**********************HTML5:**************************

https://www.tutorialspoint.com/html5/html5

1.features?

Ans:html 5 has introduced some new features they are

1.semantic elements,2.forms , 3.web storage, 4.canvas, 5.audio and video, 6.geolocation,7.web socket,8.server sent events (SSE), 9.micro data,10.drag and drop.

2.new semantic elements and what is the use of it?

Ans: A semantic element clearly describes its meaning to both the browser and the developer.
The example of the non semantic elements are 1.<div>, 2.<span> they tells nothing about its contents
The examples of semantic elements are 1.<form>,2.<table>,3.<article>they clearly define its content.

3.SVG Vs canvas?

Canvas	**SVG**
1.pixel graphics.	1.vector graphics.
2.Resolution dependent.	2.Resolution independent.
3.drawn with javascript.	3.XML format.
4.modified through script only.	4.modified through css and script.
5.not suitable for printing at high resolution.	5.high quality print at any resolution.
6.suitable for games.	6.not suitable for games.

4.web storage(1.local storage 2.session storage)?

Ans: web storage API provides mechanisms by which browsers can store key/value pairs in a much better way than using cookies . the two mechanisms within web storage are as follows

1.**SessionStorage**: maintenance a separate area for each given origin thats available for the duration of the page session (as long as the browser is open including page reloaded and restores).Stores data only for a session meaning that the data is stored until the browser (or tab) is closed . the storage limit is larger than a cookie (at most 5MB).
2.**localStorage**: does the same thing but persists even when the browser is closed and reopened.Stores data with no expiration date and gets clear only through javascript or cleaning the browser cache /locally stored data .Storage limit is larger than the cookie and session storage .

5.cookies?

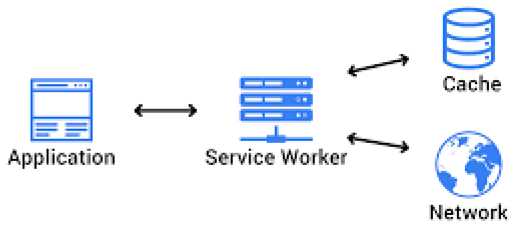
Ans: cookies are small pieces of information that are store directly in the browser .They are a part of http protocol specification .Cookies are usually set by a web server using the response Set-Cookie HTTP header. Then the browser automatically added them to every request to the same domain using Cookie HTTP header.

6.web worker & service worker?

Ans: web worker: web workers makes it possible to run a script operation in a background third separate from the main execution thread of a web application. Advantage of this is that laborious processing can be performed in a separate thread along the main (usually the UI) thread to run without being blocked/slowed down .

Service worker:

Service worker essentially acts as proxy servers that sit between web applications ,the browser and the network . thay are intended to enable the creation of effective offline experiences ,intercept network requests and take appropriate action based on whether the network is available and update assets residing on the server. They will also allow access to push notifications and background sync APIs.



7.form 2.0 ?

Ans: forms are used to collect the data from the user . In HTML file form has have a more semantic elements .In form 2.0 in addition to form input types and sever new input types are added.The following new input types are added 1.datetime-local ,2.month, 3.week, 4.date, 5.time, 6.number, 7.range, 8.email, 9.url.
The <output> tag is introduced to display the output . The attributes placeholder, autofo
cus and required are added.

8 .<!DOCTYPE>?

ANS: :The <!DOCTYPE> declaration tag is used by the web browser to understand the version of the HTML used in the document. Current version of HTML is 5 and it makes use of the following declaration −
<!DOCTYPE html>
There are many other declaration types which can be used in HTML document depending on what version of HTML is being used.

9.what are the wed APIs do you used?

Ans:1.fetch Api, 2.Hostery Api, 3.webstorage API, 4.webworker Api, 5.service worker Api and more..

*********************java script:*********************

1. var, let & const ?

Ans: var declarations are globally scoped or function scoped while let and const are block scoped.

- var variables can be updated and re-declared within its scope; let variables can be updated but not re-declared; const variables can neither be updated nor re-declared.
- They are all hoisted to the top of their scope. But while var variables are initialized with undefined, let and const variables are not initialized.
- While var and let can be declared without being initialized, const must be initialized during declaration.

