOM	React without JSX	React With JSX

```
let div = document.createElement("div");
let h1 = document.createElement("h1");
let p = document.createElement("p");
let btn =
document.createElement("button");
//set attributes to elements
div.setAttribute("id", "demo");
h1.classList.add("h1");
btn.className = "btn";
//add contents to elements
h1.textContent = "i am h1";
p.innerHTML = "I am para";
btn.innerText = "click me";
//append all elements to div
div.appendChild(h1);
div.appendChild(p);
div.appendChild(btn);
//append div to body
document.body.appendChild(div);
```


```
//react way without JSX
import ReactDOM from "react-dom";
import React from "react";
let element = React.createElement(
  "div",
  { id: "demo" },
  React.createElement("h1", { className: "h1" }, "hello i am h1"),
  React.createElement("p", null, "i am para"),
  React.createElement("button", { className: "btn" }, "click me")
);
ReactDOM.render(element, document.getElementById("root"));
```


```
//with JSX
import ReactDOM from "react-dom";
import React from "react";
let element = (
  <div id="demo">
    <h1 className="h1">i am h1</h1>
    <p>I am para</p>
    <button className="btn">Click
me</button>
  </div>
);
ReactDOM.render(element,
document.getElementById("root"));
```

Introducing JSX

Consider this variable declaration:

```
constelement =<h1>Hello, world!</h1>;
```

This funny tag syntax is neither a string nor HTML.

It is called JSX, and it is a syntax extension to JavaScript. We recommend using it with React to describe what the UI should look like. JSX may remind you of a template language, but it comes with the full power of JavaScript.

JSX produces React “elements”. We will explore rendering them to the DOM in the next section. Below, you can find the basics of JSX necessary to get you started.

From <https://reactjs.org/docs/introducing-jsx.html>

React doesn’t require using JSX, but most people find it helpful as a visual aid when working with UI inside the JavaScript code. It also allows React to show more useful error and warning messages.

JSX RULES

1. In JSX, closing tag is mandatory, regardless of paired or unpaired tag the tag must be closed
   JSX follows XML rules and therfore HTML elements must be properly closed
2. One top level element
   The html code must be wrapped inside a container always, you must not use elements at same level without a container.
   You can use fragments < > </> introduced in 16.8 react
3. Inserting a large block of HTML
   Use parentheses for large blocks of code
4. JSX expression - use one curly braces to use variables or data interpolation or evaluate javascript.
   With JSX you can write expressions inside curly braces { }.
   The expression can be a React variable, or property, or any other valid JavaScript expression. JSX will execute the expression and return the result:
   Example
   Execute the expression 5 + 5:
   const myelement = <h1>React is {5 + 5} times better with JSX</h1>;

Attribute rules

1. Use className instead of class
2. Instead of for go with htmlFor
   <label htmlFor="username">username</label>
3. Replace hypen with camel case syntax
4. Any events use camel case

How to write functions inside JSX

```
import ReactDOM from "react-dom";
import React from "react";
let languages = ["java", "js", "python"];
let element = (
  <>
    {languages.forEach(x => console.log(x))}
  </>
);
ReactDOM.render(element, document.getElementById("root"));
```

==========

Why use map in JSX over forEach()?
Map returns new array so it used for iteration in JSX
Whereas forEach returns undefined

```
import ReactDOM from "react-dom";
import React from "react";
let languages = ["java", "js", "python"];
let element = (
  <>
    {languages.map(x => (
      <li>{x}</li>
    ))}
  </>
);
ReactDOM.render(element, document.getElementById("root"));
```

==========

```
import ReactDOM from "react-dom";
import React from "react";
let courses = [
  {
    courseName: "MERNSTACK",
    trainer: "shashi",
    duration: "4mon",
    languages: ["html", "css", "js"],
  },
  {
    courseName: "MEANSTACK",
    trainer: "shashi",
    duration: "4mon",
    languages: ["html", "css", "js"],
  },
];
let element = (
  <>
    {courses.map(course => {
      let { courseName, trainer, duration, languages } = course;
      return (
        <ul>
          <li>{courseName}</li>
          <li>{trainer}</li>
          <li>{duration}</li>
          <li>
            {languages.map(c => {
              return(
              <li>{c}</li>);
            })}
          </li>
        </ul>
      );
    })}
  </>
);
ReactDOM.render(element, document.getElementById("root"));
```

- MERNSTACK
- shashi
- 4mon
- html
- css
- js

- MEANSTACK
- shashi
- 4mon
- html
- css
- js

Implicit return JSX without using return keyword

Never use { flower brackets in arrow function for impilict return

```
import ReactDOM from "react-dom";
import React from "react";
let fullstack = [
  {
    courseName: "MERNSTACK",
    trainer: "shashi",
    duration: "4mon",
    languages: ["html", "css", "js"],
  },
  {
    courseName: "MEANSTACK",
    trainer: "shashi",
    duration: "4mon",
    languages: ["html", "css", "js"],
  },
];
let element = (
  <>
    {fullstack.map(cou => (
      <ul>
        <li>{cou.courseName}</li>
        <li>{cou.trainer}</li>
        <li>{cou.duration}</li>
        <li>
          {cou.languages.map(c => (
            <li>{c}</li>
          ))}
        </li>
```

```
      </ul>
    ))}
  </>
);
ReactDOM.render(element, document.getElementById("root"));
```

For controlled component
Step1: create states
Step 2:

What is a change event in JS?
The HTML DOM onchange event occurs when the value of an element has been changed. It also works with radio buttons and checkboxes when the checked state has been changed. The JavaScript change event is an event type that gets executed when the focus on an element is changed. The change event of JavaScript inherits all the methods and properties of the Event.

Sample example

```
let input = document.querySelector("#input");
let text = document.querySelector(".text");
input.addEventListener("change", e => {
  console.log(e);
  if (e.targeet.value.length > 10) {
    e.target.style.border = "4px solid green";
    text.innerHTML = "strong password";
    text.style.color = "green";
  }
});
```

Controlled way two function way

```
//!controlled component
import React, { Component } from "react";
//create state Object
//add value attribute into the form elements
//****add onchange event to the form elements */
export default class App extends Component {
  state = {
    email: "",
    password: "",
  };
  handleSubmit = e => {
    e.preventDefault();
    console.log(this.state);
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <form onSubmit={this.handleSubmit}>
            <div>
              <label htmlFor="email">email</label>
              <input
                type="text"
                id="email"
                placeholder="email"
                vaule={this.state.email}
                onChange={e => this.setState({ email: e.target.value })}
              />
            </div>
            <div>
              <label htmlFor="password">password</label>
              <input
                type="text"
                id="password"
                placeholder="password"
                value={this.state.password}
                onChange={e => this.setState({ password: e.target.value })}
              />
            </div>
            <div>
              <button>submit</button>
            </div>
          </form>
        </article>
      </section>
    );
  }
}
```

Controlled way with single function and always wrap name attribute within array through javascript	

```
//create state Object
//add value attribute into the form elements
//****add onchange event to the form elements */
export default class App extends Component {
  state = {
    email: "",
    password: "",
  };
  handleSubmit = e => {
    e.preventDefault();
    console.log(this.state);
  };
  handleChange = e => {
    let { name, value } = e.target;
    this.setState({ [name]: value });
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <form onSubmit={this.handleSubmit}>
            <div>
              <label htmlFor="email">email</label>
              <input
                type="text"
                id="email"
                placeholder="email"
                value={this.state.email}
                name="email"
                onChange={this.handleChange}
              />
            </div>
            <div>
              <label htmlFor="password">password</label>
              <input
                type="text"
                id="password"
                placeholder="password"
                value={this.state.password}
                name="password"
                onChange={this.handleChange}
              />
            </div>
            <div>
              <button>submit</button>
            </div>
          </form>
        </article>
      </section>
    );
  }
}
```

Uncontrolled	Controlled
Refs are used to update the value through DOM	Onchange event updates state through onChange Event

components

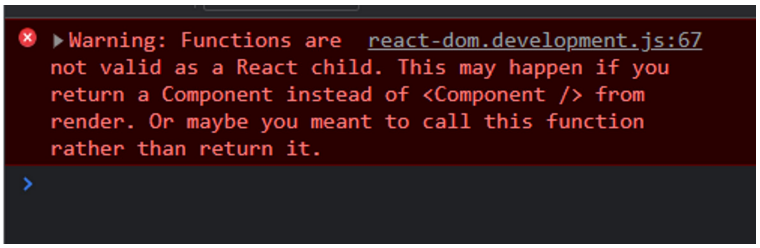
Wednesday, February 23, 2022 10:05 AM

Components are reuseable UI building blocks.

ES-6 class--> class based component

JS function-> functional component

Create a new file called App.jsx
Components are always caps
For components wrap with angular braces



CLASS BASED COMPONENT

App.jsx	Index.js

`import React from "react";`
`class App extends React.Component{`
`render() {`
`return <h1>hello world</h1>`
`}`
`}`
`export default App;`
	`import ReactDOM from "react-dom";`
	`import React from "react";`
	`import App from "./App";`
	`ReactDOM.render(<App />, document.getElementById("root"));`

FUNCTIONAL BASED COMPONENTS

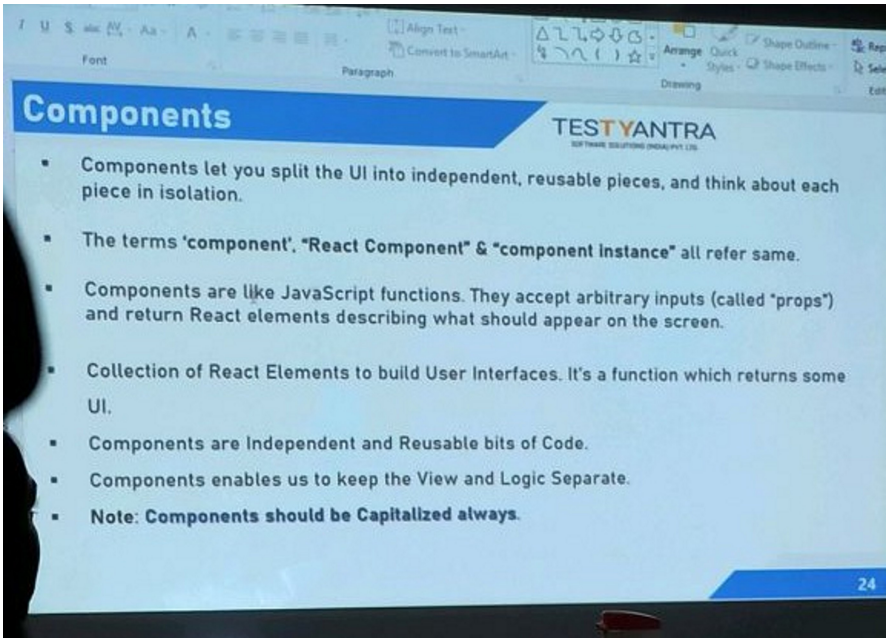
APP.jsx	Index.js

`let App = () => <h1>I am functional based</h1>`
`export default App;`
	`import ReactDOM from "react-dom";`
	`import React from "react";`
	`import App from "./App";`
	`ReactDOM.render(<App />, document.getElementById("root"));`

React 17 onwards we do not need to import react for basic JSX --> JSX transpile

COMPONENTS
Components are independent and reusabale bits of code. They serve the same purpose as Javascript functions, but work inisloation and return JSX.
Components come in two types,

- Class components
- functional components.



A Component is one of the core building blocks of React. In other words, we can say that every application you will develop in React will be made up of UI pieces called components. Components make the task of building UIs much easier.

Composition

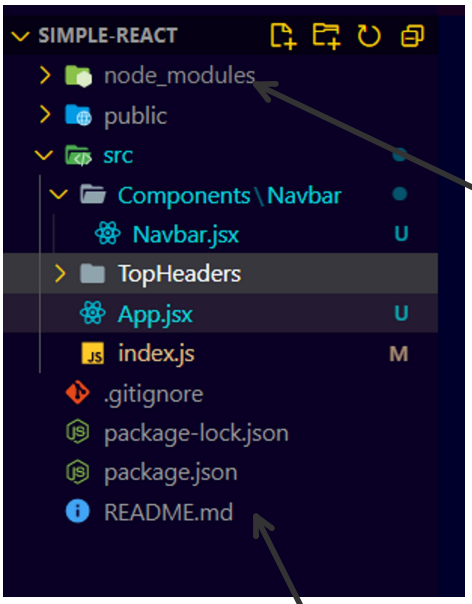
Install following extensions

1. **Import cost**
2. **Simple react snippets**
3. **Auto import**
4. **ES7 redux react**

Rcc- react class based components

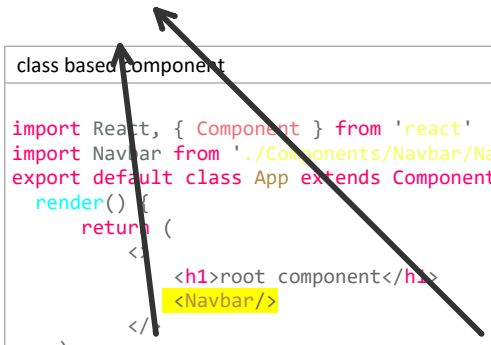
```
Rafce- react arrow functional component
```

Follow this folder structure



flow of code for components and composition	Index.js
`import ReactDOM from "react-dom";` `import React from "react";`	Root react file

```
import App from "./App";
ReactDOM.render(<App />, document.getElementById("root"));
```



class based component

App.jsx

Root component

```
import React, { Component } from 'react'
import Navbar from './Components/Navbar/Navbar'
export default class App extends Component {
  render() {
      return (
          <>
              <h1>root component</h1>
              <Navbar/>
          </>
    )
  }
}
```

Navbar.jsx

```
import React from "react";
import Logo from "./Logo";
import Menu from "./Menu";
const Navbar = () => {
  return (
    <div>
      <Logo />
      <Menu />
    </div>
  );
};
export default Navbar;
```

Menu.jsx

```
import React from "react";
const Menu = () => {
  return <div>Menu</div>;
};
export default Menu;
```

Logo.jsx

```
import React from "react";
const Logo = () => {
  return <div>Logo</div>;
};
export default Logo;
```

Now for css in the root level create global.css
And using import statement add it to index.js root file

Props

- Props are properties to transfer data from parent component to child component.
- Props are immutable, cannot be modified
- They are only read only
- They are only unidirectional from parent to child

React props

Props stand for "properties". They are read-only components. It is an object which stores the value of attributes of a tag and work similar to HTML attributes. It gives a way to pass data from one component to other components. It is similar to function arguments. Props are passed to the component in the same way as arguments passed to a function.
Props are immutable so we cannot modify the props from inside the component.

Consuming props

Function based	Class based

```jsx
import React from 'react'
const Mernstack = props => {
    console.log(props);
  return (
      <div>I love {props.courses}</div>
  )
}
export default Mernstack;
```

```jsx
import React, { Component } from 'react'
export default class Mernstack extends Component {
  render() {
    return (
        <div>Mernstack i love { this.props.courses}</div>
    )
  }
}
```

BETTER WAY TO DESTURCTURE FROM AN OBJECT by passing the parameter

```jsx
import React from "react";
const Mernstack = ({ courses }) => {
  // console.log();
  return <div>I love {courses}</div>;
};
export default Mernstack;
```

How to add array into props and consume it

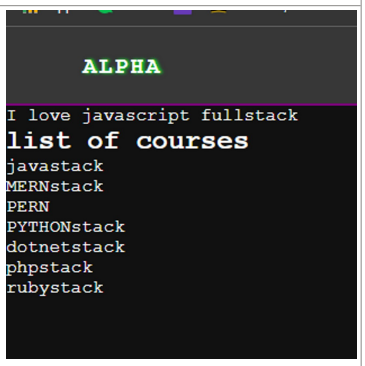
App.jsx

```jsx
import React, { Component } from "react";
import Navbar from "./Components/Navbar/Navbar";
import Fullstack from "./topics/Fullstack";
import Mernstack from "./topics/Mernstack";
export default class App extends Component {
  render() {
    return (
      <section>
        <header>
          {/* <h1>root component</h1> */}
          <Navbar />
        </header>
        <main>
          <Mernstack courses="javascript fullstack" />
          <Fullstack
            courses={[
              "javastack","MERNstack","PERN","PYTHONstack",
              "dotnetstack","phpstack","rubystack",
            ]}
          />
        </main>
      </section>
    );
  }
}
```

Fullstack.jsx

```jsx
import React from "react";
const Fullstack = props => {
  return (
    <div>
      <h1>list of courses</h1>
      {props.courses.map(course
=> (
        <ul>
          <li>{course}</li>
        </ul>
      ))}
    </div>
  );
};
export default Fullstack;
```

output



Props and children

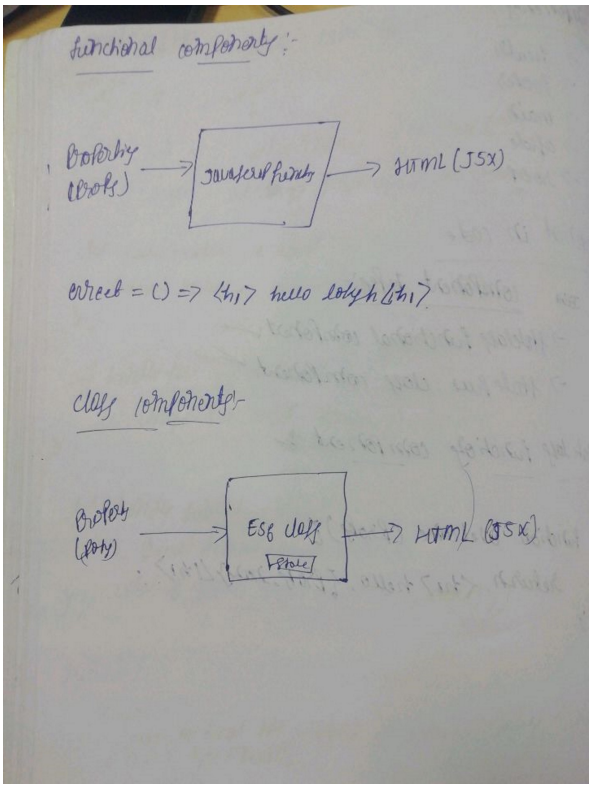
Thursday, February 24, 2022 10:05 AM

Props in React

Props, short for Properties in React Props, short for properties, allow the user to pass arguments or data to components. These props help make the components more dynamic. Props in a component are read-only and cannot be changed.