2.== Vs === ?

Ans: == in JavaScript is used for comparing two variables, but it ignores the datatype of variable. === is used for comparing two variables, but this operator also checks datatype and compares two values. Checks the equality of two operands without considering their type.

3.Is const object property changeable?

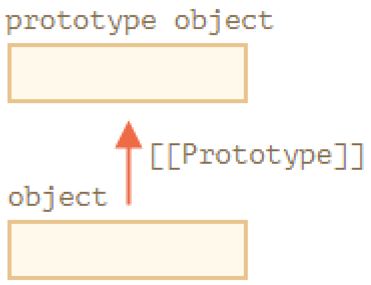
Ans: It does NOT define a constant value. It defines a constant reference to a value.

4.diff null & undefined ?

In JavaScript, undefined is a type, whereas null an object. It means a variable declared, but no value has been assigned a value. Whereas, null in JavaScript is an assignment value. You can assign it to a variable.

5.prototypal inheritance ?

Ans: In JavaScript, objects have a special hidden property [[Prototype]] (as named in the specification), that is either null or references another object. That object is called “a prototype”:



When we read a property from object, and it’s missing, JavaScript automatically takes it from the prototype. In programming, such thing is called “prototypal inheritance”

6.diff function declaration & function expression?

Ans: The main difference between a function expression and a function declaration is the function name, which can be omitted in function expressions to create anonymous functions. A function expression can be used as an IIFE (Immediately Invoked Function Expression) which runs as soon as it is defined.

7.currying ?

Ans:A curried function is a function that takes multiple arguments one at a time.

Let's create a curried version of the function and see how we would call the same function (and get the same result) in a series of calls:

```
function sum(x){
        return function(y){
                return function(z){
                        return x+y+z;
                }
         }
}
sum(2)(3)(4)//Output->9
```

8.setTimeout() ?

Ans: The setTimeout() method of the Window Or WorkerGlobalScope mixin (and successor to Window. setTimeout()) sets a timer which executes a function or specified piece of code once the timer expires.

9. Display Hello World 5 times with delay of 2s.

clearTimeout()

The clearTimeout() function in javascript clears the timeout which has been set by setTimeout()function before that.

- **setTimeout()** function takes two parameters. First a function to be executed and second after how much time (in ms).
- setTimeout() executes the passed function after given time. The number id value returned by setTimeout() function is stored in a variable and it's passed into the clearTimeout() function to clear the timer.

```
function fun() {
   t = setTimeout(color, 3000);
}
function stop() {
   clearTimeout(t);
}
```

10.closure and how it is used?

https://www.w3schools.com/js/js_function_closures.asp

A closure is a function having access to the parent scope, even after the parent function has closed.

Explanation:

In javascript it is possible to write one function inside another function.inner function can be able to access the outer function variables even if the outer function is closed.

Ex:

//variable declared here global

```
var add = (function () {
  var counter = 0;// lexical scope
  return function () {
```

```
            //variable declared here  local
            counter += 1;
            return counter;
            }
        })();
        add();
        add();
        add();
```

11. lexical scope in js?

Ans: A lexical scope in JavaScript means that a variable defined outside a function can be accessible inside another function defined after the variable declaration. But the opposite is not true; the variables defined inside a function will not be accessible outside that function.

12.polyfill ?

Ans:A polyfill is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.

13.hoisting ?

Ans: Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution. Inevitably, this means that no matter where functions and variables are declared, they are moved to the top of their scope regardless of whether their scope is global or local.

14. Write code for add(2, 3)=add(2)(3)?

Ans:

```
console.log(add(2, 3));
    console.log(add(2)(3));
    function add(a, b) {
            return a && b ? a+b : function(c) {return a+c;}
       }
```

15. for..in and for...of ?

Ans: Both for..of and for..in statements iterate over lists; the values iterated on are different though, for..in returns a list of keys on the object being iterated, whereas for..of returns a list of values of the numeric properties of the object being iterated.
Here is an example that demonstrates this distinction:

```
let list = [4, 5, 6];
for (let i in list) { console.log(i); // "0", "1", "2", }
for (let i of list) { console.log(i); // "4", "5", "6" }
```

16. How to clone an object in js?

Ans: Object.assign({},some_object);

17. Array flattening ?

Ans:`function flatten(a) { return Array.isArray(a) ? [].concat.apply([], a.map(flatten)) : a; }`

A bit more compactly with ES6:

```
var flatten = a => Array.isArray(a) ? [].concat(...a.map(flatten)) : a;
```

18. Object flattening ?

Ans:const flattenObject = (obj, prefix = '') =>

```
Object.keys(obj).reduce((acc, k) => {

 const pre = prefix.length ? `${prefix}.` : '';

    if (
      typeof obj[k] === 'object' &&
      obj[k] !== null &&
      Object.keys(obj[k]).length > 0
    )
      Object.assign(acc, flattenObject(obj[k], pre + k));
    else acc[pre + k] = obj[k];
    return acc;
  }, {});
console.log(flattenObject({ a: { b: { c: 1 } }, d: 1 }));
ans:{"a.b.c":1,"d":1}
```

19.Javascript call() & apply() vs bind()?

Ans: With the call() method, you can write a method that can be used on different objects.With the apply() method, you can write a method that can be used on different objects.The difference is:The call() method takes arguments separately.The apply() method takes arguments as an array.Bind creates a new function that will force the `this` inside the function to be the parameter passed to `bind()`.

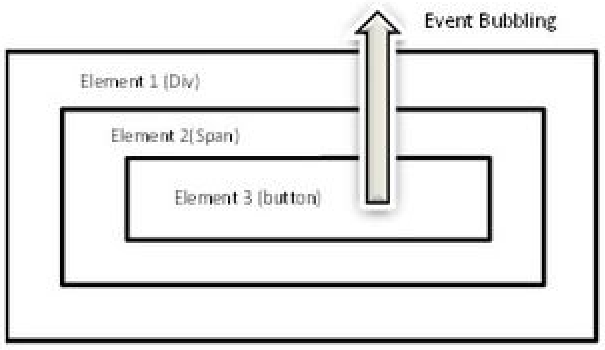
```
x = 9;
var module = { x: 81, getX: function () { return this.x; } };
module.getX(); // 81
var getX = module.getX;
getX(); // 9, because in this case, "this" refers to the global object // create a new function with 'this' bound to module
var boundGetX = getX.bind(module);
boundGetX(); // 81
```

20. What is a Promise?

Ans:A promise is an object that may produce a single value some time in the future: either a resolved value, or a reason that it's not resolved (e.g., a network error occurred). A promise may be in one of 3 possible states: fulfilled, rejected, or pending. Promise users can attach callbacks to handle the fulfilled value or the reason for rejection.

21. Event Bubbling? How to stop event bubbling?.

Ans: Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors of the target element in the same nesting hierarchy till it reaches the outermost DOM element or document object.



https://javascript.info/bubbling-and-capturing

22. What is the output of the following statements?

i. !undefined   ii. !null

Ans: i. !undefined =true   ii. !null = true

23. Objects Sorting?

Ans: list.sort((a, b) => a.name > b.name? 1: -1)

24. CREATE A JAVASCRIPT OBJECT ?

Ans: You can make a JavaScript object in four different ways:

1. with object literals
```
var user001 = {
    firstName: "John",
    lastName: "Smith",
    dateOfBirth: 1985
  };
```
2. using a constructor function
```
function User(firstName, lastName, dateOfBirth) {
     this.firstName = firstName;
     this.lastName = lastName;
     this.dateOfBirth = dateOfBirth;
}
var user001 = new User("John", "Smith", 1985);
```

3. with ECMAScript 6 classes

```
class User {
  constructor(firstName, lastName, dateOfBirth) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.dateOfBirth = dateOfBirth;
    this.getName = function(){
        return "User's name: " + this.firstName + " " + this.lastName;
    }
  }
}
var user001 = new User("John", "Smith", 1985);
```

4. with the Object.create() method

```
        var user001 = {
  firstName: "John",
  lastName: "Smith",
  dateOfBirth: 1985,
  getName: function(){
    return "User's name: " + this.firstName + " " + this.lastName;
  }
};

var user002 = Object.create(user001);
user002.firstName = "Jane";
user002.lastName = "King";
user002.dateOfBirth = 1989;
```

25. What will the following code output?
Ans:

```
const arr = [10, 12, 15, 21];

for (var i = 0; i < arr.length; i++) {

 setTimeout(function() {

   console.log('Index: ' + i + ', element: ' + arr[i]);

 }, 3000);
}
```

Index: 4, element: undefined(printed 4 times).