From <https://www.simplilearn.com/tutorials/reactjs-tutorial/what-is-reactjs?source=sl_frs_nav_playlist_video_clicked>

- Props are immutable objects
- DATA Can be sent from parent to child



Props useages

1. To send props inside a component.
2. Props can pass data from one component to another component.
3. Props through variable name
4. Props as a object
5. Props inside constructor

Send boolean value AS PROPS

App.jsx	Fullstack.jsx

`import React from 'react'`
`import Fullstack from './topics/Fullstack'`
`const app = () => {`
`  return (`
`  <div>`
`  <Fullstack isLoggedin={false}/>`
`  </div>`
`  )`
`}`
`export default app`
	`import React from "react";`
	`const Fullstack = props => {`
	`  if (props.isLoggedin === true) {`
	`  return (`
	`  <>`
	`  <button>Logout</button>`
	`  </>`
	`  );`
	`  } else {`
	`  return (`
	` <>

```
        <button>Login</button>
        <button>signup</button>
      </>
    );
  }
};
export default Fullstack;
```

Pass number as a value always use it inside curly brackets as expression

App.jsx

```
import React from "react";
import Fullstack from "./topics/Fullstack";
const app = () => {
  return (
    <div>
      <Fullstack age={20} />
    </div>
  );
};
export default app;
```

Fullstack.jsx

```
import React from "react";
const Fullstack = props => {
  return <div>{props.age}</div>;
};
export default Fullstack;
```

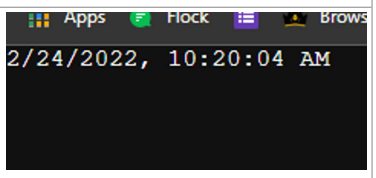
Passing date object

App.jsx

```
//!passing date
import React from 'react'
import Fullstack from './topics/Fullstack'
const app = () => {
  return (
    <div>
      <Fullstack date={new Date()}/>
    </div>
  )
}
export default app
```

Fullstack.jsx

```
import React from 'react'
const Fullstack = props => {
  return (
    <div>{ props.date.toLocaleString()}</div>
  )
}
export default Fullstack
```



Passing math object

App.jsx

```
//!pass math object
import React from 'react'
import Fullstack from './topics/Fullstack'
const app = () => {
  return (
    <div>
      <Fullstack totalMarks={Math.random()}/>
    </div>
  )
}
export default app
```

Fullstack.jsx

```
import React from "react";
const Fullstack = props => {
  return <div>{Math.round(props.totalMarks * 10)}</div>;
};
export default Fullstack;
```

Passing null values

App.jsx

```
//!passing null value
import React from "react";
import Fullstack from "./topics/Fullstack";
const app = () => {
  return (
    <div>
      <Fullstack nullvalue={null} />
    </div>
  );
};
export default app;
```

Fullstack.jsx

```
//!passing null value
import React from "react";
const Fullstack = props => {
  if (props.nullvalue === null) {
    return <h1>loading...</h1>;
  } else {
    return (
      <>
        <h1>Lorem ipsum dolor sit amet consectetur.</h1>
        <p>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Architecto
          vitae qui dolorem porro nobis hic aliquid voluptatem reiciendis, nam
          ipsam accusamus adipisci incidunt repellat! Eligendi voluptates rerum
          quis et optio!
```

```
        </p>
      </>
    );
  }
};
export default Fullstack;
```

Passing undefined value

App.jsx	Fullstack.jsx

```
//!passing undefined
import React from "react";
import Fullstack from "./topics/Fullstack";
const app = () => {
  return (
    <div>
      <Fullstack nullvalue={undefined} />
    </div>
  );
};
export default app;
```

For passing JSON data

App.jsx	ListUsers.jsx	GitUsers.jsx

```
//!passing JSON
import JSON from "./data.json";
import React from "react";
import ListUser from
"./Components/gitUsers/ListUser";
const app = () => {
  return <div>
    <ListUser JSON={JSON}/>
  </div>;
};
export default app;
```

```
import React from "react";
import GitUser from "./GitUser";
const ListUser = props => {
  return (
    <div>
      {props.JSON.map(user => {
       return <GitUser {...user}
/>;
      })}
    </div>
  );
};
export default ListUser;
```

```
import React from "react";
const GitUser = ({ login, avatar_url, html_url }) => {
  return (
    <>
      <figure>
        <img src={avatar_url} alt={login} />
        <h1>{login}</h1>
        <a href={html_url}>checkout</a>
      </figure>
    </>
  );
};
export default GitUser;
```

Data flow Json files is imported into app.jsx from here JSON data goes to ListUsers.jsx which is a child of app.jsx	Json data comes from app.jsx Here we are spreading that data and sending It to listerUser child "GitUser"	Now the spread data comes from ListUsers.jsx and instead of using props we destructure it in the parameter itself and use it to render JSX

Destructuring and giving alternates to keys or alias

```
import React from "react";
const GitUser = ({ login: name, avatar_url: URL, html_url }) => {
  return (
    <>
      <figure>
        <img src={URL} alt={name} />
        <h1>{name}</h1>
        <a href={html_url}>checkout</a>
      </figure>
    </>
  );
};
export default GitUser;
```

Passing a function for props

App.jsx	Fullstack.jsx

```
//!passing function as prop
import React from "react";
import Fullstack from "./topics/Fullstack";
const app = () => {
  return (
    <div>
      <Fullstack
        handleClick={() => {
```

```
//!passing function as prop
import React from "react";
const Fullstack = props => {
  return (
    <div>
      <button onClick={props.handleClick}> Click for tazz</button>
    </div>
  );
```

```
          alert("hello where are you?");
        }}
      />
    </div>
  );
};
export default app;
```

```
};
export default Fullstack;
```

Children is a default prop for every component

What is props.children?

props. children is a special prop, automatically passed to every component, that can be used to render the content included between the opening and closing tags when invoking a component.

App.jsx

```
//!creating multiple props
import React from "react";
import AllEmployess from
"./Components/Employess/AllEmployess";
const app = () => {
  return (
    <section id="empBlock">
      <article>
        <div>
          <AllEmployess
            emp_id="typ1"
            emp_name="shashi"
            emp_salary={10000}
            emp_de="web dev"
          >
            I am shashi very good html and css
          </AllEmployess>
          <AllEmployess
            emp_id="typ2"
            emp_name="bhuvan"
            emp_salary={10000}
            emp_de="web dev"
          >
            I am bhuvan average html and css
          </AllEmployess>
          <AllEmployess
            emp_id="typ3"
            emp_name="lokesh"
            emp_salary={10000}
            emp_de="github panta"
          >
            I am lokesh copy paste github kelsa
          </AllEmployess>
          <AllEmployess
            emp_id="typ4"
            emp_name="Girish"
            emp_salary={1000000000}
            emp_de="God dev"
          >
            I am Girish bug belters
          </AllEmployess>
        </div>
      </article>
    </section>
  );
};
export default app;
```

ALLEMPLOYESS.jsx

```
import React from "react";
const AllEmployess = props => {
  return (
      <>
          <br />
      <h1>{props.emp_name}</h1>
      <p>EMP ID:{props.emp_id}</p>
      <p>EMP Sal:{props.emp_salary}
</p>
      <p>EMP
designation:{props.emp_de}</p>
      <p>{props.children}</p>
    </>
  );
};
export default AllEmployess;
```

```
shashi
EMP ID:typ1
EMP Sal:10000
EMP designation:web dev
I am shashi very good html and css

bhuvan
EMP ID:typ2
EMP Sal:10000
EMP designation:web dev
I am bhuvan average html and css

lokesh
EMP ID:typ3
EMP Sal:10000
EMP designation:github panta
I am lokesh copy paste github kelsa

Girish
EMP ID:typ4
EMP Sal:1000000000
EMP designation:God dev
I am Girish bug belters
```

Setting default props

The defaultProps is a React component property that allows you to set default values for the props argument. If the prop property is passed, it will be changed. The defaultProps can be defined as a property on the component class itself, to set the default props for the class.

One way == defaultProps

```
import React from "react";
const AllEmployess = props => {
  return (
```

Another way === or operator

```
import React from "react";
const AllEmployess = props => {
  return (
```

```
    <>
      <br />
      <h1>{props.emp_name}</h1>
      <p>EMP ID:{props.emp_id}</p>
      <p>EMP Sal:{props.emp_salary}</p>
      <p>EMP designation:{props.emp_de}</p>
      <p>{props.children}</p>
    </>
  );
};
export default AllEmployess;
AllEmployess.defaultProps = {
  emp_name: "geek",
  emp_id: "xxx",
  emp_salary: 100000,
  emp_de: "dev",
};
```

```
geek
EMP ID:xxx
EMP Sal:100000
EMP designation:dev
I am Girish bug belters
```

```
    <>
      <br />
      <h1>{props.emp_name || "GAMEBOI"}</h1>
      <p>EMP ID:{props.emp_id}</p>
      <p>EMP Sal:{props.emp_salary}</p>
      <p>EMP designation:{props.emp_de}</p>
      <p>{props.children}</p>
    </>
  );
};
export default AllEmployess;
```

```
lokesh
EMP ID:typ3
EMP Sal:10000
EMP designation:github panta
I am lokesh copy paste github kelsa

GAMEBOI
EMP ID:
EMP Sal:
EMP designation:
I am Girish bug belters
```

Priority order
First in component tag
Second .defaultProps
Third or operator

npm install prop-types

What Are PropTypes In React? # PropTypes are a mechanism to ensure that components use the correct data type and pass the right data, and that components use the right type of props, and that receiving components receive the right type of props

```
export default AllEmployess;
AllEmployess.defaultProps = {
  emp_name: 100,
  emp_id: "xxx",
  emp_salary: 100000,
  emp_de: "dev",
};
AllEmployess.prototype = {
  emp_name: PropsTypes.string,
};
```

Go through npm documentation

```
Warning: Failed prop type: Invalid prop `emp_name` of type `number` supplied to `AllEmployees`, expected `string`.
    at AllEmployees (http://localhost:3000/static/js/bundle.js:209:23)
    at App
```

Props drilling

Prop Drilling is the process by which you pass data from one part of the React Component tree to another by going through other parts that do not need the data but only help in passing it around.

App.jsx	Navbar.jsx	Menu.jsx

`//!props drilling`
`import React from "react";`
`import Navbar from "./Components/Navbar/Navbar";`
	`import React from "react";`
	`import "./navbar.style.css";`
	`import Logo from "./Logo";`
	`import Menu from "./Menu";`
		`import React from "react";`
		`const Menu =props => {`
		`  if (props.isLoggedIn === true)`
		`{`

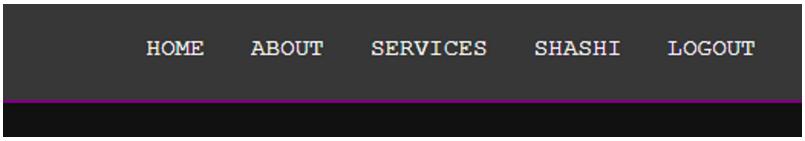
```jsx
const app = () => {
  return (
    <section id="empBlock">
      <article>
        <Navbar isLoggedIn={true} username="Shashi" />
      </article>
    </section>
  );
};
export default app;
```

The same props is sent to navbar

```jsx
const Navbar = () => {
  return (
    <section id="_NavbarSection">
      <article>
        <Logo />
        <Menu isLoggedIn={true} username="Shashi" />
      </article>
    </section>
  );
};
export default Navbar;
```

The same props is sent from navbar to menu
The data is sent from grand parent to grand child which is called props drilling

```jsx
    return (
      <div
className="__menuBlock">
        <nav>
          <ul>
            <li>
              <a href="/">
Home</a>
            </li>
            <li>
              <a href="/">
About</a>
            </li>
            <li>
              <a href="/">
Services</a>
            </li>
            <li>
              <a href="/">
{props.username}</a>
            </li>
            <li>
              <a href="/">
Logout</a>
            </li>
          </ul>
        </nav>
      </div>
    );
  } else {
    return (
      <div
className="__menuBlock">
        <nav>
          <ul>
            <li>
              <a href="/">
Home</a>
            </li>
            <li>
              <a href="/">
About</a>
            </li>
            <li>
              <a href="/">
Services</a>
            </li>
            <li>
              <a href="/">
Register</a>
            </li>
            <li>
              <a href="/">
Login</a>
            </li>
          </ul>
        </nav>
      </div>
    );
  }
};
export default Menu;
```



State

React components has a built-in state object. The state object is where you store property values that belongs to the component. When the state object changes, the component re-renders.

The state is an instance of React Component Class can be defined as ***an object of a set of observable properties that control the behavior of the component***. In other words, the State of a component is an object that holds some information that may change over the lifetime of the component.

```
//!class based components state object
import React, { Component } from "react";
export default class app extends Component {
  //*one way
  state = {
    username: "shashi",
  };
  //* another way
  constructor() {
    super();
    this.state = {
      company: "QSP",
    };
  }
  render() {
    console.log(this.state);
    return <div>App</div>;
  }
}
```

Ways	

`//*one way`
`state = {`
`username: "shashi",`
`};`
	`//* another way`
	`constructor() {`
	`super();`
	`this.state = {`
	`company: "QSP",`
	`};`
	`}`

Another example

```
//!class based components state object
import React, { Component } from "react";
export default class app extends Component {
  state = {
    firstname: "shashi",
    lastname: "kumar",
  };
  render() {
    return <div>{this.state.firstname + this.state.lastname}</div>;
  }
}
```

Initially functional based components we stateless
Before react 16.8 you cannot use states with functional based compnents
after react version 16.8 onwards REACT HOOKS came into picture
After react 16.8 onwards make stateless component into statefull component with help of react Hooks(useState())
Via HOOKs we can make functional based components into stateful.
React Hook use state cannot be called in a class component. React Hooks must be called in a react function component or a custom React hook function.

Always hooks should be called or written at the top or right after creation of functional component

Example for class based component statefull using this.state ob

```
import React, { Component } from "react";
export default class app extends Component {
  constructor(props) {
    super(props);
    this.state = {
      employees: [
        {
          emp_id: "typ1",
          emp_name: "manu",
          emp_salary: 20000,
          emp_designation: "node developer",
          emp_city: "mandya",
        },
        {
          emp_id: "typ2",
          emp_name: "shashi",
          emp_salary: 2000,
```

```
          emp_designation: "html developer",
          emp_city: "mysore",
        },
      ],
    };
  }
  render() {
    return (
      <div>
        <h1>I am class based components</h1>
        <>
          {this.state.employees.map(emp => {
            return (
              <>
                <h1>{emp.emp_name}</h1>
                <p>{emp.emp_id}</p>
                <p>{emp.emp_city}</p>
                <p>{emp.emp_salary}</p>
              </>
            );
          })}
        </>
      </div>
    );
  }
}
```

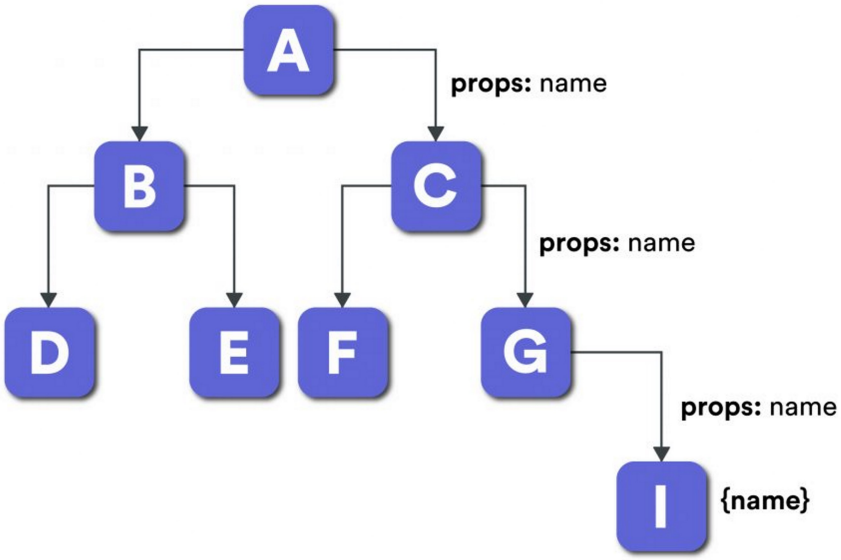
EmployeeFunction.jsx

```
import React, { useState } from "react";
let EmployeeFunction = () => {
  let [ employee, setEmployee ] = useState([
    {
      emp_id: "typ1",
      emp_name: "manu",
      emp_salary: 20000,
      emp_designation: "node developer",
      emp_city: "mandya",
    },
    {
      emp_id: "typ2",
      emp_name: "shashi",
      emp_salary: 2000,
      emp_designation: "html developer",
      emp_city: "mysore",
    },
  ]);
  return (
    <>
      {employee.map(emp => {
        return (
          <>
            <hr />
            <h1>{emp.emp_name}</h1>
            <p>{emp.emp_id}</p>
            <p>{emp.emp_city}</p>
            <p>{emp.emp_salary}</p>
          </>
        );
      })}
    </>
  );
};
export default EmployeeFunction;
```

App.jsx

```
import React, { Component } from "react";
import EmployeeFunction from "./Components/Employess/EmployeeFunction";
export default class app extends Component {
  constructor() {
    super();
    this.state = {
      employees: [
        {
          emp_id: "typ1",
          emp_name: "manu",
          emp_salary: 20000,
          emp_designation: "node developer",
          emp_city: "mandya",
        },
        {
          emp_id: "typ2",
          emp_name: "shashi",
          emp_salary: 2000,
          emp_designation: "html developer",
          emp_city: "mysore",
        },
      ],
    };
  }
  render() {
    return (
      <div>
        <h1>I am class based components</h1>
        <>
          <hr />
          {this.state.employees.map(emp => {
            return (
              <>
                <h1>{emp.emp_name}</h1>
                <p>{emp.emp_id}</p>
                <p>{emp.emp_city}</p>
                <p>{emp.emp_salary}</p>
              </>
            );
          })}
          <EmployeeFunction />
        </>
      </div>
    );
  }
```

```
}
```



Elements

- Elements are the smallest building blocks of React apps.
- React elements are plain object (JSX Objects) and are cheap to create.
- React DOM takes care of updating the DOM to match the React Elements.
- React only updates what's necessary.
- React Dom compares the element and its children to the previous one and only applies the Dom updates necessary to bring DOM to the desired state.

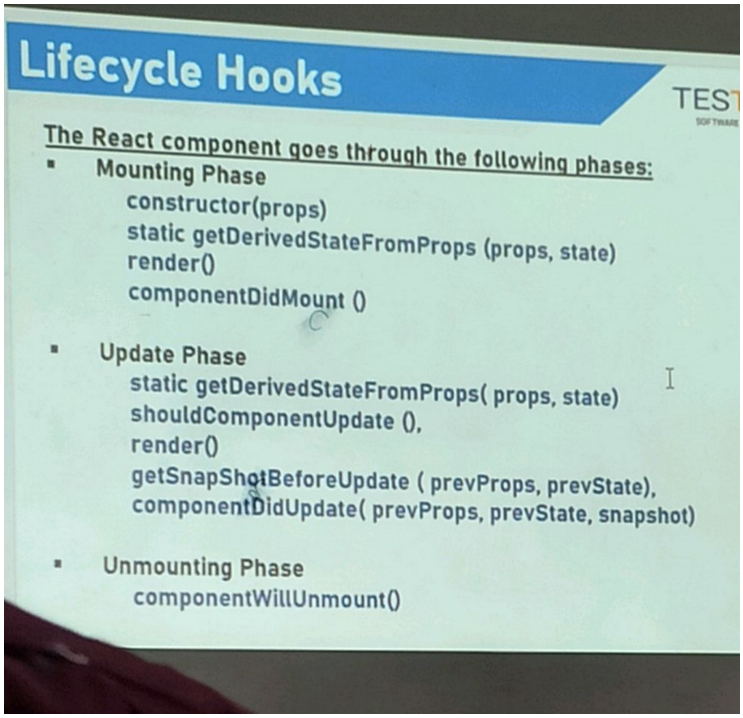
React life cycle

Friday, February 25, 2022 10:33 AM

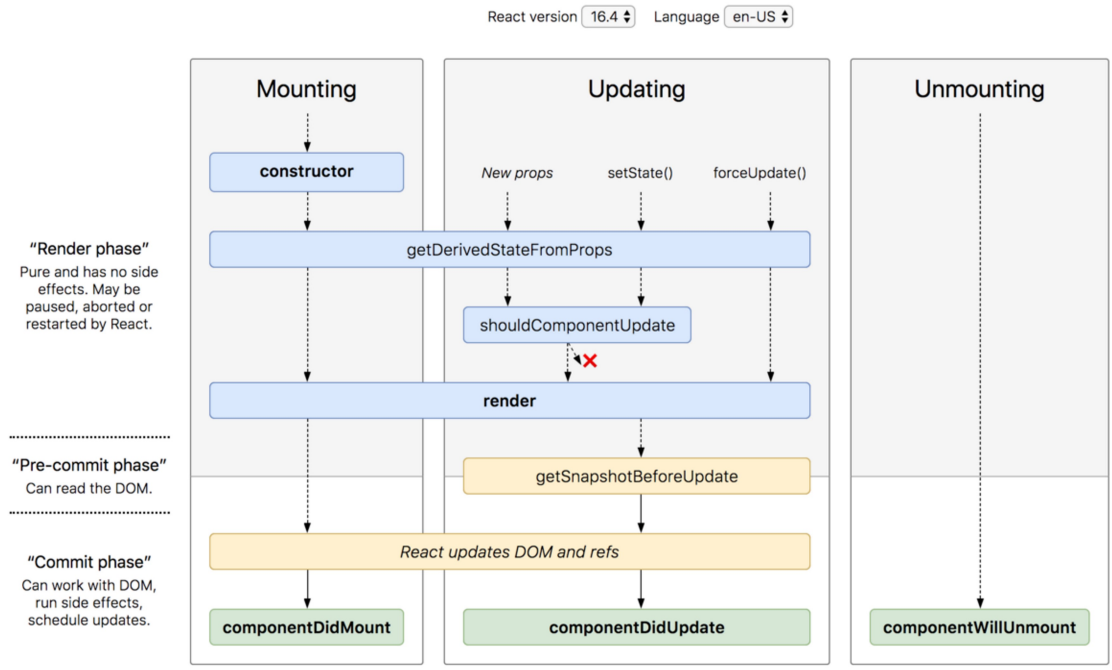
React Lifecycle

1. Mounting phase
2. Updating phase
3. Unmounting phase
4. Error boundaries

Mounting phase	Can be performed once
updating	Any number of times
unmounting	Can be only once
Error boundaries	



phases	methods	
Mounting phase	1. constructor(props) 2. static getDerivedStateFromProps (props, state) 3. render( ) 4. componentDidMount( ) - called once the component is rendered on the UI	Helps to insert, create or add elements into DOM
updating	1. static getDerivedStateFromProps (props,state) 2. shouldComponentUpdate( ), default value is true 3. render( ) 4. getSnapShotBeforeUpdate (prevProps , prevState), - dependent on componentDidUpdate 5. componentDidUpdate(prevProps, prevState, snapshot)	Helps to update the mounted elements in DOM
unmounting	1. componentWillUnmount( )	Removes or delete an element from the DOM
Error boundaries		




```
static getDerivedStateFromProps(props, state)
```

App.jsx

```
import React, { Component } from "react";
import Mounting from "../src/Components/lifecycle/mounting/Mounting";
export default class App extends Component {
  render() {
    return (
      <div>
        <Mounting />
      </div>
    );
  }
}
```

Mounting.jsx

```
import React, { Component } from 'react'
export default class Mounting extends Component {
    constructor(props) {
      super(props)

      this.state = {
         cars:"audi"
      }
    }
    static getDerivedStateFromProps(props, state) {
        return {
            cars: props.drive
        }
    }
  render() {
    return (
        <div>Mounting { this.state.cars}</div>
    )
  }
}
```

```
componentDidMount()
```