26.What will the code below output? Explain your answer.?
Ans:

```
console.log(0.1 + 0.2);//0.30000000000000004
console.log(0.1 + 0.2 == 0.3);//false
```

27.In what order will the numbers 1-4 be logged to the console when the code below is executed? Why?
Ans:

```
(function() {
   console.log(1);
   setTimeout(function(){console.log(2)}, 1000);
   setTimeout(function(){console.log(3)}, 0);
   console.log(4);
})();
//1 4 3 2
```

28. Double the array values [1,2,3,4,5].?
Ans:

```
console.log([1,2,3,4,5].map(i=>2*i))
```

29.

```
function replace(a,index,val){
        a.splice(index,1,val);
        return a;
    }
    var x=[1,2,3,4,5];
    console.log(replace(x,2,10))
```

30. Number of occurrences of a character in string.?
Ans:

```
// program to check the number of occurrence of a character
function countString(str, letter) {
   let count = 0;                                    // looping through the items
   for (let i = 0; i < str.length; i++) { // check if the character is at that position
      if (str.charAt(i) == letter) {
         count += 1;
      }
   }
   return count;
}                                                    // take input from the user
const string = prompt('Enter a string: ');
const letterToCheck = prompt('Enter a letter to check: ');//passing parameters and calling the function
const result = countString(string, letterToCheck);// displaying the result
console.log(result);
```

Output

```
Enter a string: school
Enter a  letter to check: o
2
```

31.Write a program that will merge the objects based on their 'id' property, and return the results in a new array. The original arrays should not mutate.?
Ans:

```
const people = [
  {
    id:10021,
    name: 'Anna',
    email: 'anna@greatcompany.com',
  }, {
    id:10022,
    name: 'Riley',
    email: 'riley@greatcompany.com',
  }, {
    id:10023,
    name: 'Nathan',
    email: 'nathan@greatcompany.com',
  }, {
     id:10024,
    name: "Jessica",
    email: 'jessica@greatcompany.com',
  }];

const jobs = [
  {
    id:10021,
    title: 'Engineer'
  },{
    id:10022,
    title: 'Quality Assurance'
  },{
    id:10023,
    title: 'Product Owner'
  }];

const mergedWorkers = [];
for(let peo of people){
     let flag=false;
     for(let job of jobs){
     if(peo.id === job.id){
     mergedWorkers.push({...peo, ...job});
      flag=true;
      break;
    }
  }
  if(!flag)
     mergedWorkers.push({...peo});
}
console.log(mergedWorkers);
```

*******************ES6********************************

1.What is ES6?

Ans:1.ES6 or ECMASCRIPT 2015 is sixth major release of ECMAScript language which comes with a lot of new features and syntax for writing web applications in javascript
2.As currently ,not all browsers support ES6,they support pre-versions of ES6.
3. So to write web applications in ES6 that will support all Browsers we needed tools like Babel and webpack.

2.what are template literals in ES6?

Ans: Template literals are the string with embedded code and variables inside.
2.Template literal allows concatenation and interpolation in much more comprehensive and clear in comparison with prior versions of ECMAScript.
3.In ES6 concatenation and interpolation is done by backtick `` in a single line.To interpolate a variable simply put in to {} braces forwarded by $ sign

```
Let a=”Hello”;
Let b=”john”;
Let c=`$(a)$(b)`;
console.log(c);// output Hello john;
```

3.what is the spread operator in ES6?

Ans: 1. Spread operator provides a new way to manipulate array and objects in ES6.
2.A spread operator is represented by...followed by the variable name

```
Let a=[7,8,9];
Let b=[1,2,3,...a,10];
console.log(b);//[1,2,3,7,8,9,10]
```

3.So spread operator spreads the contents of variable a and concatenates it in b.

4.what is Set in ES6?