App.jsx

```
import React, { Component } from "react";
import Mounting from "../src/Components/lifecycle/mounting/Mounting";
export default class App extends Component {
  render() {
    return (
      <div>
        <Mounting />
      </div>
    );
  }
}
```

Mounting.jsx

```
import React, { Component } from "react";
export default class Mounting extends Component {
  constructor(props) {
    super(props);
    this.state = {
      cartoon: "mickey mouse",
    };
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({
        cartoon: "road runner",
      });
```

```
    }, 5000);
  }
  render() {
    return <div>{this.state.cartoon}</div>;
  }
}
```

```
shouldComponentUpdate() {
    return true;
  }
```

Default value of this method is true

getSnapShotBeforeUpdate (prevProps , prevState), this method is dependent on componentDidUpdate(prevProps, prevState, snapshot)

Reactupdate.jsx

```
import React, { Component } from "react";
export default class ReactUpdate extends Component {
  constructor(props) {
    super(props);
    this.state = {
      company: "TYSS",
    };
  }
  //mounting phase
  componentDidMount() {
    setTimeout(() => {
      this.setState({
        company: "JSPIDERS",
      });
    }, 2000);
  }
  shouldComponentUpdate() {
    return true;
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    console.log("before update", prevState);//TYSS
  }
  componentDidUpdate(prevProps, prevState, snapshot) {
    console.log(prevState);//TYSS
    console.log("After state update is", this.state.company);
    console.log("the snapshot is", snapshot);
  }
  render() {
    return <div>{this.state.sweet}</div>;
  }
}
```

Code

```
import React, { Component } from "react";
export default class ReactUpdate extends Component {
  constructor(props) {
    super(props);
    this.state = {
      company: "TYSS",
      location: "Bangalore",
    };
  }
  //mounting phase
  componentDidMount() {
    setTimeout(() => {
      this.setState({
        company: "JSPIDERS",
      });
    }, 2000);
  }
  shouldComponentUpdate() {
    return true;
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
```

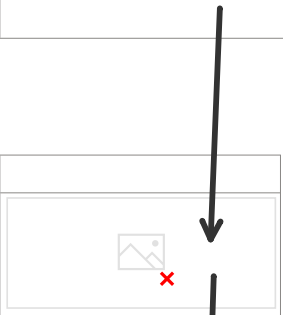
On click

```
    console.log("before update", prevState); //TYSS
  }
  componentDidUpdate(prevProps, prevState, snapshot) {
    console.log(prevState); //TYSS
    console.log("After state update is", this.state.company);
    console.log("the snapshot is", snapshot);
  }
  changeloc = () => {
    this.setState({
      location: "mysore",
      company: "PYSPIDERS",
    });
  };
  render() {
    return (
      <div>
        <div>company name:{this.state.company}</div>
        <div>location:{this.state.location}</div>
        <button onClick={this.changeloc}>change location</button>
      </div>
    );
  }
}
```

Login --> logout ---> login

ReactUpdate.jsx

```
import React, { Component } from "react";
export default class ReactUpdate extends Component {
  constructor(props) {
    super(props);
    this.state = {
      value: "login",
    };
  }
  logout = () => {
    if (this.state.value === "logout") {
      this.setState({
        value: "login",
      });
    } else {
      this.setState({
        value: "logout",
      });
    }
  };
  render() {
    return (
      <div>
        ReactUpdate
        <button onClick={this.logout}>{this.state.value}</button>
      </div>
    );
  }
}
```



Onclick

On click

MOUNTING PHASE

1. Constructor(props)
    a. Only to initialize state
    b. No business logic
    c. No sideEffects (no api calls like setimeout/setinterval)
    d. No setState( ) is used.
    e. Super(props)
        i. Initialize the parent constructor method.
        ii. It helps to inherit methods from parent (React.Component)
2. Static getDerivedStateFromProps(props, state)
    a. Static method
    b. Called after constructor( )
    c. It returns null/ returns new state
    d. It takes state as an argument and returns object with the change in the state.
    e. We can set the state of the object based on the initial props
3. Render( )
    a. It is used to write UI logic.
    b. No side effects
    c. No business logic
4. componentDidMount( )
    a. Called after the component is rendered.
    b. Can write the Business Logic
    c. Can have sideEffects
    d. When the component is mounted to the Real DOM this method will be executed.
    e. Can use the setState( )

UPDATING PHASE

4. getSnapShotBeforeUpdate( ) :
    a. It has access for props and state before the update.
    b. When using the getSnapShotBeforeUpdate( ) always need to include/call componentDidUpdate( ) or else Errors.

5. ComponetDidUpdate( ) :
    a. It called after the component is updated in the DOM.
    b. We can use this method individually.

UNMOUNTING PHASE:

Fragments:

1. Adding list to the group of elements adding without extra node.
2. It helps in dealing with the issue of json components.
3. React introduced Fragments from the 16.2 and above versions.
4. Fragments allow you to group a list of children without adding extra nodes to the DOM.
5. It helps in dealing with the issue of JSON components

Use of fragments

1. It makes the execution of code faster as compared to the div tag
2. It takes less memory

==========
useEffect( )
Note: it takes a callback function as a parameter.
useEffect( ) replaces all the below three methods from Lifecycle

1. componentDidMount( )
2. componentDidUpdate( )
3. componentWillUnmount( )

useEffect( )

1. Will be called after the Renders.
2. It mainly helps in SMART RENDERING.
3. It does not allow the unnecessary triggering.

REACT HOOKS

Monday, February 28, 2022 3:07 PM

React Hooks
can convert stateless components into stateful
Hooks are functions introduced in 16.8 onwards
Without writing a class implements class based features
Eg: useState

Rules for Hooks:

1. Do not call inside conditional blocks like if else etc
2. Do not call inside loops
3. Do not use it inside functions or callbacks funcitons
4. Do not call hooks from regualur javascript functions instead call from react function components

useState cannot be called inside a callback react hook must be called in a a react function component ot a custom React hook function.

Main hooks

1. useState( )
2. useEffect( )
3. useContext( )

Other hooks

1. useMemo( )
2. useLayoutEffect( )
3. useCallback( )
4. useReducer( )
5. useDebug( )

React Hooks are JS functions.
It's a collection of functions which is used by the React for its functional components.
Or
Hooks is a function that allows you to add the functionality for react component.

Using react hooks in functional components:

1. No state
2. No use of render
3. Use of arrow functions
4. All hooks start with keyword "use"

Hooks are only avilable in functional based components

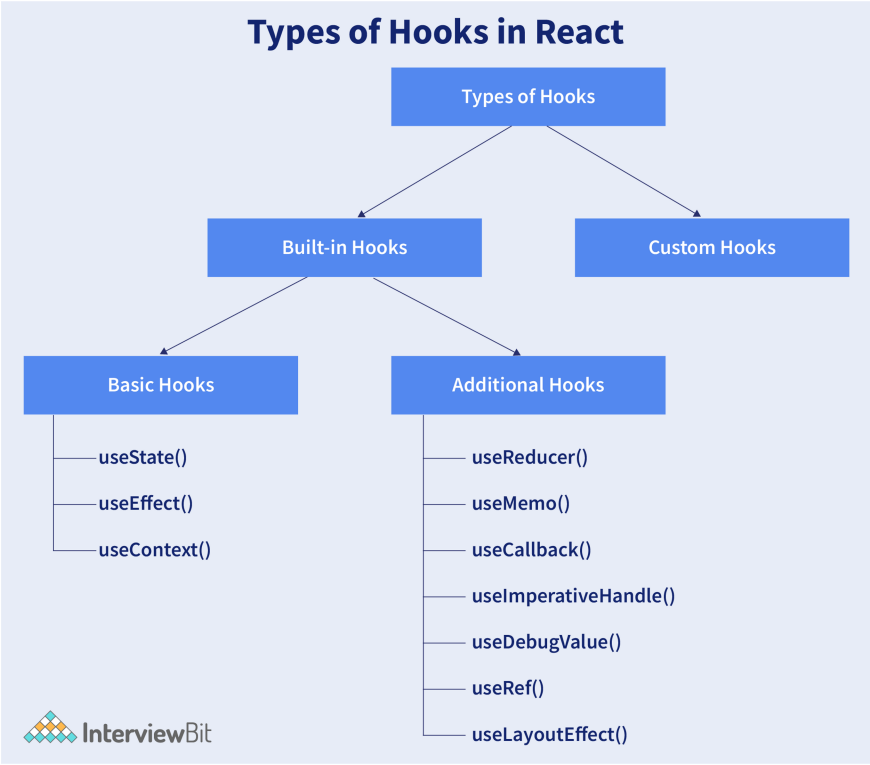
Functional based	Class based

`import React, { useState } from "react";`
`const App = () => {`
`  //declare a useState hook`
`  let [value, setValue] = useState("ajay");`
`  return (`
`  <div>`
`  App sir you are best <strong>{value}</strong>`
`  </div>`
`  );`
`};`
`export default App;`
	`import React, { Component } from "react";`
	`export default class App extends Component {`
	`  state = {`
	`  username: "priya",`
	`  };`
	`  render() {`
	`  return <div>App{this.state.username}</div>;`
	`  }`
	`}`

Cannot update state directly

setState() enqueues changes to the component state and tells React that this component and its children need to be re-rendered with the updated state. This is the primary method you use to update the user interface in response to event handlers and server responses.

We have two types of hooks



What Are React Hooks?

These are in-built functions that allow developers to use state and lifecycle methods within components in React. Each component's lifecycle has 3 phases which are mount, unmount, and update. Alongside that, components have states and properties. Hooks allow developers to use these methods whilst improving code reuse with greater flexibility navigating the component tree.

Why react hooks was introduced?

How useState hook works? What are the arguments accepted by this hook and what is returned by this hook?

useState( ) hook Is a function which is used to store state value in the functional component
useState accepts initial value, it can take number, boolean , array, object and string.

Basic syntax of useState( )
const [currentStateValue, functionToUpdateState]= useState(initialStateValue);

Question:
What are the differences in using hooks and class components with respect to state management?
Answer:
When using `setState()` in class components, always the state variable is an object. Where as, the state variable in hooks can be of any type like number, string, boolean, object or array.
When state variable is an object, `setState()` in class components automatically merges the new value to the state object. But in case of setter function in `useState()`, we need to explicitly merge the updated object property using spread operator.

From <https://backbencher.dev/articles/react-hooks-interview-questions>

We will introduce setState and prevState and use them in React.

setState and prevState in React

setState() and prevState() are useState hooks that are used to change state in a React class component.

setState() indicates that this component and its children components are changed and need to be re-rendered with the updated state. setState is the primary method used to update the UI in response to event handlers and server responses.

prevState() is the same as the setState but the only difference between them is that if we want to change the state of a component based on the previous state of that component, we use this.setState(), which provides us the prevState.

Let's check an example of a counter app. That can increment, decrement, reset numbers.

First of all, we will create a new file, Counter.js. In Counter.js we will import React from "react"; and create a function called Counter().

From <https://www.delftstack.com/howto/react/react-setstate-prevstate/>

https://dev.to/abhisheknaiidu/10-react-hooks-explained-3ino

https://www.smashingmagazine.com/2020/04/react-hooks-api-guide/

https://www.delftstack.com/howto/react/disable-button-in-react/

https://jschris.com/what-is-the-usestate-hook-with-examples

Bind method ways

Tuesday, March 1, 2022 10:49 AM

Function methods

https://medium.com/@omergoldberg/javascript-call-apply-and-bind-e5c27301f7bb
Call

```
//call apply bind methods
let employee = {
  emp_id: "typ1",
  emp_name: "shashi",
};
let qsp_employee = {
  emp_id: "qsp1",
  emp_name: "priya",
};
function Testyantra() {
  console.log(this);
}
Testyantra.call(qsp_employee);
```

Another example==========

```
let obj1 = {
    num=10,
}
function addNumbers(num2) {
    return this.num + num2;
}
//20
let total = addNumbers.call(obj1, 10);
console.log(total);
let users = {
    firstname:"shashi"
}
function getFullName(lastname) {
    return this.firstname + lastname;
}
let fullname = getFullName.call(users, "kunal");
console.log(fullname);
```

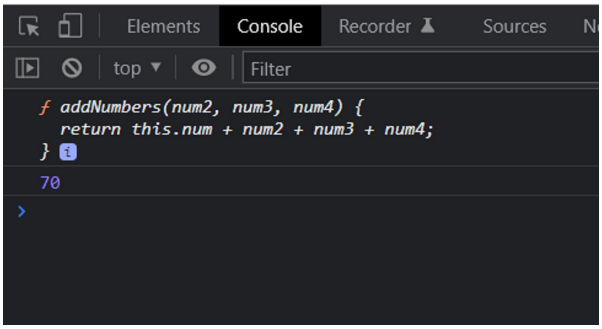
Apply

```
let obj1 = {
  num: 10,
};
function addNumbers(num2, num3, num4) {
  return this.num + num2 + num3 + num4;
}
let total = addNumbers.apply(obj1, [10, 20, 30]);
console.log(total);
```

Bind

```
let obj1 = {
  num: 10,
};
function addNumbers(num2, num3, num4) {
  return this.num + num2 + num3 + num4;
}
let total = addNumbers.bind(obj1, 10, 20, 30);
console.log(total);
```

```
console.log(total());
```



Constructor is used for

1. Initializing state object
2. Binding this keyword

Constructor is best places for binding this keyword in class based component

Binding with constructor	Binding without constructor

`import React, { Component } from "react";`
`export default class ClassBasedCounter extends Component`
`{`
`  constructor() {`
`  super();`
`  this.state = {`
`  counter: 0,`
`  };`
`  this.Increment = this.Increment.bind(this);`
`  this.Decrement = this.Decrement.bind(this);`
`  this.Reset = this.Reset.bind(this);`
`  }`
`  Increment() {`
`  console.log(this);`
`  }`
`  Decrement() {`
`  console.log(this);`
`  }`
`  Reset() {`
`  console.log(this);`
`  }`
`  render() {`
`  return (`
`  <section className="CounterApp">`
`  <article>`
`  <div>`
`  <h2>{this.state.counter}</h2>`
`  <button onClick={this.Increment}>`
`increment</button>`
`  <button onClick={this.Decrement}>`
`decrement</button>`
`  <button onClick={this.Reset}>reset</button>`
`  </div>`
`  </article>`
`  </section>`
`  );`
`  }`
`}`
	`import React, { Component } from "react";`
	`export default class ClassBasedCounter extends Component {`
	`  // constructor() {`
	`  //  super();`
	`  //  this.state = {`
	`  //  counter: 0,`
	`  //  };`
	`  //  this.Increment = this.Increment.bind(this);`
	`  //  this.Decrement = this.Decrement.bind(this);`
	`  //  this.Reset = this.Reset.bind(this);`
	`  // }`
	`  state = {`
	`  counter: 0,`
	`  }`
	`  Increment() {`
	`  console.log(this);`
	`  }`
	`  Decrement() {`
	`  console.log(this);`
	`  }`
	`  Reset() {`
	`  console.log(this);`
	`  }`
	`  render() {`
	`  return (`
	`  <section className="CounterApp">`
	`  <article>`
	`  <div>`
	`  <h2>{this.state.counter}</h2>`
	`  <button onClick={this.Increment.bind(this)}>`
	`increment</button>`
	`  <button onClick={this.Decrement.bind(this)}>`
	`decrement</button>`
	`  <button onClick={this.Reset.bind(this)}>reset</button>`
	`  </div>`
	`  </article>`
	`  </section>`
	`  );`
	`  }`
	`}`

We can also use arrow functions, as the concept of this does not exist in a arrow function it points to class

Arrow functions way	

```
import React, { Component } from "react";
export default class ClassBasedCounter extends Component {
  // constructor() {
  //   super();
  //   this.state = {
  //     counter: 0,
  //   };
  //   this.Increment = this.Increment.bind(this);
  //   this.Decrement = this.Decrement.bind(this);
  //   this.Reset = this.Reset.bind(this);
  // }
  state = {
    counter: 0,
  };
  Increment = () => {
    console.log(this);
  };
  Decrement = () => {
    console.log(this);
  };
  Reset = () => {
    console.log(this);
  };
  render() {
    return (
      <section className="CounterApp">
        <article>
          <div>
            <h2>{this.state.counter}</h2>
            <button onClick={this.Increment}>increment</button>
            <button onClick={this.Decrement}>decrement</button>
            <button onClick={this.Reset}>reset</button>
          </div>
        </article>
      </section>
    );
  }
}
```

There are two ways to achieve the same

Class based

```
import React, { Component } from "react";
export default class ClassBasedCounter extends Component {
  // constructor() {
  //   super();
  //   this.state = {
  //     counter: 0,
  //   };
  //   this.Increment = this.Increment.bind(this);
  //   this.Decrement = this.Decrement.bind(this);
  //   this.Reset = this.Reset.bind(this);
  // }
  state = {
    counter: 0,
  };
  Increment = () => {
    this.setState({ counter: this.state.counter + 1 });
  };
  Decrement = () => {
    this.setState({ counter: this.state.counter - 1 });
  };
  Reset = () => {
    this.setState({ counter: 0 });
  };
  render() {
    return (
      <div>
        <strong>class based Component</strong>
        <h2>{this.state.counter}</h2>
        <button onClick={this.Increment}>increment</button>
        <button onClick={this.Decrement}>decrement</button>
        <button onClick={this.Reset}>reset</button>
      </div>
```

Functional based

```
import React, { useState } from "react";
const FunctionalBasedCounter = () => {
  let [counter, setCounter] = useState(0);
  let Increment = () => {
    setCounter(counter + 1);
  };
  let Decrement = () => {
    setCounter(counter - 1);
  };
  let Reset = () => {
    setCounter(0);
  };
  return (
    <div>
      <strong>functional based Component</strong>
      <h2>{counter}</h2>
      <button onClick={Increment}>increment</button>
      <button onClick={Decrement}>decrement</button>
      <button onClick={Reset}>reset</button>
    </div>
  );
};
export default FunctionalBasedCounter;
```

```
    );
  }
}
```

For cleaner updates use this

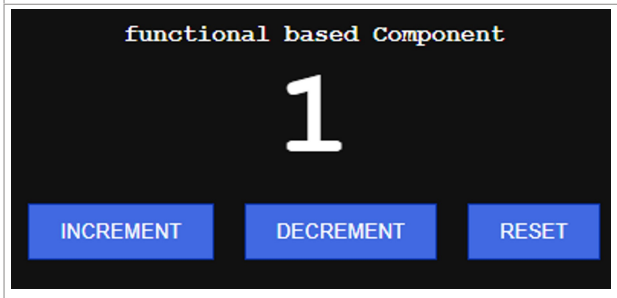
Before

```
import React, { useState } from "react";
const FunctionalBasedCounter = () => {
  let [counter, setCounter] = useState(0);
  let Increment = () => {
    setCounter(counter + 1);
     setCounter(counter - 1);
  };
  let Decrement = () => {
    setCounter(counter - 1);
  };
  let Reset = () => {
    setCounter(0);
  };
  return (
    <div>
      <strong>functional based Component</strong>
      <h2>{counter}</h2>
      <button onClick={Increment}>increment</button>
      <button onClick={Decrement}>decrement</button>
      <button onClick={Reset}>reset</button>
    </div>
  );
};
export default FunctionalBasedCounter;
```

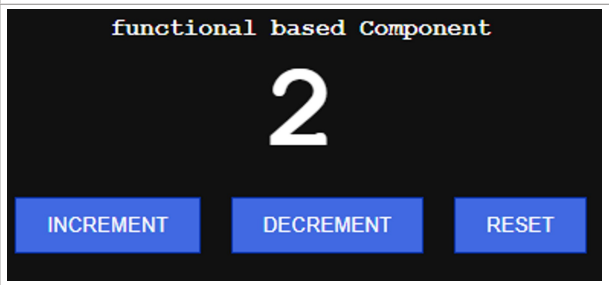
After

```
import React, { useState } from "react";
const FunctionalBasedCounter = () => {
  let [counter, setCounter] = useState(0);
  let Increment = () => {
    setCounter(prevState => prevState + 1);
     setCounter(prevState => prevState - 1);
  };
  let Decrement = () => {
    setCounter(prevState => prevState - 1);
  };
  let Reset = () => {
    setCounter(0);
  };
  return (
    <div>
      <strong>functional based Component</strong>
      <h2>{counter}</h2>
      <button onClick={Increment}>increment</button>
      <button onClick={Decrement}>decrement</button>
      <button onClick={Reset}>reset</button>
    </div>
  );
};
export default FunctionalBasedCounter;
```

Expected output is increment by 2 but it only increments by one



Expected to increment by 2



yarn add faker

Events, conditional rendering

Tuesday, March 1, 2022 3:14 PM

React events

Events without react	Events with react

```
import React, { Component } from "react";
import ClassBasedCounter from
"./Components/counter/ClassBasedCounter";
import FunctionalBasedCounter from
"./Components/counter/FunctionalBasedCounter";
import ExampleEvents from
"./Components/Events/ExampleEvents";
export default class App extends Component {
  componentDidMount() {
    let btn = document.querySelector("#btn");
    btn.addEventListener("mouseenter", e => {
      e.target.style.background = "red";
      e.target.innerHTML = "clicked it";
    });
    btn.addEventListener("mouseleave", e => {
      e.target.style.background = "white";
      e.target.style.color = "black";
      e.target.innerHTML = "reverting back";
    });
============================or============
     btn.onmouseenter = function (e) {
           e.target.style.background =
     "crimson";
           e.target.style.color = "red";
         };
         btn.onmouseleave = function (e) {
           e.target.style.background = "Blue";
           e.target.style.color = "green";
         };
     Inside componentDidMount( )
=======================================
Form events
let input = document.querySelector("#input");
    let template =
document.querySelector("#template");
    input.addEventListener("keyup", e => {
      console.log(e);
      if (e.type === "keyup") {
        template.innerHTML = e.target.value;
      }
    });
Inside componentDidMount( )
=======================================
  }
  render() {
    return (
      <section className="counterApp">
        <article>
          {/* <ClassBasedCounter />
          <hr />
          <FunctionalBasedCounter />
          <hr /> */}
          <ExampleEvents />
          <button id="btn">click me</button>
        </article>
      </section>
    );
  }
}
```

Speech event

```
import React, { Component } from "react";
import ClassBasedCounter from "./Components/counter/ClassBasedCounter";
import FunctionalBasedCounter from "./Components/counter/FunctionalBasedCounter";
import ExampleEvents from "./Components/Events/ExampleEvents";
export default class App extends Component {
  componentDidMount() {
    // let btn = document.querySelector("#btn");
    // btn.addEventListener("mouseenter", e => {
    //   e.target.style.background = "red";
    //   e.target.innerHTML = "clicked it";
    // });
    // btn.addEventListener("mouseleave", e => {
    //   e.target.style.background = "white";
    //   e.target.style.color = "black";
    //   e.target.innerHTML = "reverting back";
    // });
    // ========================================
    // btn.onmouseenter = function (e) {
    //   e.target.style.background = "crimson";
    //   e.target.style.color = "red";
    // };
    // btn.onmouseleave = function (e) {
    //   e.target.style.background = "Blue";
    //   e.target.style.color = "green";
    // };
    // let input = document.querySelector("#input");
    // let template = document.querySelector("#template");
    // input.addEventListener("keyup", e => {
    //   console.log(e);
    //   if (e.type === "keyup") {
    //     template.innerHTML = e.target.value;
    //   }
    // });
    //====================================
    let speech = window.speechRecognition || window.webkitSpeechRecognition;
    let recognition = new speech();
    recognition.addEventListener("result", e => {
      console.log(e);
      let value = e.results[0][0].transcript;
      document.querySelector("#template").innerHTML = value;
    });
    recognition.addEventListener("end", recognition.start);
    recognition.start();
  }
  render() {
    return (
      <section className="counterApp">
        <article>
          {/* <ClassBasedCounter />
          <hr />
          <FunctionalBasedCounter />
          <hr /> */}
          <ExampleEvents />
          {/* <button id="btn">click me</button> */}
          <div id="template"></div>
          {/* <input type="text" id="input" placeholder="enter some text" /> */}
        </article>
      </section>
    );
  }
}
```

By default synthetic events are in bubbling phase

addEventlistener is also attached internally
The following are the syntax differences compared to DOM

- React events are named using camelCase, rather than lowercase
- With JSX you can pass a function as the event handler, rather than a string.

For example, the HTML:

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

is slightly different in React:

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

Another difference is that you cannot return `false` to prevent default behavior in React. You must call `preventDefault` explicitly. For example, with plain HTML, to prevent the default form behavior of submitting, you can write:

```
<form onsubmit="console.log('You clicked submit.'); return false">
  <button type="submit">Submit</button>
</form>
```

Synthetic events are cross browser events compared to DOM
React events are slightly different from DOM events

React defines these synthetic events according to the W3C spec, so you don't need to worry about cross-browser compatibility. React events do not work exactly the same as native events. See the SyntheticEvent reference guide to learn more.

When using React, you generally don't need to call addEventListener to add listeners to a DOM element after it is created. Instead, just provide a listener when the element is initially rendered.

From <https://reactjs.org/docs/handling-events.html>

Simple example	

```
import React, { Component } from "react";
import ExampleEvents from "./Components/Events/ExampleEvents";
export default class App extends Component {
  handleSubmit = e => {
    e.preventDefault();
    let input = document.querySelector("#input");
    console.log(input.value);
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <ExampleEvents />
          <form>
```

```
            <input type="text" id="input" placeholder="enter some text" />
            <button onClick={this.handleSubmit}>submit</button>
          </form>
        </article>
      </section>
    );
  }
}
```

Form Dom events	Form react events

```
import React, { Component } from "react";
import ExampleEvents from "./Components/Events/ExampleEvents";
export default class App extends Component {
  componentDidMount() {
    let form = document.querySelector("form");
    form.addEventListener("submit", e => {
      e.preventDefault();
      console.log(e);
      if (e.type === "submit") {
        console.log(e.target[0].value);
      }
    });
  }
  render() {
    return (
      <section className="counterApp">
        <article>
          <ExampleEvents />
          <form>
            <input type="text" id="input" placeholder="enter some text" />
            <button onClick={this.handleSubmit}>submit</button>
          </form>
        </article>
      </section>
    );
  }
}
```

Using and operator

```
import React, { useState } from "react";
const ConditionalRendering = () => {
  let [messages, setMessages] = useState([
    "shashi is waiting for something",
    "priya is waiting",
    "reply to  ajay",
  ]);
  let PrintMessages = () => {
    return (
      <span>
        unread messages
        <strong id="unread">{messages.length}</strong>
      </span>
    );
  };
    return <div>{ messages.length>0 && <PrintMessages/>}</div>;
};
export default ConditionalRendering;
```

Using ternary operator

```
import React, { useState } from "react";
const ConditionalRendering = () => {
  let [state, setState] = useState(false);
  let clickMe = () => {
    setState(!state);
  };
  return (
```

```
    <div>
      <div>{state ? "priyanka" : "shashi"}</div>
      <button onClick={clickMe}>click me</button>
    </div>
  );
};
export default ConditionalRendering;
```

Switch case

```
//!using switch case
import React, { useState } from "react";
const ConditionalRendering = () => {
  let [auth, setAuth] = useState(false);
  switch (auth) {
    case true:
      return <button>logout</button>;
    case false:
      return <button>login</button>;
    default:
      return null;
  }
};
export default ConditionalRendering;
```

Conditional Rendering

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like if or the conditional operator to create elements representing the current state, and let React update the UI to match them.

From <https://reactjs.org/docs/conditional-rendering.html>

A. Logical and
B. If else
C. Ternary
D. Switch case

React Refs

Wednesday, March 2, 2022 2:16 PM

Refs and the DOM

Refs provide a way to access DOM nodes or React elements created in the render method.

In the typical React dataflow, props are the only way that parent components interact with their children. To modify a child, you re-render it with new props. However, there are a few cases where you need to imperatively modify a child outside of the typical dataflow. The child to be modified could be an instance of a React component, or it could be a DOM element. For both of these cases, React provides an escape hatch.

When to Use Refs

There are a few good use cases for refs:

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Avoid using refs for anything that can be done declaratively.

For example, instead of exposing open() and close() methods on a Dialog component, pass an isOpen prop to it.

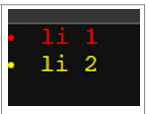
Don’t Overuse Refs

Your first inclination may be to use refs to “make things happen” in your app. If this is the case, take a moment and think more critically about where state should be owned in the component hierarchy. Often, it becomes clear that the proper place to “own” that state is at a higher level in the hierarchy. See the Lifting State Up guide for examples of this.

From <https://reactjs.org/docs/refs-and-the-dom.html>

Ref's are simple properties
Do not over use refs it is a costly resource

```
import React, { Component, createRef } from "react";
export default class App extends Component {
  li1 = createRef();
  li2 = createRef();
  componentDidMount() {
    this.li1.current.style.color = "red";
    this.li2.current.style.color = "yellow";
  }
  render() {
    return (
      <div>
        <li ref={this.li1}>li 1</li>
        <li ref={this.li2}>li 2</li>
      </div>
    );
  }
}
```



For class based use createRef( )
In case of functional based use hooks useref( )

The useRef is a hook that allows to directly create a reference to the DOM element in the functional component. Syntax: const refContainer = useRef(initialValue);
The useRef returns a mutable ref object. This object has a property called .

```
import React, { useRef } from "react";
const App = () => {
  let inputRef = useRef();
  inputRef.current.style.color = "red";
  return <div ref={inputRef}>App</div>;
};
export default App;
```

	Class based	Functional based

For media playback And an example of conditional rendering	`import React, { Component, createRef } from "react";`
	`import VIDEO from "./video.mp4";`
	`export default class App extends Component {`
	`  constructor() {`
	`  super();`
	`  this.state = {`
	`  play: true,`
	`  btn: "play",`
	`  };`
	`  this.videoRef = createRef();`
	`  }`
	`  PlayOrpuase = () => {`
	`  this.setState({ play: !this.state.play });`
	`  if (this.state.play) {`
	`  this.videoRef.current.play();`
	`  } else {`
	`  this.videoRef.current.pause();`
	`  }`
	`  };`
	`  render() {`
	`  return (`
	`  <section className="counterApp">`
	`  <article>`
	`  <video`
	`  src={VIDEO}`
	`  width="100%"`
	`  ref={this.videoRef}`
	`  onClick={this.PlayOrpuase}`
	`  ></video>`
	`  </article>`
	`  </section>`
	`  );`
	`  }`
	`}`
		`import React, { useRef, useState } from "react";`
		`import VIDEO from "./video.mp4";`
		`const App = () => {`
		`  let videoRef = useRef();`
		`  let [play, setPlay] = useState(true);`
		`  let PlaySong = () => {`
		`  setPlay(!play);`
		`  play === true ? videoRef.current.play() :`
		`videoRef.current.pause();`
		`  };`
		`  return (`
		`  <section id="counterApp">`
		`  <article>`
		`  <video src={VIDEO} ref={videoRef} onClick={PlaySong}>`
		`</video>`
		`  </article>`
		`  </section>`
		`  );`
		`};`
		`export default App;`

Camera	`import React, { useRef } from "react";`
	`const App = () => {`
	`  let videoRef = useRef();`
	`  let GetCamera = async () => {`
	`  let media = await window.navigator.mediaDevices.getUserMedia({`
	`  audio: true,`
	`  video: { width: 1200, height: 600 },`
	`  });`
	`  videoRef.current.srcObject = media;`
	`  videoRef.current.onloadedmetadata = function () {`
	`  videoRef.current.play();`
	`  };`
	`  };`
	`  return (`
	`  <section id="counterApp">`
	`  <article>`
	`  <video ref={videoRef}></video>`
	`  <button onClick={GetCamera}>Get Camera</button>`
	`  </article>`
	`  </section>`
	`  );`
	`};`

```
export default App;
```

LIST and keys

List

```
//!list and keys
import React, { Fragment, useState } from "react";
const App = () => {
  let [languages, setlang] = useState(["java", "python", "js", "vuejs"]);
  return (
    <div>
      {languages.map((lang, index) => {
        return (
          <Fragment key={index}>
            <li>{lang}</li>
          </Fragment>
        );
      })}
    </div>
  );
};
export default App;
```

React, really simplifies the rendering of lists inside the JSX by supporting the JavaScript .map( ) method.

`key` is a special attribute that you need to include in the element, and it should be a string. Keys in each list should be unique, which means that you shouldn't use values that can be the same as the key. In other words, `keys` should be unique among the siblings, not globally.

From <https://www.blog.duomly.com/list-and-keys-in-react/>

==========
React Refs:

-> Refs is the shorthand used for references in React. (Similar to keys).
-> Its an attribute which makes it possible to store a reference to particular DOM nodes or React elements.
-> It provides a way to access React DOM nodes or React elements and how to interact with it.
-> It is used when we want to change the value of a child component, without making the use of props.

When to Use Refs :
* Managing focus, text selection, or media playback.
* Triggering imperative animations.
* Integrating with third-party DOM libraries.
* Also used in callbacks

Note: The ref attribute cannot be used on function components because they don't have instances.

forms

Thursday, March 3, 2022 11:44 AM

Using react way by using DOM	

```
//!form using normal DOM way
import React, { Component } from "react";
export default class App extends Component {
  getData = () => {
    let form = document.querySelector("form");
    form.addEventListener("submit", e => {
      e.preventDefault();
      if (e.type === "submit") {
        let username = e.target[0].value;
        let password = e.target[1].value;
        console.log({ username, password });
      }
    });
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <form>
            <div>
              <label htmlFor="username">username</label>
              <input type="text" id="username" placeholder="username" />
            </div>
            <div>
              <label htmlFor="password">password</label>
              <input type="text" id="password" placeholder="password" />
            </div>
            <div>
              <button onClick={this.getData}>submit</button>
            </div>
          </form>
        </article>
      </section>
    );
  }
}
```

Two ways in reactjs for forms (interview question)

1. Uncontrolled way not recommended using react refs
2. Controlled way using react states recommended way

	Functional way	Class way
Uncontrolled way		

```
import React, { useRef } from "react";
const App = () => {
  let usernameRef = useRef();
  let passwordRef = useRef();
  let handleSubmit = e => {
    e.preventDefault();
    let username = usernameRef.current.value;
    let password = passwordRef.current.value;
    console.log({ username, password });
  };
  return (
    <section className="counterApp">
      <article>
        <form>
          <div>
            <label htmlFor="username">
username</label>
            <input
              type="text"
              id="username"
```

```
import React, { Component, createRef } from "react";
export default class App extends Component {
  constructor() {
    super();
    this.usernameRef = createRef();
    this.passwordRef = createRef();
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleSubmit(e) {
    e.preventDefault();
    let username = this.usernameRef.current.value;
    let password = this.passwordRef.current.value;
    console.log({ username, password });
  }
  render() {
    return (
      <section className="counterApp">
        <article>
          <form>
            <div>
```

```
              placeholder="username"
              ref={usernameRef}
            />
          </div>
          <div>
            <label htmlFor="password">
password</label>
            <input
              type="text"
              id="password"
              placeholder="password"
              ref={passwordRef}
            />
          </div>
          <div>
            <button onClick={handleSubmit}>
submit</button>
          </div>
        </form>
      </article>
    </section>
  );
};
export default App;
```

```
              <label htmlFor="username">username</label>
              <input
                type="text"
                id="username"
                placeholder="username"
                ref={this.usernameRef}
              />
            </div>
            <div>
              <label htmlFor="password">password</label>
              <input
                type="text"
                id="password"
                placeholder="password"
                ref={this.passwordRef}
              />
            </div>
            <div>
              <button onClick={this.handleSubmit}>
submit</button>
            </div>
          </form>
        </article>
      </section>
    );
  }
}
```

What is UNCONTROLLED component?

An uncontrolled component is a component that renders form elements, where the form element's data is handled by the DOM (default DOM behavior). To access the input's DOM node and extract its value you can use a ref.

Uncontrolled Components

In most cases, we recommend using controlled components to implement forms. In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.

To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

From <https://reactjs.org/docs/uncontrolled-components.html>

Since an uncontrolled component keeps the source of truth in the DOM, it is sometimes easier to integrate React and non-React code when using uncontrolled components. It can also be slightly less code if you want to be quick and dirty. Otherwise, you should usually use controlled components.

controlled inside functional component	Controlled way class based

```
//*controlled component functional way
import React, { useState } from "react";
const App = () => {
  let [email, setEmail] = useState("");
  let [password, setPassword] =
useState("");
  let handleSubmit = e => {
    e.preventDefault();
    console.log({ email, password });
  };
  return (
    <section className="counterApp">
      <article>
        <form onSubmit={handleSubmit}>
          <div>
```

```
//!controlled component
import React, { Component } from "react";
//create state Object
//add value attribute into the form elements
//****add onchange event to the form elements */
export default class App extends Component {
  state = {
    email: "",
    password: "",
  };
  handleSubmit = e => {
    e.preventDefault();
    console.log(this.state);
  };
  handleChange = e => {
```

```
            <label htmlFor="email">
email</label>
            <input
              type="text"
              id="email"
              placeholder="email"
              value={email}
              name="email"
              onChange={e =>
setEmail(e.target.value)}
            />
          </div>
          <div>
            <label htmlFor="password">
password</label>
            <input
              type="text"
              id="password"
              placeholder="password"
              value={password}
              name="password"
              onChange={e =>
setPassword(e.target.value)}
            />
          </div>
          <div>
            <button>submit</button>
          </div>
        </form>
      </article>
    </section>
  );
};
export default App;
```

```
    let { name, value } = e.target;
    this.setState({ [name]: value });
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <form onSubmit={this.handleSubmit}>
            <div>
              <label htmlFor="email">email</label>
              <input
                type="text"
                id="email"
                placeholder="email"
                value={this.state.email}
                name="email"
                onChange={this.handleChange}
              />
            </div>
            <div>
              <label htmlFor="password">password</label>
              <input
                type="text"
                id="password"
                placeholder="password"
                value={this.state.password}
                name="password"
                onChange={this.handleChange}
              />
            </div>
            <div>
              <button>submit</button>
            </div>
          </form>
        </article>
      </section>
    );
  }
}
```

For controlled way In class based components the setState merges implicitliy but in functional based component we need to manually merge it.

Controlled class merging implicitly

```
//!controlled component class way
import React, { Component } from "react";
//create state Object
//add value attribute into the form elements
//****add onchange event to the form elements */
export default class App extends Component {
  state = {
    email: "",
    password: "",
  };
  handleSubmit = e => {
    e.preventDefault();
    console.log(this.state);
  };
  handleChange = e => {
    let { name, value } = e.target;
    this.setState({ [name]: value });
  };
  render() {
    return (
<section className="counterApp">
  <article>
    <form onSubmit={this.handleSubmit}>
      <div>
        <label htmlFor="email">email</label>
        <input
          type="text"
          id="email"
          placeholder="email"
          value={this.state.email}
```

Controlled functional mergering manually (IMPORTANT)

```
//!controlled funcitonal manual merging
import React, { useState } from "react";
const App = () => {
  let [state, setState] = useState({
    email: "",
    password: "",
  });
  let { email, password } = state;
  let handleChange = e => {
    let { name, value } = e.target;
    setState({ ...state, [name]: value });
  };
  let handleSubmit = e => {
    e.preventDefault();
    console.log(state);
  };
  return (
    <section className="counterApp">
      <article>
        <form onSubmit={handleSubmit}>
          <div>
            <label htmlFor="email">email</label>
            <input
              type="text"
              id="email"
              placeholder="email"
              value={email}
              name="email"
              onChange={handleChange}
            />
          </div>
```

```
          name="email"
          onChange={this.handleChange}
        />
      </div>
      <div>
        <label htmlFor="password">password</label>
        <input
          type="text"
          id="password"
          placeholder="password"
          value={this.state.password}
          name="password"
          onChange={this.handleChange}
        />
      </div>
      <div>
        <button>submit</button>
      </div>
    </form>
  </article>
</section>
    );
  }
}
```

```
          <div>
            <label htmlFor="password">password</label>
            <input
              type="text"
              id="password"
              placeholder="password"
              value={password}
              name="password"
              onChange={handleChange}
            />
          </div>
          <div>
            <button>submit</button>
          </div>
        </form>
      </article>
    </section>
  );
};
export default App;
```

FILES

Files in uncontrolled way class based

```
//!file upload using uncontrolled way
import React, { Component, createRef } from "react";
export default class App extends Component {
  constructor() {
    super();
    this.usernameRef = createRef();
    this.imageRef = createRef();
  }
  handleSubmit = e => {
    e.preventDefault();
    let username = this.usernameRef.current.value;
    let image = this.imageRef.current.files[0];
    //handling files in a uncontrolled way
    console.log(username);
    console.log(image);
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <form onSubmit={this.handleSubmit}>
            <div>
              <label htmlFor="email">email</label>
              <input
                type="text"
                id="email"
                placeholder="email"
                ref={this.usernameRef}
              />
            </div>
            <div>
              <label htmlFor="file">upload file</label>
              <input type="file" ref={this.imageRef} />
            </div>
            <div>
              <button>submit</button>
            </div>
          </form>
        </article>
      </section>
    );
  }
}
```

Files controlled way class based

Very very important

```
//!file upload controlled class based
import React, { Component } from "react";
export default class App extends Component {
  state = {
    username: "",
    image: "",
  };
  handleChange = e => {
    this.setState({ username: e.target.value });
  };
  handleFile = e => {
    this.setState({ image: e.target.files[0] });
  };
  handleSubmit = e => {
    e.preventDefault();
    console.log(this.state);
  };
  render() {
    return (
      <section className="counterApp">
        <article>
          <form onSubmit={this.handleSubmit}>
            <div>
              <label htmlFor="email">email</label>
              <input
                type="text"
                id="email"
                placeholder="email"
                value={this.state.username}
                onChange={this.handleChange}
              />
            </div>
            <div>
              <label htmlFor="file">upload file</label>
              <input type="file" onChange={this.hnadleFile} />
            </div>
            <div>
              <button>submit</button>
            </div>
          </form>
        </article>
      </section>
    );
```

```
  }
}
```