Ans: Set is a collection of unique values.The values could be also primitives or object references.
Ex:

```
Let set =new set();
set.add(1);
set.add(‘1’);
set.add({ key:’value’});
console.log(set);//set{1,’1’,object{key:’value’}}
```

5.Explain Generator function in ES6?

Ans: Generators are funcion that can be exited and later re-entered.
2.Their context(variable bindings)will be saved across re-entrances.
3.Afunction keyword followed by an asterisk defines a generator function,which returns a generator object.

```
function *generator(i){
Yield i;
Yield i+10;
}
```

```
Var gen = generator(10);
console.log(gen.next().value);
//expected output:10
```

6.List some new features of ES6?

Ans: The new features of ES6 are

1.Block-Scope support for both variables ,constants,functions.
2.Arrow Functions
3.Extended parameter Handling.
4.Template Literals
5.Extended Literals
6.Enhanced Regular Expression
7.Enhanced object properties
8.Destructuring Assignment
9.Modules,classes,iterators,Generators
10.Support for Map/set & WeakMap/WeakSet
11.promises,Meta-Programming,Internationalization & Localization.

7.What is Babel?

Ans: Bable is one of the most popular javascript transpilers and becomes the industry standard.
2.it allows us to write ES6 code and convert it back in pre-ES6 javaScript that browser supports.

8.What is Wedpack?

Ans: Webpack allows we to run an environment that hosts babel.
2.Webpack is open source javascript module bundler that takes modules with dependencies and generates static assets representing those modules.

9.Expain Destructuring Assignment in ES6?

Ans: Destructing assignment allows us to extract data from array and objects into separate variables.

```
Let c=[100,200,300];
let[a,...b]=c;
console.log(a,b);//output 100[200,300,400]
```

10. Spread vs rest ?

Ans: It’s the opposite to rest parameter , where rest parameter collects items into an array, the spread operator unpacks the collected elements into single elements.

```
.var myName = ["Marina" , "Magdy" , "Shafiq"];
var newArr = [...myName ,"FrontEnd" , 24];
console.log(newArr) ; // ["Marina" , "Magdy" , "Shafiq" , "FrontEnd" , 24 ] ;

ex2:let a=[1,2,3,4],b=[5,6,7,8];
        Let c=[...a,...b,9,10];
       console.log(c);//[1,2,3,4,5,6,7,8,9,10]
```

rest:It is a collection of all remaining elements (hence, the name rest, as in “the rest of the elements”) into an array.

```
var myName = ["Marina" , "Magdy" , "Shafiq"] ;
const [firstName , ...familyName] = myName ;
```

```
console.log(firstName); // Marina ;
console.log(familyName); // [ "Magdy" , "Shafiq"] ;
```

**********************CSS*********************************

1.boxmodel?

Ans: Every element surrounded with a box, with height,width,padding,border and margin. In general the content of an element will take width and height , padding and margin will take extra space in the webpage .On the other hand if you want a set hole box within width and height we need to use CSS property box-sizing:border-box.

2.CSS selectors?

Ans:CSS selectors are used to "find" (or select) the HTML elements you want to style.

We can divide CSS selectors into five categories:

- Simple selectors (select elements based on name, id, class)
- Combinator selectors (select elements based on a specific relationship between them)
- Pseudo-class selectors (select elements based on a certain state)
- Pseudo-elements selectors (select and style a part of an element)
- Attribute selectors (select elements based on an attribute or attribute value)

2.specificity?

Ans: If there are two or more conflicting CSS rules that point to the same element, the browser follows some rules to determine which one is most specific and therefore wins out.
The universal selector (*) has low specificity, while ID selectors are highly specific!

3.How to exactly center an element in the web page?

Ans:
```
#1: div {
      color: white; background: red; padding: 15px;
      position: absolute; top: 50%; left: 50%;
      -ms-transform: translateX(-50%) translateY(-50%);
      -webkit-transform: translate(-50%,-50%);
      transform: translate(-50%,-50%);
}
```

#2:
```
    .centerFlex {
          display: flex;
          align-items: center;
          justify-content: center;
    }
```

4.diff btw Static,Relative ,Absolute,sticky and Fixed positions ?

Ans:

Static:when we set position as static it is placed in the normal flow as appear in the html file.

Relative:In relative also the element position same as static but hear left ,right, top and bottom are applicable for static these are not applicable.