Monday==========
All controlled components
Select list importing data from countries.json

```
import React, { Fragment, useState } from "react";
import COUNTRY from "./countries.json";
const App = () => {
  let [country, setCountry] = useState(COUNTRY);
  return (
    <>
      <section id="counterApp">
        <article>
          <form>
            <label htmlFor="username">select country</label>
            <select
              name="country"
              id="country"
              onChange={e => setCountry(e.target.value)}
            >
              {country.map((val, index) => (
                <Fragment key={index}>
                  <option value={val.code}>{val.name}</option>
                </Fragment>
              ))}
            </select>
          </form>
          <div>
            <button>submit</button>
          </div>
        </article>
      </section>
    </>
  );
};
export default App;
```

How to handle radio button in react

```
import React, { useState } from "react";
const App = () => {
  let [username, setUsername] = useState("");
  let [gender, setGender] = useState(false);
  let handleSubmit = e => {
    e.preventDefault();
    console.log({ username, gender });
  };
  return (
    <>
      <section className="counterApp">
        <article>
          <form onSubmit={handleSubmit}>
            <div>
              <label htmlFor="username">username</label>
              <input
                type="text"
                placeholder="enter username"
                id="username"
                value={username}
                onChange={e => setUsername(e.target.value)}
              />
            </div>
            <aside onChange={e => setGender(e.target.value)}>
              <input type="radio" name="gender" id="gender" value="male" />
              male
              <input type="radio" name="gender" id="gender" value="female" />
              female
            </aside>
```

```
            <button type="submit">submit</button>
          </form>
        </article>
      </section>
    </>
  );
};
export default App;
```

How to handle check box in react = assignment

How to handle date object

```
//!date object and gender
import React, { useState } from "react";
const App = () => {
  let [username, setUsername] = useState("");
  let [gender, setGender] = useState(false);
  let [dob, setDob] = useState("");
  let handleSubmit = e => {
    e.preventDefault();
    console.log({ username, gender, dob });
  };
  return (
    <>
      <section className="counterApp">
        <article>
          <form onSubmit={handleSubmit}>
            <div>
              <label htmlFor="username">username</label>
              <input
                type="text"
                placeholder="enter username"
                id="username"
                value={username}
                onChange={e => setUsername(e.target.value)}
              />
            </div>
            <aside onChange={e => setGender(e.target.value)}>
              <input type="radio" name="gender" id="gender" value="male" />
              male
              <input type="radio" name="gender" id="gender" value="female" />
              female
            </aside>
            <div>
              <label htmlFor="dob">DOB</label>
              <input
                type="date"
                placeholder="enter DOB"
                id="dob"
                value={dob}
                onChange={e => setDob(e.target.value)}
              />
            </div>
            <button type="submit">submit</button>
          </form>
        </article>
      </section>
    </>
  );
};
export default App;
```

Input range

```
<div>
              <label htmlFor="range">price range</label>
              <input
                type="range"
                name="range"
                id="range"
```

```
                min={0}
                max={1000}
                value={price}
                onChange={e => setPrice(e.target.value)}
              />
              {price}
            </div>
```

Components	UnControlled Components (uses refs)	Controlled Components (uses state)
Class Based Components	1. Import the createRef 2. Add the ref attribute, ref={} 3. Use the same ref attribute value, and get the current value. let name = refValue.current.value	1. Create the State Object 2. Add the name, value attributes 3. Use the onChange event
Functional Based Components	1. Import the useRef Hook 2. Add the ref attribute, ref={} 3. Use the same ref attribute value, and get the current value. let name = refValue.current.value	1. Import the useState Hook 2. Add the name or value attributes 3. Use the onChange event

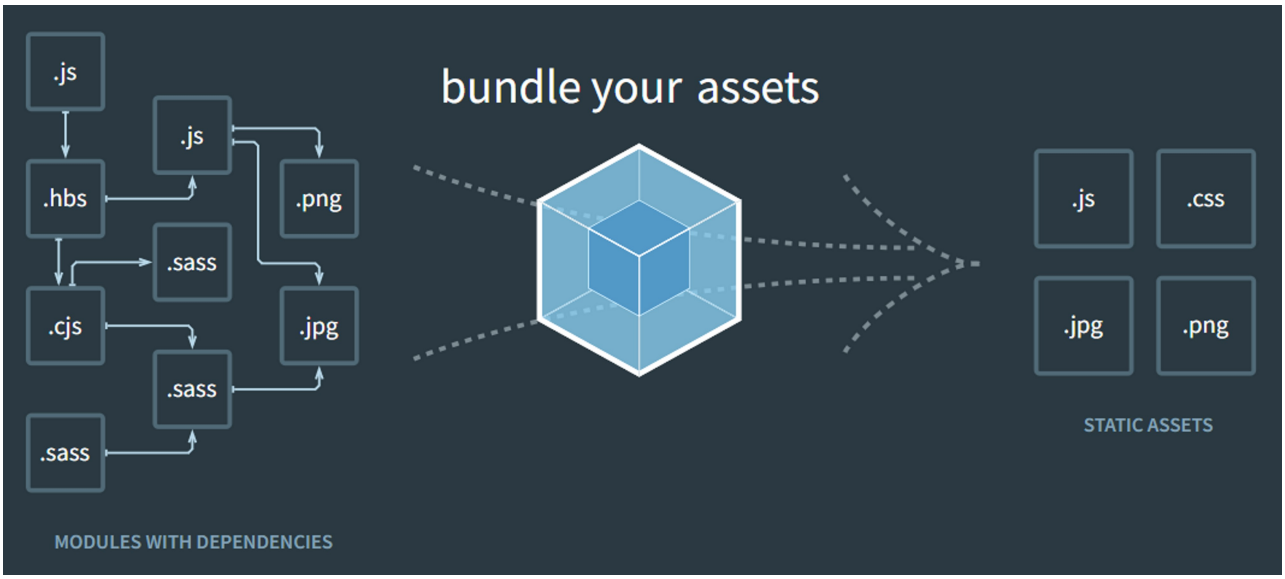
Styling in react js

Monday, March 7, 2022 11:27 AM

Webpack is a JS bundler

What is webpack?

Webpack is an open-source JavaScript module bundler. It is made primarily for JavaScript, but it can transform front-end assets such as HTML, CSS, and images if the corresponding loaders are included. webpack takes modules with dependencies and generates static assets representing those modules.



First way of attaching - the normal way

```
import "./global.css";
```

inside index.js and it will render as a webpack

Second way of attaching
JS way by passing objects

App.jsx

```
import React from "react";
import { STYLES } from "./style";
const App = () => {
  return (
    <section style={STYLES.containerStyle}>
      <article>
        <p style={STYLES.para}>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Temporibus
          expedita fuga id ducimus aperiam voluptas ullam, omnis nam eum ea.
        </p>
      </article>
    </section>
  );
};
export default App;
```

Style.js

```
export let STYLES = {
  containerStyle: {
    margin: "0 auto",
    width: "80%",
    background: "red",
    color: "white",
  },
  para: {
    fonSize: "16px",
    color: "#fff",
    padding: "20px",
  },
};
```

CSS modules third way of attaching
What are CSS modules in React?
CSS module is a CSS file in which all class names and animation names are scoped locally by default. In short, all the CSS declared in the file are local to the file in which this CSS file is imported. We will use CSS modules in the context of React but they are not limited to just React.

Stylemodule.jsx	Style.module.css

```jsx
import React, { useState } from "react";
import Style from "./style.module.css";
const StyleModule = () => {
  let [rating, setRating] = useState(4);
  return (
    <section className={Style.mainContainerParent}>
      <article className={Style.container}>
        <h2>Lorem ipsum, dolor sit amet consectetur adipisicing.</h2>
        <p>
          <span className={rating > 10 ? Style.paraGreen : Style.paraRed}>
            hello
          </span>
        </p>
      </article>
    </section>
  );
};
export default StyleModule;
```

```css
.mainContainerParent {
  color: black;
  padding: 10px;
  width: 100%;
  height: auto;
}
.container {
  width: 80%;
  margin: 0 auto;
}
.paraGreen {
  color: green;
}
.paraRed {
  color: red;
}
```

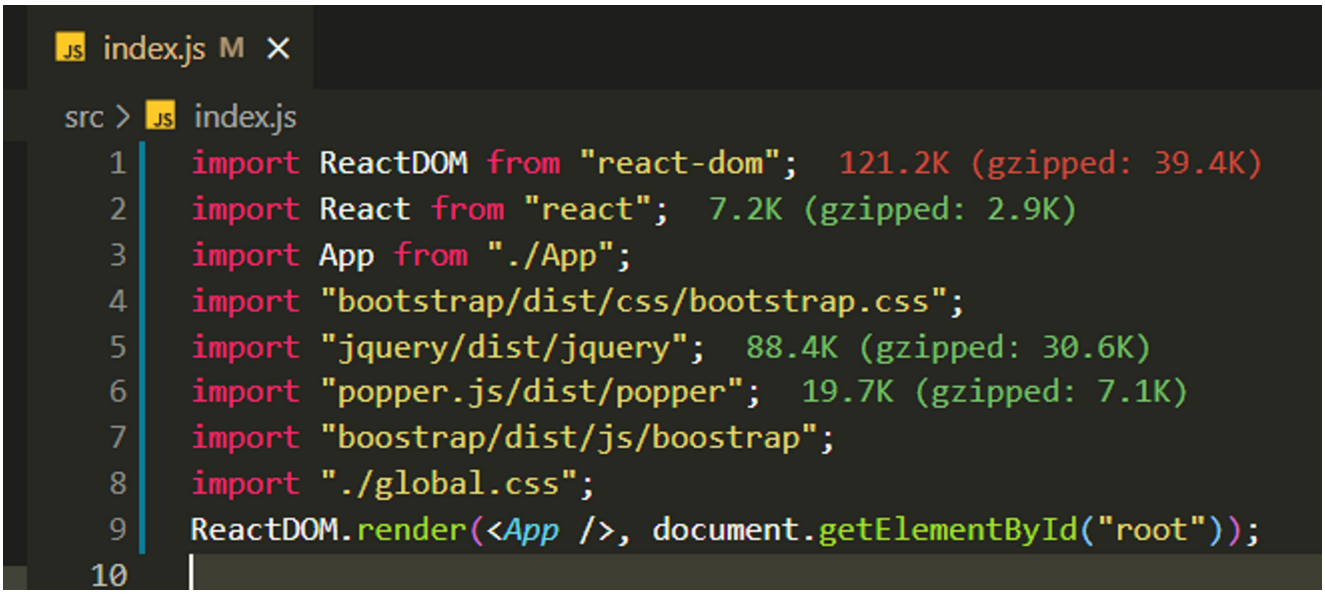
Bootstrap in React

Thursday, March 10, 2022 10:04 AM

yarn add bootstrap@4.6 jquery popper.js

Add these in index.js	

```
import "bootstrap/dist/css/bootstrap.css";
import "jquery/dist/jquery";
import "popper.js/dist/popper";
import "boostrap/dist/js/boostrap";
import "./global.css";
```



Routing

Thursday, March 10, 2022 10:20 AM

Routing means path
If you want to navigate or change path.

1. React doesnot have routing. Reat routing is not part of React API. This is an external Api that is provided by React-router-dom
2. React-router-dom is a library which provides routing concept
3. React-router-dom latest version is 6

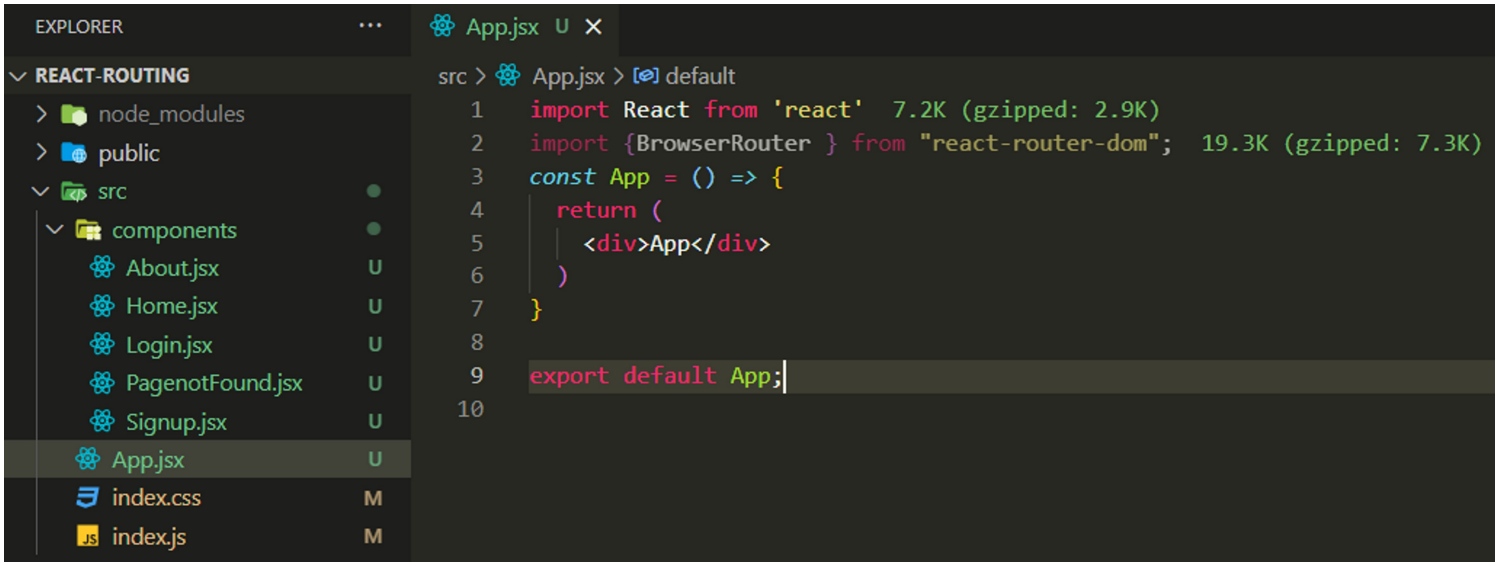
npx create-react-app react-routing

yarn add react-router-dom@5.3.0

npm cache clean --force

BrowserRouter: BrowserRouter is a router implementation that uses the HTML 5 history API(pushState, replaceState and the popstate event) to keep your UI in sync with the URL. It is the parent component that Is used to store all the other components.

Import react-router-dom into root file App.jsx



React-router

1. React router is a routing library built on top of the React which is used to create the routing in react applications.
2. React router dom is the version of React router Version5 or V5 designed for web applications.
3. React router v5 was divided into three packages.
    a. React-router : it is common core components between DOM and Native version.
    b. React-router-dom: this is the DOM version designed for browsers or web applications.
    c. React-router-native: this is the native version designed for react native mobile applications.

Route is used for calling path and component

Why we need switch in ReactDOM?
Thw switch component looks through all of its child routes and it displays the first one whose path matches the current URL. This component is what we use in most cases for most applications, because we have multiple routes and multuple plate pages in our app but we only want to show one page at a time.

exact prop is used
Switch is used
	`import React from "react";`
	`import { BrowserRouter, Route, Switch } from "react-router-dom";`
	`import Home from "./components/Home";`
	`import About from "./components/About";`
	`const App = () => {`
	`return (`
	`<BrowserRouter>`
	`<Switch>`
	`<Route path="/" component={Home} exact />`

```
        <Route path="/about" component={About} exact />
      </Switch>
    </BrowserRouter>
  );
};
export default App;
```

Why we need exact prop in react router dom?
React router does partial matching, so /users partially matches /users/create , so it would incorrectly return the Users route again. The exact param disables the partial matching for a route and makes sure that it only returns the route if the path is an EXACT match to the current url.

Page not found
Do not use exact
And give path as *

```
import React from "react";
import { BrowserRouter, Route, Switch } from "react-router-dom";
import Home from "./components/Home";
import About from "./components/About";
import Login from "./components/Login";
import Signup from "./components/Signup";
import PagenotFound from "./components/PagenotFound";
const App = () => {
  return (
    <BrowserRouter>
      <Switch>
        <Route path="/" component={Home} exact />
        <Route path="/about" component={About} exact />
        <Route path="/login" component={Login} exact />
        <Route path="/signup" component={Signup} exact />
        <Route path="*" component={PagenotFound} />
      </Switch>
    </BrowserRouter>
  );
};
export default App;
```

Second way to write the same routes

Above V5.0

```
import React, { Fragment } from "react";
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import Home from "./components/Home";
import Login from "./components/Login";
import About from "./components/About";
import PagenotFound from "./components/PagenotFound";
import Signup from "./components/Signup";
const App = () => {
  return (
    <Fragment>
      <Router>
        <Route>
          <Switch>
            <Route path="/" exact>
              <Home />
            </Route>
            <Route path="/login" exact>
              <Login />
            </Route>
            <Route path="/about" exact>
              <About />
            </Route>
            <Route path="/signup" exact>
              <Signup />
            </Route>
            <Route path="*">
              <PagenotFound />
            </Route>
          </Switch>
        </Route>
      </Router>
    </Fragment>
  );
};
export default App;
```

Navbar.jsx

```jsx
import React from "react";
import { Link } from "react-router-dom";
const Navbar = () => {
  return (
    <section>
      <article>
        <div className="logoBlock">
          <a href="#">Jspiders</a>
        </div>
        <div className="menuBlock">
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
            <li>
              <Link to="/login">Login</Link>
            </li>
            <li>
              <Link to="/signup">Signup</Link>
            </li>
          </ul>
        </div>
      </article>
    </section>
  );
};
export default Navbar;
```

Writing the same in react-router-dom 6.0

V6.0
Do not use exact

```jsx
import React, { Fragment } from "react";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Home from "./Components/Home";
const App = () => {
  return (
    <Fragment>
      <Router>
        <Navbar />
        <Routes>
          <Route path="/" element={<Home />} />
          <Route path="/about" element={<About />} />
          <Route path="/signup" element={<Signup />} />
          <Route path="/login" element={<Login />} />
          <Route path="*" element={<PagenotzFound />} />
        </Routes>
      </Router>
    </Fragment>
  );
};
```

React Router Components

1. Browser Router
2. Route
3. Link
4. Switch

- BrowserRouter: It **helps to keep UI in sync with URL and it's a parent component**, which stores all other components.
- Route: its conditionally shown component, it renders the UI only if the path matches the current URL.
- Link: it helps to create the links to different routes and implements the navigation around Application. (works like <a> tag)
  - Problems with anchor tag and why we use link?
  - It allows to refreshing of a page
  - No default export
- Switch: It helps to render only the first route that matches the location. Basically to render a single components.
- Exact: it is used to match the exact value with URL.
- Path: path specifies a path name we assign to our component.
- Component: it refers to the component which will render a matching path.