Absolute:in absolute positioning the element placed relative to its parent .

Fixed:in fixed position the element fixed relative to its view port even the page is scrolled the element position will not be changed.

Sticky:At first it is relative and when page scrolled it moves to viewport border and its position is fixed.

5.diff btw visibility:hidden , display:none ?

Ans: In the both properties the element will not appear in the web page . in visibility hidden the element takes space but in display:none completely removes element from the DOM.

8.Pseudo element vs pseudo class in css ?

Ans: Basically a pseudo-class is a selector that assists in the selection of something that cannot be expressed by a simple selector, for example :hover. A pseudo-element however allows us to create items that do not normally exist in the document tree, for example ``::after`.

9.Flex box model?

Ans: The Flexbox Layout (Flexible Box) module aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word “flex”).

The flex container properties are:

- `flex-direction`
- `flex-wrap`
- `flex-flow`
- `justify-content`
- `align-items`
- `align-content`

10.flex box vs grid model?

ANS:

Flex box	Grid
1.one dimensional	1.two dimensional
2.content based	2.container based
3.doen’t have gap property	3.does have gap property

11.align-self: ?

Ans: The align-self property specifies the alignment for the selected item inside the flexible container.
The align-self property overrides the default alignment set by the container's align-items property.
ex:
```
  <div class="flex-container">
   <div>1</div>
   <div>2</div>
   <div style="align-self: center">3</div>
   <div>4</div>
</div>
```
12.diff btw em and rem?

Ans: em units for the font-size property will be relative to the font-size of the parent element. rem units sizes will always be relative to the font-size of the root html element.

13.how to design responsive web pages using css?

Ans: By using media queries
Syx:
```
@media only screen and (max-width: 600px) {
  body {
     background-color: lightblue;
  }
}
```

**********************react js***************************

1.what is react?

Ans:React is a frontend javascript library ,for creating reusable UI components for both mobile and web.React is used to build single page applications.React is developed by Facebook in 2011
but it opened sourced in 2015 ,it has one of the largest community support.

2.React lifecycle methods , should component update , get derived state from props(w3schools)

https://www.w3schools.com/react/react_lifecycle.asp

1.React lifecycle methods , should component update(w3schools)

Mounting

Mounting means putting elements into the DOM.
React has four built-in methods that gets called, in this order, when mounting a component:

1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

getDerivedStateFromProps

The `getDerivedStateFromProps()` method is called right before rendering the element(s) in the DOM.

This is the natural place to set the `state` object based on the initial `props`.

It takes `state` as an argument, and returns an object with changes to the `state`.

The example below starts with the favorite color being "red", but the `getDerivedStateFromProps()` method updates the favorite color based on the `favcol` attribute:

Example:
The getDerivedStateFromProps method is called right before the render method:

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}
ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```

Updating

The next phase in the lifecycle is when a component is updated.
A component is updated whenever there is a change in the component's state or props.
React has five built-in methods that gets called, in this order, when a component is updated:

1. getDerivedStateFromProps()
2. shouldComponentUpdate()
3. render()
4. getSnapshotBeforeUpdate()
5. componentDidUpdate()

shouldComponentUpdate:

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.
The default value is true.
The example below shows what happens when the shouldComponentUpdate() method returns false:
Example:
Stop the component from rendering at any update:

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  shouldComponentUpdate() {
    return false;
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