How to pass parameters to a route?
This hook helps us get the parameter passed on the URL without using any props object.
`useParams` hook here to access the dynamic peces of the URL.
useParams returns an object of key/value pairs of the URL parameters. Use it to access match.params of the current <Route>

We use a hook called useParams it works like slug in express

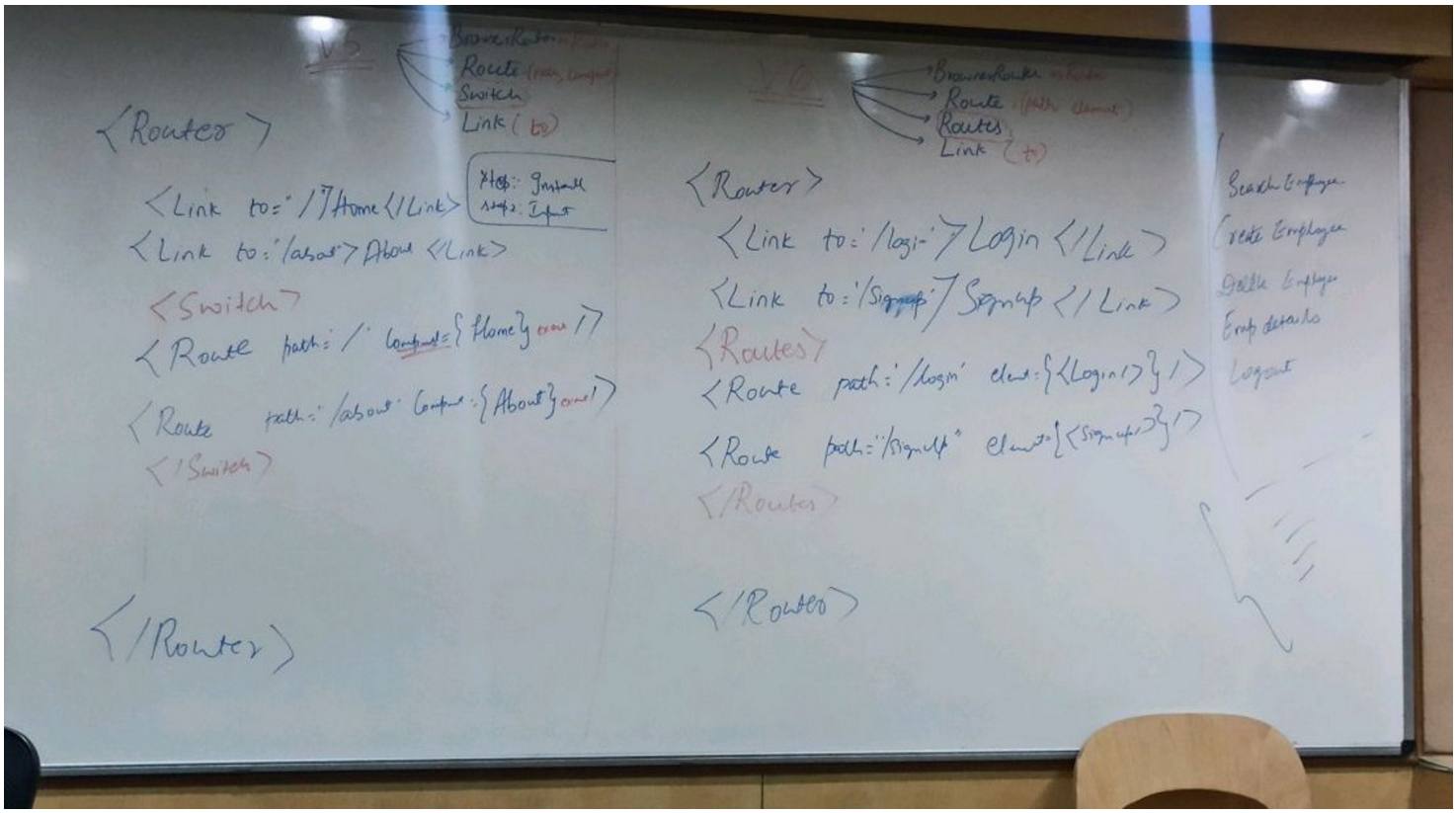
To nagivate programmatically we use useNavigate hook

React router hooks

1. useRoutes
2. useparams
3. useLocation
4. useNavigate

React Hooks

1. useState( )
2. useEffect( )
3. useRef( )
4. useContext( )
5. useMemo( )
6. useReducer( )



Next day React Routing

Creating child components

```
 <Route path="admin" element={<Admin />}>
            <Route path="profile" element={<Profile />} />
            <Route path="courses" element={<Courses />} />
          </Route>
```

Now using otlet we can attach children routes

```
import React from "react";
import { Link, Outlet } from "react-router-dom";
const Admin = () => {
  return (
    <section className="adminBlock">
      <article>
        <div className="sideBar">
          <nav>
            <ul>
              <li>
                <Link to="/admin/profile">Profile</Link>
              </li>
              <li>
                <Link to="/admin/courses">Courses</Link>
              </li>
            </ul>
          </nav>
        </div>
        <div className="contentBlock">
          <Outlet />
        </div>
      </article>
    </section>
  );
};
export default Admin;
```

Refer the files to better understand

Better way using useRoutes hook

Traditional way in app.jsx	Using hooks in app.jsx

`import React, { Fragment } from "react";`
`import { BrowserRouter as Router, Routes, Route } from "react-router-dom";`
`import Home from "./Components/Home";`
`import About from "./components/About";`
`import Signup from "./components/Signup";`
`import Login from "./components/Login";`
`import Admin from "./components/Admin/Admin";`
`import Profile from "./components/Admin/Profile";`
`import Courses from "./components/Admin/Courses";`
`import UpdatePassword from "./components/Admin/Profile/UpdatePassword";`
`import UploadProfilePhoto from "./components/Admin/Profile/UploadProfilePhoto";`
`const App = () => {`
`  return (`
`    <Fragment>`
`      <Router>`
`        <Navbar />`
`        <Routes>`
`          <Route path="/" element={<Home />} />`
`          <Route path="/about" element={<About />} />`
`          <Route path="/signup" element={<Signup />} />`
`          <Route path="/login" element={<Login />} />`
`          <Route path="admin" element={<Admin />}>`
`            <Route path="profile" element={<Profile />}>`
`              <Route path="upload-photo" element={<UploadProfilePhoto />} />`
`              <Route path="update-password" element={<UpdatePassword />} />`
`            </Route>`
`            <Route path="courses" element={<Courses />} />`
`          </Route>`
`          <Route path="*" element={<PagenotzFound />} />`
`        </Routes>`
`      </Router>`
`    </Fragment>`
`  );`
`};`
	App.jsx see below

App.jsx

```jsx
import React from "react";
import { BrowserRouter } from "react-router-dom";
import Navbar from "./components/navbar/Navbar";
import JspidersRoutes from
"./components/Routes/JspidersRoutes";
const App = () => {
  return (
    <BrowserRouter>
      <Navbar />
      <JspidersRoutes />
    </BrowserRouter>
  );
};
export default App;
```

Jspider

```jsx
import React from "react";
import { useRoutes } from "react-router-dom";
import Home from "./Components/Home";
import About from "./components/About";
import Signup from "./components/Signup";
import Login from "./components/Login";
import Admin from "./components/Admin/Admin";
import Profile from
"./components/Admin/Profile";
import Courses from
"./components/Admin/Courses";
import PagenotFound from
"./components/PagenotFound";
import UpdatePassword from
"./components/Admin/Profile/UpdatePassword";
import UploadProfilePhoto from
"./components/Admin/Profile/UploadProfilePhoto"
;
const JspidersRoutes = () => {
  let Rute = useRoutes([
    {
      path: "/",
      element: <Home />,
    },
    {
      path: "about",
      element: <About />,
    },
    {
      path: "login",
      element: <Login />,
    },
    {
      path: "signup",
      element: <Signup />,
    },
    {
      path: "admin",
      element: <Admin />,
      children: [
        {
          path: "profile",
          element: <Profile />,
          children: [
            {
              path: "upload-photo",
              element: <UploadProfilePhoto />,
            },
            {
              path: "update-password",
              element: <UpdatePassword />,
            },
          ],
        },
        {
          path: "courses",
          element: <Courses />,
        },
      ],
    },
    {
      path: "*",
      element: <PagenotFound />,
    },
  ]);
  return Rute;
};
export default JspidersRoutes;
```

Navbar.jsx

```jsx
import React, { useState } from "react";
import { Link, useParams } from "react-router-dom";
const Navbar = () => {
  let [admin, setAdmin] = useState(true);
  let { name } = useParams();
```

Admin.jsx

```jsx
import React from "react";
import { Link, Outlet } from "react-router-dom";
const Admin = () => {
  return (
    <section className="adminBlock">
```

```
  return (
    <section id="HeaderBlock">
      <article>
        <div className="logoBlock">
          <a href="#">Jspiders</a>
        </div>
        <div className="menuBlock">
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
            <li>
              <Link to="/login">Login</Link>
            </li>
            <li>
              <Link to="/signup">Signup</Link>
            </li>
            <li>
              <Link to={`/product/:${name}`}>product</Link>
            </li>
            {admin && (
              <li>
                <Link to="/admin">Admin</Link>
              </li>
            )}
          </ul>
        </div>
      </article>
    </section>
  );
};
export default Navbar;
```

```
      <article>
        <div className="sideBar">
          <nav>
            <ul>
              <li>
                <Link to="/admin/profile">Profile</Link>
              </li>
              <li>
                <Link to="/admin/courses">Courses</Link>
              </li>
            </ul>
          </nav>
        </div>
        <div className="contentBlock">
          <Outlet />
        </div>
      </article>
    </section>
  );
};
export default Admin;
```

See how to make child component

Using outlet in tag	children as js object key

Using slug /:id

App.jsx

```
//!using slug as id
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Navbar from "./components/navbar/Navbar";
import Product from "./components/Product/product";
const App = () => {
  return (
    <BrowserRouter>
      <Navbar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/product/:name" element={<Product />} />
      </Routes>
    </BrowserRouter>
  );
};
export default App;
```

Product.jsx

```
import React from "react";
import { useParams } from "react-router-dom";
const Product = () => {
  let { name } = useParams();
  return <div>product:catargory:{name}</div>;
};
export default Product;
```

If you click on the button go to home page

App.jsx

```
//!using slug as id
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-
dom";
import Navbar from "./components/navbar/Navbar";
import Product from "./components/Product/product";
const App = () => {
  return (
```

Products.jsx

```
import React from "react";
import { useParams, useNavigate } from "react-router-dom";
const Product = () => {
  let { name } = useParams();
  let navigate = useNavigate();
  let GoToHome = e => {
    navigate("/");
  };
```

```
    <BrowserRouter>
      <Navbar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/product/:name" element={<Product />}
/>
      </Routes>
    </BrowserRouter>
  );
};
export default App;
```

```
  return (
    <div style={{ padding: "10px" }}>
      <h1>{name.toUpperCase()}</h1>
      <button onclick={GoToHome}>product:catargory:{name}
</button>
    </div>
  );
};
export default Product;
```

What is HashRouter in reactjs?
A browserRouter that using html5 historyAPI (pushState, replaceState and the popstate event).

Some props of BrowserRouter
Example:
<BrowserRouter
 basename={optionalString}
 forceRefresh={optionalBool}
 getUserConfirmation={optionalFunc}
 keyLength={optionalNumber}
>
 <App />
</BrowserRouter>;

HashRouter
hashType has three properties
Slash, noslash, hashbang

Go through this link	https://medium.com/@daniel.hramkov/browserrouter-vs-hashrouter-e8bf1c3824ce

basename — string

Here you can give a baseURL for your application.
`<BrowserRouter basename="/main" />`

`<NavLink to="/profile"/> // renders <a href="/main/profile">`

That props need for correct setting in Github Pages.

forceRefresh — bool

BrowserRouter don't reload a page after changing url location.

If `true` the router will use full page refreshes on page navigation. You may want to use this to imitate the way a traditional server-rendered app would work with full page refreshes between page navigation.
`<BrowserRouter forceRefresh={true} />`

From <https://medium.com/@daniel.hramkov/browserrouter-vs-hashrouter-e8bf1c3824ce>

getUserConfirmation — function

A function to use to confirm navigation. Defaults to using `window.confirm`.
`functionName = (message, callback) => {`

`// this is the default behavior`

`const allowTransition = window.confirm(message);`

`callback(allowTransition);`

`}<BrowserRouter`

`getUserConfirmation={functionName}`

`/>`

From <https://medium.com/@daniel.hramkov/browserrouter-vs-hashrouter-e8bf1c3824ce>

keyLength — number

The length of `location.key`. Defaults to 6.
`<BrowserRouter keyLength={8} />`

If you only start to learn React, you needn't it.

From <https://medium.com/@daniel.hramkov/browserrouter-vs-hashrouter-e8bf1c3824ce>

Hashrouter can take three properties
Basename, getUserConfirmation and hashType

hashType

The type of encoding to use for `window.location.hash`. Available values are:

- `"slash"` - Creates hashes like `#/` and `#/main/profile`
- `"noslash"` - Creates hashes like `#` and `#main/profile`
- `"hashbang"` - Creates "ajax crawlable" (deprecated by Google) hashes like `#!/` and `#!/main/profile`

Defaults to `"slash"`.

From <https://medium.com/@daniel.hramkov/browserrouter-vs-hashrouter-e8bf1c3824ce>

Friday, March 11, 2022 10:13 AM

Three basic hooks

1.useState
2.useEffect
3.useContext

Classbased	Functional based

`import React, { Component } from "react";`
`export default class ClassBasedComponent extends Component {`
`  state = {`
`  title: window.document.title,`
`  };`
`  componentDidMount() {`
`  this.setState({ title: "new title from class" });`
`  }`
`  render() {`
`  return (`
`  <div>`
`  <h3>ClassBasedComponent</h3>`
`  <p>{this.state.title}</p>`
`  </div>`
`  );`
`  }`
`}`
	`import React, { useEffect, useState } from "react";`
	`const FunctionalBasedComponent = () => {`
	`  let [state, setState] = useState(document.title);`
	`  useEffect(() => {`
	`  setState("this is rendering from functional based");`
	`  }, []);`
	`  return (`
	`  <div>`
	`  <h3>Functional Comp</h3>`
	`  <p>{state}</p>`
	`  </div>`
	`  );`
	`};`
	`export default FunctionalBasedComponent;`

Classbased	Functional

`import React, { Component } from "react";`
`export default class ClassBasedComponent extends Component {`
`  state = {`
`  count: 0,`
`  };`
`  Increment = () => {`
`  this.setState({ count: this.state.count + 1 });`
`  };`
`  componentDidMount() {`
`  window.document.title = `you clicked ${this.state.count}`;`
`  console.log("Please check how many times rendered");`
`  }`
`  componentDidUpdate() {`
`  window.document.title = `you clicked ${this.state.count}`;`
`  console.log("Please check how many times rendered");`
`  }`
`  render() {`
`  return (`
`  <div>`
`  <h3>ClassBasedComponent</h3>`
`  <p>{this.state.count}</p>`
`  <button onClick={this.Increment}>Increment</button>`
`  </div>`
`  );`
`  }`
`}`
	`import React, { useEffect, useState } from "react";`
	`const FunctionalBasedComponent = () => {`
	`  let [count, setCount] = useState(0);`
	`  let Increment = () => {`
	`  setCount(count++);`
	`  };`
	`  useEffect(() => {`
	`  window.document.title = `you clicked ${this.state.count}`;`
	`  console.log("Please check how many times rendered");`
	`  });`
	`  return (`
	`  <div>`
	`  <h3>Functional Comp</h3>`
	`  <p>{count}</p>`
	`  <button onClick={Increment}>Increment</button>`
	`  </div>`
	`  );`
	`};`
	`export default FunctionalBasedComponent;`

renders the entire DOM only once, provides control	`import React, { useEffect, useState } from "react";` `const FunctionalBasedComponent = () => {`

```
  let [count, setCount] = useState(0);
  let [width, setWidth] = useState(window.innerWidth);
  let Increment = () => {
    setCount(count++);
  };
  useEffect(() => {
    window.document.title = `you clicked ${this.state.count}`;
    console.log("Please check how many times rendered");
    window.addEventListener("resize", () => {
      setWidth(window.innerWidth);
    });
  },[]);
  return (
    <div>
      <h3>Functional Comp</h3>
      <p>{count}</p>
      <button onClick={Increment}>Increment</button>
    </div>
  );
};
export default FunctionalBasedComponent;
```

What is useEffect in react?

What does useEffect do? By using this Hook, **you tell React that your component needs to do something after render**. React will remember the function you passed (we'll refer to it as our “effect”), and call it later after performing the DOM updates.

Why is useEffect used? (Interview question)

The motivation behind the introduction of useEffect Hook is **to eliminate the side-effects of using class-based components**. For example, tasks like updating the DOM, fetching data from API end-points, setting up subscriptions or timers, etc can be lead to unwarranted side-effects.

Evening and next day sir is doing weather API

Whenever you have form got for controlled component and custom method handleSubmit

useEffect( )
Note: it takes a callback function as a parameter.
useEffect( ) replaces all the below three methods from Lifecycle

1. componentDidMount( )
2. componentDidUpdate( )
3. componentWillUnmount( )

useEffect( )

1. Will be called after the Renders.
2. It mainly helps in SMART RENDERING.
3. It does not allow the unnecessary triggering.

HOC

Monday, March 14, 2022 9:48 AM

What is HOC?
A higher-order component (HOC) is **an advanced technique in React for reusing component logic**.
HOCs are not part of the React API, per se. They are a pattern that emerges from React's compositional nature. Concretely, a higher-order component is a function that takes a component and returns a new component.

HOC.jsx

```jsx
import React, { Component } from "react";
const Hoc = WrappedComponent => {
  class Jio extends Component {
    state = {
      network: "jio",
      price: 400,
      validity: "1month",
    };
    render() {
      return <WrappedComponent {...this.state} />;
    }
  }
  return Jio;
};
export default Hoc;
```

App.jsx

```jsx
import React from "react";
import User from "./Components/User";
const App = () => {
  return (
    <div>
      <User  />
    </div>
  );
};
export default App;
```

User.jsx

```jsx
import React from "react";
import Hoc from "./Hoc";
const User = props => {
  return (
    <>
      {props.network},<br />
      {props.validity}
    </>
  );
};
export default Hoc(User);
```

Another way to write same HOC.jsx

Hoc.jsx

```jsx
import React, { useState } from "react";
const Hoc = WrappedComponent => {
  let Jio = () => {
    let [state, setState] = useState({
      network: "jio",
      price: 400,
      validity: "1month",
    });
    return <WrappedComponent {...state} />;
    };
    return Jio;
};
export default Hoc;
```

A HOC order Component is a function that takes a componet and returns a new component.

What is withRouter hoc?

withRouter is **a higher-order component provided by react-router-dom which gives you access to history , match and location object from props of particular object which is wrapped with withRouter** .

WithRoute.jsx

```jsx
import React from "react";
import { BrowserRouter as Router,
Switch, Route } from "react-router-dom";
import User from "./User";
import User1 from "./User1";
const WithRoute = () => {
  return (
    <div>
      <Router>
        <Switch>
          <Route path="/" exact>
            <User />
          </Route>
          <Route path="/user1" exact>
            <User1 />
          </Route>
        </Switch>
      </Router>
```

User.jsx

```jsx
import React from "react";
import { withRouter } from "react-router-dom";
const User = props => {
  console.log(props);
  let RedirectToHome = () => {
    props.history.push("/user1");
  };
  return (
    <div>
      List of users
      <button onClick={RedirectToHome}>Send</button>
    </div>
  );
};
export default withRouter(User);
```

User1.jsx

```jsx
import React from "react";
import { withRouter } from "react-
router-dom";
const User1 = props => {
  return <div>User1</div>;
};
export default withRouter(User1);
```

```
    </div>
  );
};
export default WithRoute;
```

==========

HOC:

* HOC is an advanced technique for reusing component logic.
* HOC are common in third-party libraries.
* It is a function that takes a component and returns a new component.
* A component transforms props into UI, a higher-order component transforms a component into another component.
* A higher order component function accepts another function as an argument.

Ex: map function.

Syntax: const NewComponent = higherOrderComponent(WrappedComponent);

Note:

-> Do not use HOCs inside the render method of a component.

-> HOCs does not work for refs as 'Refs' does not pass through as a parameter or argument.

-> The primary use of Higher-Order Component is to enhance the reusability of particular components in multiple modules or components.

Context API

Monday, March 14, 2022 11:25 AM

What is Context ?

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

In a typical React application, data is passed top-down (parent to child) via props, but such usage can be cumbersome for certain types of props (e.g. locale preference, UI theme) that are required by many components within an application. Context provides a way to share values like these between components without having to explicitly pass a prop through every level of the tree.

When to Use Context

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language. For example, in the code below we manually thread through a “theme” prop in order to style the Button component:

From <https://reactjs.org/docs/context.html>

React.createContext

```
constMyContext =React.createContext(defaultValue);
```

Creates a Context object. When React renders a component that subscribes to this Context object it will read the current context value from the closest matching Provider above it in the tree.

From <https://reactjs.org/docs/context.html>

Context.Provider

```
<MyContext.Providervalue={/* some value */}>
```

Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.

The Provider component accepts a value prop to be passed to consuming components that are descendants of this Provider. One Provider can be connected to many consumers. Providers can be nested to override values deeper within the tree.

All consumers that are descendants of a Provider will re-render whenever the Provider’s value prop changes. The propagation from Provider to its descendant consumers (including .contextType and useContext) is not subject to the shouldComponentUpdate method, so the consumer is updated even when an ancestor component skips an update.

Changes are determined by comparing the new and old values using the same algorithm as Object.is.
Note
The way changes are determined can cause some issues when passing objects as value: see Caveats.

From <https://reactjs.org/docs/context.html>

contextApi.js	App.js	User.jsx

`import { createContext, useState } from "react";`
`export let ThemeContext = createContext();`
`//set a provider`
`const ThemeProvider = ({ children }) => {`
`let [theme, setTheme] = useState(false);`
`let [state, setState] = useState("dark");`
	`import React from "react";`
	`import ThemeProvider from "./Apis/ContextApi";`
	`import User from "./Components/User";`
	`const App = () => {`
	`return (`
		`import React, { useContext } from "react";`
		`import { ThemeContext } from "../Apis/ContextApi";`
		`const User = () => {`
		`let value = useContext(ThemeContext);`

```
    let updateTheme = () => {
        setTheme(!theme);
        if (theme === true) {
            setState("light");
        } else {
            setState("dark");
        }
    }
    return (
        <ThemeContext.Provider
value={{ state, updateTheme }}>
            {children}
        </ThemeContext.Provider>
    )
}
export default ThemeProvider;
```

```
    <div>
      <ThemeProvider>
        <User />
      </ThemeProvider>
    </div>
  );
};
export default App;
```

```
  console.log(value);
  return (
    <div
      style={
        value.state === "dark"
          ? { background: "#111",
color: "#fff" }
          : { background: "#fff",
color: "#111" }
      }
    >
      {value.state === "dark" ?
"dark" : "white"}
    </div>
  );
};
export default User;
```

Three ways
1. Context-Provider
2. useContext hook way
3. Context-type
4. Context.consumer

Class.contextType

The contextType property on a class can be assigned a Context object created by React.createContext(). Using this property lets you consume the nearest current value of that Context type using this.context. You can reference this in any of the lifecycle methods including the render function.
Note:
You can only subscribe to a single context using this API. If you need to read more than one see Consuming Multiple Contexts.

If you are using the experimental public class fields syntax, you can use a **static** class field to initialize your contextType.
classMyClassextendsReact.Component{staticcontextType =MyContext;render(){letvalue =this.context;/* render something based on the value */}}

From <https://reactjs.org/docs/context.html>

Context.Consumer

```
<MyContext.Consumer>
  {value=>/* render something based on the context value */}
</MyContext.Consumer>
```

A React component that subscribes to context changes. Using this component lets you subscribe to a context within a function component.

Requires a function as a child. The function receives the current context value and returns a React node. The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree. If there is no Provider for this context above, the value argument will be equal to the defaultValue that was passed to createContext().
Note
For more information about the 'function as a child' pattern, see render props.

From <https://reactjs.org/docs/context.html>

Context.displayName

It is used for debugging
It helps in identifying which component belongs to whom

Context object accepts a displayName string property. React DevTools uses this string to determine what to display for the context.

For example, the following component will appear as MyDisplayName in the DevTools:

```
constMyContext =React.createContext(/* some value */);
MyContext.displayName ='MyDisplayName';
<MyContext.Provider> // "MyDisplayName.Provider" in DevTools<MyContext.Consumer> // "MyDisplayName.Consumer" in DevTools
```

From <https://reactjs.org/docs/context.html>

Ma'am examples

Eg1

```
import React, { createContext, Component } from "react";
const MyContext = createContext();
class Provider extends Component {
  state = {
    place: "Goa",
  };
  render() {
    return (
      <>
        <MyContext.Provider value={this.state.place}>
          {this.props.children}
        </MyContext.Provider>
      </>
    );
  }
}
// child 2 : (consumer)
let Trial = props => {
  return (
    <>
      <MyContext.Consumer>
        {allcontext => {
          return <h2>My First salary trip is to {allcontext}
</h2>;
        }}
      </MyContext.Consumer>
    </>
  );
};
// child 1
let Lift = props => {
  return <Trial />;
};
// parent
export default class ResortNew extends Component {
  render() {
    return (
      <>
        <Provider>
          <Lift />
        </Provider>
      </>
    );
  }
}
```

Eg2

```
import React, { Component } from "react";
// child 2 : (consumer)
let Trial = props => {
  return <h2>My Destination place is {props.myplace}
</h2>;
};
// child 1
let Lift = props => {
  return <Trial myplace={props.name} />;
};
// parent
export default class Resort extends Component {
  state = {
    place: "Goa",
  };
  render() {
    return (
      <>
        <Lift name={this.state.place} />
      </>
    );
  }
}
```

==========

React Context API:

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

React.createContext
 const MyContext = React.createContext(defaultValue)

Context.Provider
 <MyContext.Provider value = { some value }>

Context.Consumer
 <MyContext.Consumer>

```
{ value => /* render something based on the context value */ }
</MyContext.Consumer>
```

Error Boundaries

Wednesday, March 16, 2022 4:24 PM

Error boundaries very important interview question

Introducing Error Boundaries

A JavaScript error in a part of the UI shouldn't break the whole app. To solve this problem for React users, React 16 introduces a new concept of an "error boundary".

Error boundaries are React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

From <https://reactjs.org/docs/error-boundaries.html>

Note
Error boundaries do **not** catch errors for:

- Event handlers (learn more)
- Asynchronous code (e.g. setTimeout or requestAnimationFrame callbacks)
- Server side rendering
- Errors thrown in the error boundary itself (rather than its children)

From <https://reactjs.org/docs/error-boundaries.html>

It only works in class based components

Error boundaries work like a JavaScript catch {} block, but for components. Only class components can be error boundaries. In practice, most of the time you'll want to declare an error boundary component once and use it throughout your application.

From <https://reactjs.org/docs/error-boundaries.html>

Axios

Thursday, March 17, 2022 2:34 PM

Axios is a promise based HTTP Client for the browser and Nodejs.
Axios makes it easy to send asynchronous HTTP requests to REST endpoints and perform CRUD operations. It can be used in plain JavaScript ow with a library such as Vue or React.

Axios is a JS library used to make HTTP requests from node.js or XMLHTTPRequests or AJAX requests from the browserr ans it supports the promise Api that is native to JS ES6.

How to use Axios library?
https://jsonplaceholder.typicode.com/
Step1: install axios => npm install axios or yarn add axios
Step2: Create an instance of axios (but not mandatory)
Inside src=> create a folder libraries => create a file Axios.js

How to create an instance? (interview question)

```
export default axios.create()
```

Full code Axios.js

```
import axios from "axios";
export default axios.create({
  baseURL: "https://jsonplaceholder.typicode.com/",
  headers: {
    "content-type": "application/json",
  },
});
```

Now go to routing install router
Step 4:Yarn add react-router-dom@latest

How to consume axios

```
import React, { useEffect, useState } from "react";
import Axios from "../libraries/Axios";
const Employees = () => {
  let [emp, setEmp] = useState([]);
  let [loading, setLoading] = useState(false);
  useEffect(() => {
    const fetchData = async () => {
      try {
        let body = await Axios.get("/users");
        let { data } = body;
        setEmp(data);
        setLoading(true);
      } catch (error) {
        console.log(error);
      }
    };
    fetchData();
  }, []);
  return <div>Employees</div>;
};
export default Employees;
```

Employee.jsx

```
import React, { useEffect, useState, Fragment } from "react";
import Axios from "../libraries/Axios";
import Spinner from "./Spinner";
const Employees = () => {
  let [emp, setEmp] = useState([]);
  let [loading, setLoading] = useState(false);
  useEffect(() => {
    const fetchData = async () => {
      try {
        let body = await Axios.get("/users");
        let { data } = body;
```

```
        setEmp(data);
        setLoading(true);
      } catch (error) {
        console.log(error);
      }
      setLoading(false);
    };
    fetchData();
  }, []);
  return (
    <Fragment>
      {loading ===true ? (
        <Spinner />
      ) : (
        emp.map(user => {
          return (
            <Fragment key={user.id}>
              <h2>{user.name}</h2>
            </Fragment>
          );
        })
      )}
    </Fragment>
  );
};
export default Employees;
```

FAKE JSON SERVER

Npm install -g json-server
Create fake backend servers
Step1: in root folder create a folder called backend or db in root
Step 2: inside backend create a file called employee.json
Step3: create entries

```
{
  "employess": [{ "id": 1, "title": "json-server", "author": "bhuvan" }]
}
```

Step 4: use the command in backend folder through cmd

Command==>json-server --watch employee.json --port=5000

```
C:\Users\bhuvan\Desktop\MERN\React\Projects\axios-crud\backend>json-server --watch employee.json --port=5000

  \{^_^}/ hi!

  Loading employee.json
  Done

  Resources
  http://localhost:5000/employess

  Home
  http://localhost:5000

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

Employees.jsx

```
import React, { useEffect, useState, Fragment } from "react";
import Axios from "../libraries/Axios";
import Spinner from "./Spinner";
const Employees = () => {
  let [emp, setEmp] = useState([]);
  let [loading, setLoading] = useState(false);
  useEffect(() => {
    const fetchEmp = async () => {
```

```
      try {
        let payload = await Axios.get("/employees");
        let { data } = payload;
        setEmp(data);
        setLoading(true);
      } catch (error) {
        console.log(error);
      }
      setLoading(false);
    };
    fetchEmp();
  }, []);
  return (
    <section id="empTable">
      <article>
        <table>
          <thead>
            <tr>
              <th>emp id</th>
              <th>emp name</th>
              <th>Designation</th>
            </tr>
          </thead>
          <tbody>
            {loading ? (
              <Spinner />
            ) : (
              emp.map(user => {
                return (
                  <tr key={user.id}>
                    <td>{user.id}</td>
                    <td>{user.name}</td>
                    <td>{user.designation}</td>
                  </tr>
                );
              })
            )}
          </tbody>
        </table>
      </article>
    </section>
  );
};
export default Employees;
```

Createemp.jsx

```
import React, { useState } from "react";
const CreateEmp = () => {
  let [state, setState] = useState({
    name: "",
    designation: "",
    loading: false,
  });
  let { name, designation, loading } = state;
  let handleChange = e => {
    let { name, value } = e.target;
    setState({ ...state, [name]: value });
  };
  let handleSubmit = e => {
    e.preventDefault();
    try {
      console.log({ name, designation });
    } catch (error) {
      console.log(error);
    }
  };
  return (
    <section id="authBlock">
      <article>
        <div>
          <h2>Create Emp</h2>
          <form onSubmit={handleSubmit}>
```

```
            <div className="form-group">
              <label htmlFor="emp_name">emp name</label>
              <input
                type="text"
                name="name"
                id="emp_name"
                value={name}
                required
                onChange={handleChange}
              />
            </div>
            <div className="form-group">
              <label htmlFor="emp_dest">Designation</label>
              <input
                type="text"
                name="designation"
                id="emp_dest"
                value={designation}
                required
                onChange={handleChange}
              />
            </div>
            <div className="form-group">
              <button>create employee</button>
            </div>
          </form>
        </div>
      </article>
    </section>
  );
};
export default CreateEmp;
```

CREATEMP.JSX

```
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import Axios from "../libraries/Axios";
const CreateEmp = () => {
  let navigate = useNavigate();
  let [state, setState] = useState({
    name: "",
    designation: "",
    loading: false,
  });
  let { name, designation, loading } = state;
  let handleChange = e => {
    let { name, value } = e.target;
    setState({ ...state, [name]: value });
  };
  let handleSubmit = async e => {
    e.preventDefault();
    try {
      setState({ loading: true });
      let payload = { name, designation };
      await Axios.post("/employees", payload);
      navigate("/");
    } catch (error) {
      console.log(error);
    }
    setState({ loading: false, name: "", designation: "" });
  };
  return (
    <section id="authBlock">
      <article>
        <div>
          <h2>Create Emp</h2>
          <form onSubmit={handleSubmit}>
            <div className="form-group">
              <label htmlFor="emp_name">emp name</label>
              <input
                type="text"
                name="name"
```

```
                id="emp_name"
                value={name}
                required
                onChange={handleChange}
              />
            </div>
            <div className="form-group">
              <label htmlFor="emp_dest">Designation</label>
              <input
                type="text"
                name="designation"
                id="emp_dest"
                value={designation}
                required
                onChange={handleChange}
              />
            </div>
            <div className="form-group">
              <button>
                {loading === true ? "loading..." : "Create Employee"}
              </button>
            </div>
          </form>
        </div>
      </article>
    </section>
  );
};
export default CreateEmp;
```

Yarn add react-icons

==========

Axios:

The Axios is a Promise based HTTP client for the browser and node.js

The AXIOS HTTP Client Request Parameters
baseUrl : This is a base URL, it'll be pre-pended to any relative URL.
headers : This is an object of key/value pairs to be sent as headers.
params : This contains an object of key/value pairs that will be serialized and appended to the URL as a query string.
responseType : Defined response type in a format other than JSON.
auth : This is the HTTP Basic auth string which will send with each http request.
method : HTTP methods.

The Response Object of Axios
data : The payload returned from the server.
status : The HTTP code returned from the server.
statusText : The HTTP status message returned by the server.
headers : All the headers sent back by the server.
config : The original request configuration.
request : The actual XMLHttpRequest object.(edited)

netflix

Monday, March 21, 2022 2:16 PM

Yarn add react-router-dom react-toastify moment react-moment jquery axios firebase react-icons

Day 2
Used navlink instead of link

Navlink	Link
Used to navigate the different rooutes on the site. We can use active classname here	But navlink is used to add the style attributes to the active routes.

useLocation hook- is used to get the path name
This hook returns the location object used by the react-router. This object represents the Current url and is immutable. Whenever the URL changes, the useLocation() hook returns a newly updated location object.

Based on path conditional rendering

```
<section
      id={Styles.navbarBlock}
      className={location.pathname === "/" ? Styles.homeClass : ""}>
```

Toastify message

App.jsx

```
import React from "react";
import { BrowserRouter as
Router } from "react-router-
dom";
import Navbar from
"./pages/navbar/Navbar";
import StreamBaseRoutes from
"./routes/StreamBaseRoutes";
import { ToastContainer }
from "react-toastify";
import "react-
toastify/dist/ReactToastify.
css";
const App = () => {
  return (
    <Router>
      <header>
        <Navbar />
      </header>
```

Signup.jsx

```
import React, { useState } from "react";
import Styles from "./auth.module.css";
import { toast } from "react-toastify";
const Signup = () => {
  let [username, setUsername] = useState("");
  let [email, setEmail] = useState("");
  let [password, setPassword] = useState("");
  let [confirmPassword, setConfirmPassword] =
useState("");
  let [loading, setLoading] = useState(false);
  let handleSubmit = e => {
    e.preventDefault();
    try {
      setLoading(true);
      if (password !== confirmPassword) {
        toast.error("password does not match");
      } else {
        console.log({ username, email,
password });
```

```
      <main>
        <ToastContainer
pauseOnHover />
        <StreamBaseRoutes />
      </main>
    </Router>
  );
};
export default App;
```

```
      }
    } catch (error) {
      console.log(error);
    }
    setLoading(false);
    setUsername("");
    setEmail("");
    setPassword("");
    setConfirmPassword("");
  };
  return (
    <section id={Styles.authSection}>
      <article className={Styles.authArticle}>
        <h2 style={{ padding: "10px 0" }}>
Signup</h2>
        <div className={Styles.formBlock}>
          <form onSubmit={handleSubmit}>
            <div className="form-group">
              <label htmlFor="username"
className={Styles.formLabel}>
                username
              </label>
              <input
                type="text"
                className={Styles.formControl}
                value={username}
                onChange={e =>
setUsername(e.target.value)}
              />
            </div>
            <div className="form-group">
              <label htmlFor="email"
className={Styles.formLabel}>
                email
              </label>
              <input
                type="email"
                className={Styles.formControl}
                value={email}
                onChange={e =>
setEmail(e.target.value)}
              />
            </div>
            <div className="form-group">
              <label htmlFor="password"
className={Styles.formLabel}>
                password
              </label>
              <input
                type="password"
                className={Styles.formControl}
                value={password}
                onChange={e =>
setPassword(e.target.value)}
              />
            </div>
            <div className="form-group">
              <label htmlFor="confirm password"
```

```
className={Styles.formLabel}>
                confirm password
              </label>
              <input
                type="password"
                className={Styles.formControl}
                value={confirmPassword}
                onChange={e =>
setConfirmPassword(e.target.value)}
              />
            </div>
            <div className="form-group">
              <button className={Styles.btn}>
                {loading ? "loading..." :
"Signup"}
              </button>
            </div>
          </form>
        </div>
      </article>
    </section>
  );
};
export default Signup;
```

What is fire base used for?
Google Firebase is a Google-backed application development software that enables developers to develop iOS, Android and Web apps. Firebase provides tools for tracking analytics, reporting and fixing app crashes, creating marketing and product experiment.

Go to google sign in

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries
// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyBZ4x8H1YwWVQkxLRrtyq9Ds4fsSd-l9rI",
  authDomain: "streambase-36423.firebaseapp.com",
  projectId: "streambase-36423",
  storageBucket: "streambase-36423.appspot.com",
  messagingSenderId: "933305670020",
  appId: "1:933305670020:web:21ed5a8aa225ffbfffe5d1",
};
// Initialize Firebase
 const firebase = initializeApp(firebaseConfig);
export default firebase;
```

Go to authentication click on get started

Enable email
And put this in above code

```
import { getAuth } from "firebase/auth";
```

And export it as a local instance

```
export let auth = getAuth(firebase);
```

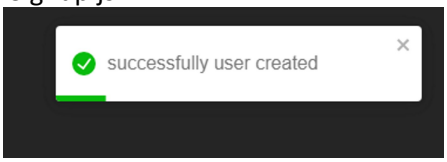
Go to signup.jsx and import them

```
import firebase,{ auth } from "../../api/firebase";
```

```
import { createUserWithEmailAndPassword } from "firebase/auth";
```

Now use these methods

Signup.jsx




```
let handleSubmit = async e => {
    e.preventDefault();
    try {
      setLoading(true);
      if (password !== confirmPassword) {
        toast.error("password does not match");
      } else {
        let userData = await
createUserWithEmailAndPassword(auth, email,
password);
        toast.success(`successfully user
created`);
        console.log(userData);
      }
    } catch (error) {
      console.log(error);
    }
```

Custom hooks

Wednesday, March 30, 2022 4:44 PM

useFetch.js

```
import { useEffect, useState } from "react";
const useFetch = url => {
  let [state, setState] = useState([]);
  let [loading, setLoading] = useState(false);
  useEffect(() => {
    let fetchData = async () => {
      let value = await window.fetch(url);
      let finalValue = await value.json();
      setState(finalValue);
      setLoading(true);
    };
    fetchData();
  }, [url]);
  return { state, loading };
};
export default useFetch;
```

App.jsx

```
import React, { Fragment } from "react";
import useFetch from "./customhook/useFetch";
const App = () => {
  let { state, loading } = useFetch(
    `https://jsonplaceholder.typicode.com/users`
  );
  console.log(state, loading);
  return (
    <div>
      {loading !== true
        ? "loading.."
        : state.map(x => {
            return (
              <Fragment key={x.id}>
                <li>{x.name}</li>
                <li>{x.email}</li>
              </Fragment>
            );
          })}
    </div>
  );
};
export default App;
```

Here useFetch is the custom hook we created

React 18

Wednesday, March 30, 2022 4:48 PM

Index.js

```
import ReactDOM from "react-dom/client";
import App from "./App";
ReactDOM.createRoot(document.getElementById("root")).render(<App />);
```

ForwardREF

Wednesday, March 30, 2022 4:39 PM

React forwardRef is **a method that allows parent components pass down (i.e., "forward") refs to their children**. Using forwardRef in React gives the child component a reference to a DOM element created by its parent component. This then allows the child to read and modify that element anywhere it is being used.

From <https://www.google.com/search?q=forward+ref+react+descriptions&rlz=1C1JJTC_enIN971IN971&oq=forward+ref+react+descriptions&aqs=chrome..69i57j33i10i160l2.8916j0j4&sourceid=chrome&ie=UTF-8>

Forwarding Refs

Ref forwarding is a technique for automatically passing a ref through a component to one of its children. This is typically not necessary for most components in the application. However, it can be useful for some kinds of components, especially in reusable component libraries. The most common scenarios are described below.

From <https://reactjs.org/docs/forwarding-refs.html>

App.jsx

```
import React from 'react'
import Login from
'./fwdREf/Login'
const Appfwdref = () => {
    return (
        <div><Login/></div>
    )
}
export default Appfwdref
```

Input.jsx

```
import React,{useRef,useEffect} from 'react';
import Login from './Login';
const Input = () => {
    let InputRef = useRef();
    let placeholder = "add some text";
    let type = "text";
    useEffect(() => {
        InputRef.current.focus();
    }, []);
    return (
        <div>
            <Login attr={{ placeholder, type }}
ref={Input}/>
        </div>
    )
}
export default Input
```

Login.jsx

```
import React,{forwardRef} from 'react'
const Login = forwardRef((props, ref) =>
{
    console.log(props);
    return (
        <div>
            <form>
                <input {...props.attr}
ref={ref} />
            </form>
        </div>
    )
});
export default Login;
```

videoPlayer.jsx

```
import React,{forwardRef} from 'react'
const VideoPlayer = React.forwardRef((props, ref) => {
    let { VIDEO, handlePlay } = props.value;
    return (
        <div>
            <video src={VIDEO} ref={ref}
onClick={handlePlay}></video>
        </div>
    )
});
export default VideoPlayer
```

videoComponnet.jsx

```
import React,{useRef,useState} from 'react'
import VIDEO from '../components/arabic_kuttu.mp4'
import VideoPlayer from '../videofwdref/VideoPlayer'
const VideoComponent = () => {
    let videoRef = useRef();
    let [play, setPlay] = useState(false);
    let handlePlay = () => {
        setPlay(!play);
        if (play === true) {
            videoRef.current.play();
        } else {
            videoRef.current.pause();
        }
```

```
    }
    return (
        <div>
            <VideoPlayer value={{ VIDEO, handlePlay }}
ref={videoRef} />
        </div>
    );
}
export default VideoComponent
```

Code spilting

Wednesday, March 30, 2022 5:25 PM

Webpack is a popular module bundling system built on top of Node.js. It can handle not only combination and minification of JavaScript and CSS files, but also other assets such as image files (spriting) through the use of plugins.

Do I need webpack With React?

Well, **we don't necessarily need webpack to work with React**, other alternatives could be Browserify, Parsel, Brunch, etc, but honestly, I don't know how well they fit in with React. js. Webpack is the most widely used and an accepted module bundler and task runner throughout React. js community.

Webpack alternatives

1. Browserify
2. Parsel
3. Brunch

Code-Splitting

Bundling

Most React apps will have their files “bundled” using tools like Webpack, Rollup or Browserify. Bundling is the process of following imported files and merging them into a single file: a “bundle”. This bundle can then be included on a webpage to load an entire app at once.

App.jsx

```
import React, { Suspense } from 'react';
let VideoComponent = React.lazy(() =>
import('./videofwdref/VideoComponent'));
const Appvideo = () => {
    return (
        <div>
            <Suspense fallback={<div>
loading...</div>}>
                <VideoComponent/>
            </Suspense>
        </div>
    )
}
export default Appvideo
```

videoComponent.jsx

```
import React,{useRef,useState} from
'react'
import VIDEO from
'../components/arabic_kuttu.mp4'
import VideoPlayer from
'../videofwdref/VideoPlayer'
const VideoComponent = () => {
    let videoRef = useRef();
    let [play, setPlay] =
useState(false);
    let handlePlay = () => {
        setPlay(!play);
        if (play === true) {
            videoRef.current.play();
        } else {
            videoRef.current.pause();
        }
    }
    return (
        <div>
            <VideoPlayer
value={{ VIDEO, handlePlay }}
ref={videoRef} />
        </div>
    );
}
export default VideoComponent
```

VideoPlayer.jsx

```
import React,{forwardRef} from
'react'
const VideoPlayer =
React.forwardRef((props, ref) => {
    let { VIDEO, handlePlay } =
props.value;
    return (
        <div>
            <video src={VIDEO}
ref={ref} onClick={handlePlay}>
</video>
        </div>
    )
});
export default VideoPlayer
```

useReducer()

Wednesday, March 30, 2022 5:57 PM

```
import React, { useReducer } from "react";
let initialState = { count: 0 };
let reducer = (state, action) => {
  switch (action.type) {
    case "increment":
      return { count: state.count + 1 };
    case "decrement":
      return { count: state.count - 1 };
    case "reset":
      return { count: 0 };
    default:
      new Error("error");
  }
};
const App = () => {
  let [state, dispatch] = useReducer(reducer, initialState);
  let Increment = () => {
    dispatch({ type: "increment" });
  };
  let decrement = () => {
    dispatch({ type: "decrement" });
  };
  let reset = () => {
    dispatch({ type: "reset" });
  };
  return (
    <div>
      <h1>{state.count}</h1>
      <button onClick={Increment}>increment</button>
      <button onClick={decrement}>decrement</button>
      <button onClick={reset}>reset</button>
    </div>
  );
};
export default App;
```

Pure Component

Friday, April 8, 2022 5:14 PM

Pure components
Def: A pure component implements shouldCompoentUpdate witb a Shallow props and state comparison.

Note:

- In SC if there us a dufference then the component will re-render.
- If a ParentComponet is a pure component the the component doesn't re-render.

Pure component always depends on shouldComponentUpdate method and if there is no change in the state it does not re-render where as Regular component will re-render.

Ma'am sent==========
Pure Components: second way to create class component i.e extending the pure componentsDef: A Pure Component implements shouldComponentUpdate with a shallow
prop & state comparison SC of prevState with currentState
SC of prevProps with currentPropsNote:
1.In SC if there is a difference then the component will re-render
2.If a ParentComponent is a PureComponent then the component doesn't re-renderWhy PC:
prevents unnecessary render give performance boost.
Never mutate Object or Array in props or state.
Always return a new object or an array which reflects the new state.
If there is no difference the component is not re-rendered - performance boostDifference b/w Component & pure componentRegular Component: doesn't implement the shouldComponentUpdate method.
it always returns true by default.Pure Component: implements shouldComponentUpdate with shallow props & state comparison.

5:26

```
import React, { PureComponent } from "react";
import PureComponents from './PureComponents';
import RegularComponent from './RegularComponent';export default class ParentComponent extends PureComponent {
 constructor(props) {
  super(props);    this.state = {
   name: "Rohan",
  };
 }  componentDidMount() {
  setTimeout(() => {
   this.setState({
    name: "Rohan",
   });
  }, 3000);
 }  render() {
  console.log("******ParentComponent******");
  return (
   <div>
    Parent Component
    {/* sending a name as a prop for Regular & Pure Components  */}
    <RegularComponent name={this.state.name} />
    <PureComponents name={this.state.name} />
   </div>
  );
 }
}
```

5:26

```
import React, { PureComponent } from 'react'export default class PureComponents extends PureComponent {
  render() {
    console.log("******PureComponent******");
    return (
```

```
        <div>
          Pure Component <b>{this.props.name}</b>
        </div>
      )
    }
}
```

5:27

```
import React, { Component } from 'react'export default class RegularComponent extends Component {
    render() {
      console.log("******RegularComponent******");
      return (
        <div>
          Regular Component <b>{this.props.name}</b>
        </div>
      )
    }
}
```

From <https://app.slack.com/client/T03AB8H7KDH/C03A8CCJ56Z>

hosting

Tuesday, April 5, 2022 4:46 PM

Npm run build
Go to firebase console > go to hosting

From hosting go to

And install npm install -g firebase-tools

Now type firebase login

Redux

Saturday, April 9, 2022 10:06 AM

Redux

Props drilling
And context API allows UNI directiona data flow

Context API can be used for medium sized application.

- Redux is not part of react js it is a third party library.
- Redux is a predictable state container for JS Apps. Or provides state management.
- It is similar to to Context API.
- You achieve centralized state

Redux must be used when we are creating huge or complex application.
Redux is very strong in state management.

What is the use of redux?

Redux can be used as a data store for any UI layer. The most common usage is with React and React Native, but there are bindings available for Angular, Angular 2, Vue, Mithril, and more. Redux simply provides a subscription mechanism which can be used by any other code.

Redux provides a few terminologies

Action creators ==========

Actions are pure JS objects.
Reducer is a pure JS function.
Data is called as store, this is also a JS object.

Redux Data flow

Store(js object) is a global state container.

Redux Data flow
UI=> action creator => actions=> dispatch => reducer => store => UI

Redux data flow or Redux Architecture

What is redux action?
Redux is a state managing library used in JavaScript apps. It simply manages the state of your application or in other words, it is used to manage the data of the application. It is used with a library like React. Uses: It makes easier to manage state and data. As the complexity of our application increases.

It carries a payload of information from your application to store.
actions are plain JavaScript object that must have a type attribute to indicate the type of action performed.

From <https://www.tutorialspoint.com/redux/redux_actions.htm>

What is redux reducer?
In Redux, a reducer is a pure function that takes an action and the previous state of the application and returns the new state. The action describes what happened and it is the reducer's job to return the new state based on that action.

What is store in redux?
A store is an immutable object tree in Redux. A store is a state container which holds the application's state. Redux can have only a single store in your application. Whenever a store is created in Redux, you need to specify the reducer. Let us see how we can create a store using the createStore method from Redux.

Yarn add redux react-redux redux-thunk

What is combine reducer?
The combineReducers helper function turns an object whose values are different reducing functions into a single reducing function you can pass to createStore . The resulting reducer calls every child reducer, and gathers their results into a single state object.

counterReducer.js
In reducer folder

```
//?reducer is a pure function
let CounterReducer = (state = 0, action) => {
  switch (action.type) {
    case "Increment":
      return state + 1;
    case "Decrement":
      return state - 1;
    case "Reset":
      return 0;
    default:
      return state;
  }
};
export default CounterReducer;
```

Index.js in reducer folder

```
//!index.js is the root reducer in redux.
import CounterReducer from "./CounterReducer";
import { combineReducers } from "redux";
const rootReducer = combineReducers({ CounterReducer });
export default rootReducer;
```

Store.js in store folder

```
//*create store function exported by redux
import { createStore } from "redux";
import rootReducer from "../reducer/index";
let store = createStore(rootReducer);
export default store;
```

App.jsx

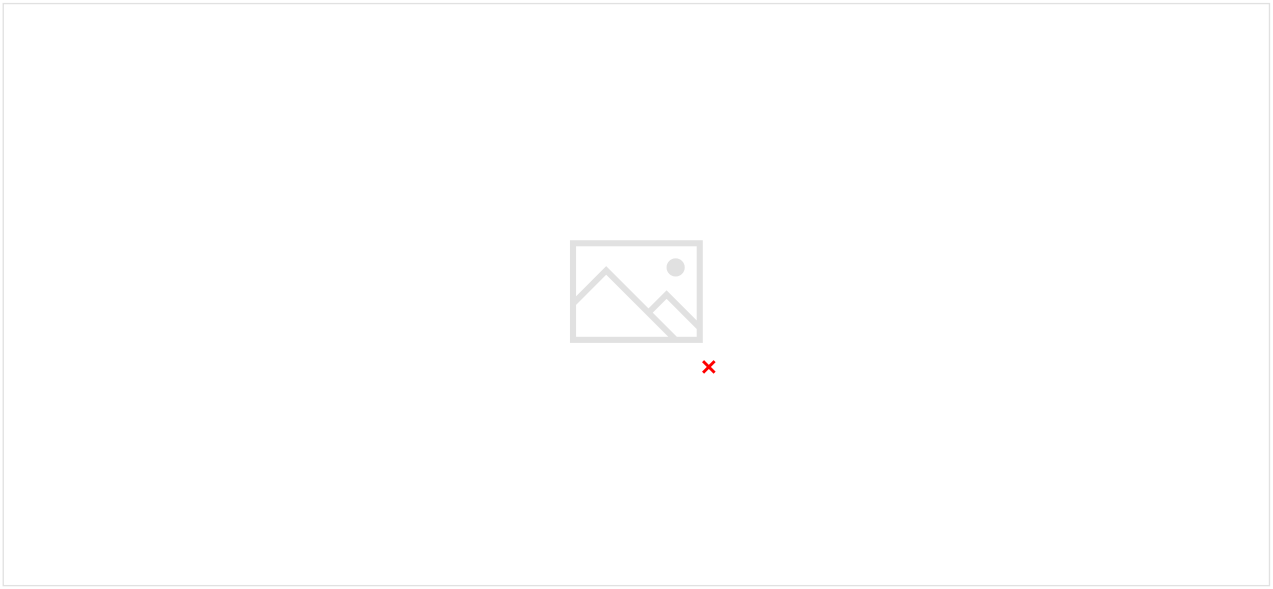
```
import React from "react";
import store from "./redux/store/store";
import { Provider } from "react-redux";
import Counter from "./components/Counter";
const App = () => {
  return (
```

```
    <Provider store={store}>
      <Counter />
    </Provider>
  );
};
export default App;
```

Hooks in redux
1.useSelector()
2.useDispatch()

Counter.jsx

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";
const Counter = () => {
  let state = useSelector(state => state);
  let dispatch = useDispatch();
  console.log(state);
  return (
    <div>
      <h1>{state.CounterReducer}</h1>
      <button onClick={() => dispatch("Increment")}>increment</button>
      <button onClick={() => dispatch("Decrement")}>decrement</button>
      <button onClick={() => dispatch("Reset")}>Reset</button>
    </div>
  );
};
export default Counter;
```



How to solve the above issue

Actions are objects they need to be passed as objects

Counter.jsx

```
import React from "react";
import { useSelector, useDispatch } from "react-redux";
const Counter = () => {
  let state = useSelector(state => state);
  let dispatch = useDispatch();
  console.log(state);
  return (
    <div>
      <h1>{state.CounterReducer}</h1>
      <button onClick={() => dispatch({ type: "Increment" })}>increment</button>
      <button onClick={() => dispatch({ type: "Decrement" })}>decrement</button>
      <button onClick={() => dispatch({ type: "Reset" })}>Reset</button>
    </div>
  );
};
export default Counter;
```

For the redux extension to work paste the highlighted code

In store.js

```
//*create store function exported by redux
import { createStore } from "redux";
import rootReducer from "../reducer/index";
let store = createStore(
  rootReducer,
  window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
);
export default store;
```

Evening session
Use crud-api-2 as backend

yarn add axios react-router-dom redux react-redux redux-thunk react-toastify react-moment

npm install --save redux-devtools-extension or use above method

```
import { createStore } from "redux";
import rootReducer from "../reducer";
import { composeWithDevTools } from "redux-devtools-extension";
let store = createStore(
  rootReducer,
  //   window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
  composeWithDevTools()
);
export default store;
```

Advanced Hooks

Tuesday, April 12, 2022 10:31 AM

useMemo():
The React `useMemo` Hook returns a memoized value.
Think of memoization as caching a value so that it does not need to be recalculated.
The `useMemo` Hook only runs when one of its dependencies update.
This can improve performance.

The `useMemo` and `useCallback` Hooks are similar. The main difference is that `useMemo` returns a memoized value and `useCallback` returns a memoized function.

- It keeps cache of previous state value and does not re-render the component unless there is a change in the state value.
- Syntax: useMemo(callback function, [dependency list]);
- useMemo(( )=>{ setNumber(5) }, [Number]);

Priyanka ma'am notes

Thursday, March 24, 2022 4:46 PM

Axios:

The Axios is a Promise based HTTP client for the browser and node.js

The AXIOS HTTP Client Request Parameters
baseUrl : This is a base URL, it'll be pre-pended to any relative URL.
headers : This is an object of key/value pairs to be sent as headers.
params : This contains an object of key/value pairs that will be serialized and appended to the URL as a query string.
responseType : Defined response type in a format other than JSON.
auth : This is the HTTP Basic auth string which will send with each http request.
method : HTTP methods.

The Response Object of Axios
data : The payload returned from the server.
status : The HTTP code returned from the server.
statusText : The HTTP status message returned by the server.
headers : All the headers sent back by the server.
config : The original request configuration.
request : The actual XMLHttpRequest object.

==========

React Context API:

Context provides a way to pass data through the component tree without having to pass props down manually at every level.

React.createContext

```
const MyContext = React.createContext( defaultValue )
```

Context.Provider

```
<MyContext.Provider value = { some value }>
```

Context.Consumer

```
<MyContext.Consumer>
{ value => /* render something based on the context value */ }
</MyContext.Consumer>
```

==========

HOC:

* HOC is an advanced technique for reusing component logic.
* HOC are common in third-party libraries.
* It is a function that takes a component and returns a new component.
* A component transforms props into UI, a higher-order component transforms a component into another component.
* A higher order component function accepts another function as an argument.

Ex: map function.
Syntax: const NewComponent = higherOrderComponent(WrappedComponent);

Note:
-> Do not use HOCs inside the render method of a component.
-> HOCs does not work for refs as 'Refs' does not pass through as a parameter or argument.
-> The primary use of Higher-Order Component is to enhance the reusability of particular components in multiple modules or components.

==========

React Refs:

-> Refs is the shorthand used for references in React. (Similar to keys).
-> Its an attribute which makes it possible to store a reference to particular DOM nodes or React elements.
-> It provides a way to access React DOM nodes or React elements and how to interact with it.
-> It is used when we want to change the value of a child component, without making the use of props.

When to Use Refs :

* Managing focus, text selection, or media playback.
* Triggering imperative animations.
* Integrating with third-party DOM libraries.
* Also used in callbacks

Note: The ref attribute cannot be used on function components because they don't have instances.

Refs can be created by using React.createRef(). It can be assigned to React elements via the ref attribute.

```
Ex:  class MyRef extends React.Component {
  constructor(props) {
  super(props);
  this.callRef = React.createRef();
  this.addingRefInput = this.addingRefInput.bind(this);
  }

  addingRefInput() {
  this.callRef.current.focus();
  }

  render() {
  return (
  <div> <input  type="text" ref={this.callRef}  />
  <input type="button" value="Add text input" onClick={this.addingRefInput} />
  </div>
```

Callback Ref:
-> It gives more control when the refs are set and unset.
-> React allows a way to create refs by passing a callback function to the ref attribute of a component.
Syntax: <input type="text" ref={element => this.callRefInput = element} />

Forward Ref:
-> Ref forwarding is a technique that is used for passing a ref through a component to one of its child components.
-> It can be performed by making use of the React.forwardRef() method.
-> This technique is particularly useful with higher-order components and specially used in reusable component libraries.
Note:
React with useRef() : react hook ref

==========

Controlled Components:

-> A controlled component is bound to a value, and its changes will be handled in code by using event-based callbacks.
-> Controlled components have functions that govern the data passing into them on every onChange event occurs. This data is then saved to state and updated with setState() method.
-> It makes component have better control over the form elements and data.(edited)

- A controlled Component is bound to a value, and its chnges will be handled in code by using event-based-callbacks.
- Controlled components have functions that govern the data passing into them on every onChange event occurs. This data is then saved to state and updated with setState( ) method.

controlled	Uncontrolled
It does not main its internal state	Here, data is controlled by DOM itself.
It accepts its current value as a prop	It uses a ref for their current values.
It allows validation control	It does not allow validation control
It has better control over the form elements and data	It has limited control over the form elements and data.

==========

React Hooks:

-> Hooks are functions that let you “hook into” React state and lifecycle features from function components.
-> Hooks is a function which allows you to add the additional functionality for the react Component.

-> React Hooks a new feature added in react 16.8 version.
-> React Hooks are used in Functional components.
-> React Hooks doesn't work inside the class Components.

Rules of Hooks: (Two rules)

Hooks are JavaScript functions, but they impose two additional rules:
1. Only call Hooks at the top level. Don't call Hooks inside loops, conditions, or nested functions.
2. Only call Hooks from React function components. Don't call Hooks from regular JavaScript functions.

useState(initialState)

1. useState returns a pair : the current state value and a function that lets you update it.
2. It helps to manage the state in functional components

Syntax:
const [stateValue, function that is used to change the state value] = useState({initial value});

useEffect( )

1. useEffect adds the ability to perform side effects from a function component.
2. It serves the same purpose as componentDidMount, componentDidUpdate and componentWillUnmount in 3. React classes, but unified into a single API.

Syntax:
```
useEffect(() => {
//code to be executed
},[ ]);
```

It takes callback function and an dependency List as a parameter.

Note:
1. useEffect will be called after the Renders.
2. It mainly helps in SMART RENDERING.
3. It doesn't allow the unnecessary Triggering.

useContext():

1. hook accepts a context object, i.e. the value that is returned from React.createContext(), and then it returns the current context value for that context.
2. the useContext hook works with the React Context API which is a way to share data deeply throughout your app.

useMemo():

==========

React Forms:

-> React uses forms to allow users to interact with webpage.

1.Adding Forms to React.

```
<form>
 <h1>My Form </h1>
 <input type="text" />
 </form>
```

2.Handling Forms with React.
It mainly depends how we handle the data when it changes or gets submitted.

In HTML,
form data --> DOM

In React,
form data --> Components
(When is handled by components all the data is stored in the form of state)

i) The control change is handled by à onChange attribute
ii) The access to the field is given using the event.target.value.

==========

Styling with React JS:

1.Regular CSS Stylesheets: This involves using separate stylesheets like our conventional way of styling our HTML websites either with CSS.
2. Inline Styling: Inline styles to any React component we want to render. These styles are written as attributes and are passed to the element.
3. CSS Modules: A CSS Module is a CSS file in which all class names and animation names are scoped locally by default.
4.CSS in JS Libraries: Styled Components, Radium etc...
Note:
1.Material UI : https://material-ui.com/
2.React-Bootstrap: https://react-bootstrap.github.io/
3.Bootstrap 5

==========

Keys:

1. A "key" is a special string attribute you need to include when creating lists of elements in React.
2. Keys are used in React to identify which items in the list are changed, updated or deleted.
3. Keys are used to give an identity to the elements in the lists.
4. It is recommended to use a string as a key that uniquely identifies the items in the list.
5. Key is an Identifier for dynamically created Element.

Note:

1. Keys are not same as props, only the method of assigning "key" to a component is same as that of props. .
2. Keys are internal to React and can not be accessed from inside of the component like props.
3. Therefore, we can use the same value we have assigned to the Key for any other prop we are passing to the Component.

==========

React Lists:

-> Lists are very useful when it comes to developing UI of any website.
-> Lists are mainly used for displaying menus in a website, for example, the navbar menu.

```
Ex:  const numbers = [1,2,3,4,5];
  const updatedNums = numbers. map((number)=>{
  return <li>{number}</li>;
 });

 ReactDOM.render( <ul> {updatedNums} </ul>, document.getElementById('root') );
```

But,
If items are fetched from database and then displayed as lists in the browser. So from the component's point of view we can say that we will pass a list to a component using props and then use this component to render the list to the DOM. We can update the above code in which we have directly rendered the list to now a component which will accept an array as props and returns an unordered list.

==========

React Fragments:

§React introduced Fragments from the 16.2 and above version.
§Fragments allow you to group a list of children without adding extra nodes to the DOM.
§It helps in dealing with the issue of json components
Syntax: <React.Fragment> or <>

```
  <Lake />  <Lake />
  <Resort />   <Resort />
 </React.Fragment>  </>
```

Use of Fragments:

-> It makes the execution of code faster as compared to the div tag.
-> It takes less memory.

Note: Key is the only attributes that can be passed with the Fragments.

==========

Conditional Rendering:

-> Conditional rendering in React works the same way conditions work in JavaScript.
-> Use JavaScript operators like if or the conditional operator to create elements representing the current state.
-> To render something based on some condition.
-y > Create multiple components and render them based on some conditions. This is also a kind of encapsulation supported by React.

==========

Error Boundary:

-> The error boundary mechanism helps to show meaningful error message to user on production instead of showing any programming language error.
-> Error boundaries are React components which catch JavaScript errors anywhere in our app, log those errors, and display a fallback UI.
-> It does not break the whole app component tree and only renders the fallback UI whenever an error occurred in a component.
-> Error boundaries catch errors during rendering in component lifecycle methods, and constructors of the whole tree complex application which have multiple components, we can declare multiple error boundaries

```
Syntax:  <ErrorBoundary>
            <Home/>
         </ErrorBoundary>
```

There are 2 methods in Error Boundary:

1.static getDerivedStateFromError(error){ } : It takes error message as an input & return New State

```
Syntax: static getDerivedStateFromError(error) {
 return ( hasError : true
  //Logic ) }
```

2. componentDidCatch (error, errorInfo){ }

```
Syntax: componentDidCatch (error, errorInfo) {
 console.log (error);
 console.log (errorInfo); }
```

Note:

Whenever there is an error in JS/React object, it creates a Object of Error.

```
  throw new Error("error");
```

* It is for the Customized Error.

==========