ReactDOM.render(<Header />, document.getElementById('root'));
```

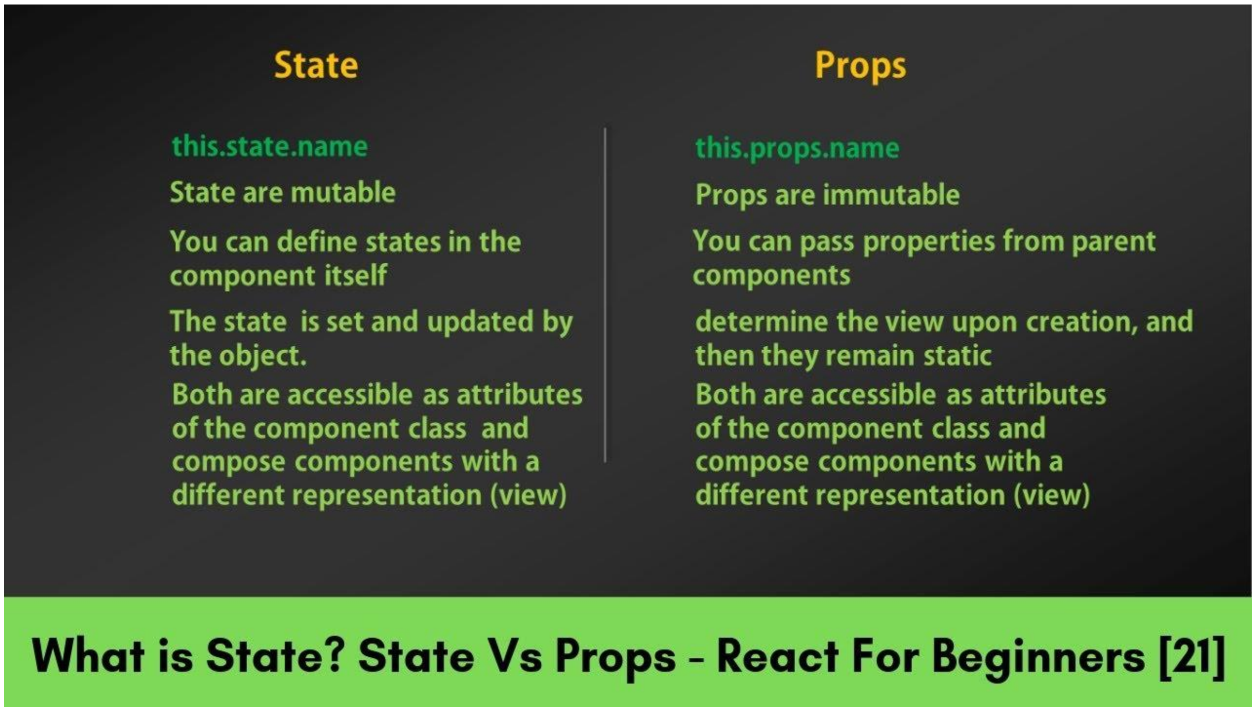
Unmount

The componentWillUnmount method is called when the component is about to be removed from the DOM.

1.coponentWillUnmount().

3.props vs state?

Ans: In react it follows the flux archturate that is data is one way binding it flowers from parent to its child from child to its child and soon .so data is send from parent to child in form of props.so props are immutable ,so the component can only render through props.
State: to design interactive components the component maintenance information the form of state ,so the state is mutable, if the state can be changed in the setState() method the component will be rerender.



State	Props
this.state.name	this.props.name
State are mutable	Props are immutable
You can define states in the component itself	You can pass properties from parent components
The state is set and updated by the object.	determine the view upon creation, and then they remain static
Both are accessible as attributes of the component class and compose components with a different representation (view)	Both are accessible as attributes of the component class and compose components with a different representation (view)

What is State? State Vs Props - React For Beginners [21]

4.React hooks?

Ans: Before react 16.8 there is no way to maintain the state in the functional component .In react 16.8 react introduce hooks concept with the help of hooks we can maintain the state in the functional component using useState hook .Using useEffect hook we can achieve component life cycle method effects.

5.what is the purpose of hooks?
Ans:Hooks is making **big** waves in the **React** community because it reduces the complexity of state management. While Redux is awesome, there are many steps in the process. In contrast, **hooks** cut out a lot of the steps and makes it a simple and singularized step, cutting out a lot of **what** can be seen as middleware functions.

6.what is useState()?
Ans:A Hook is a special function that lets you “hook into” React features. For **example**, useState is a Hook that lets you add React state to function components.

7.what is useEffect?
Ans: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.

8.what is useReducer()?
Ans:useReducer is one of a handful of **React** hooks that shipped in **React** 16.7. 0. It accepts a reducer function with the application initial state, returns the current application state, then dispatches a function.

9.what is useCallBack()?
Ans:React's useCallback Hook can be used to optimize the rendering behavior of your **React** function components. ... While **useCallback** is used to memoize functions, **React** memo is used to wrap **React** components to prevent re-renderings.

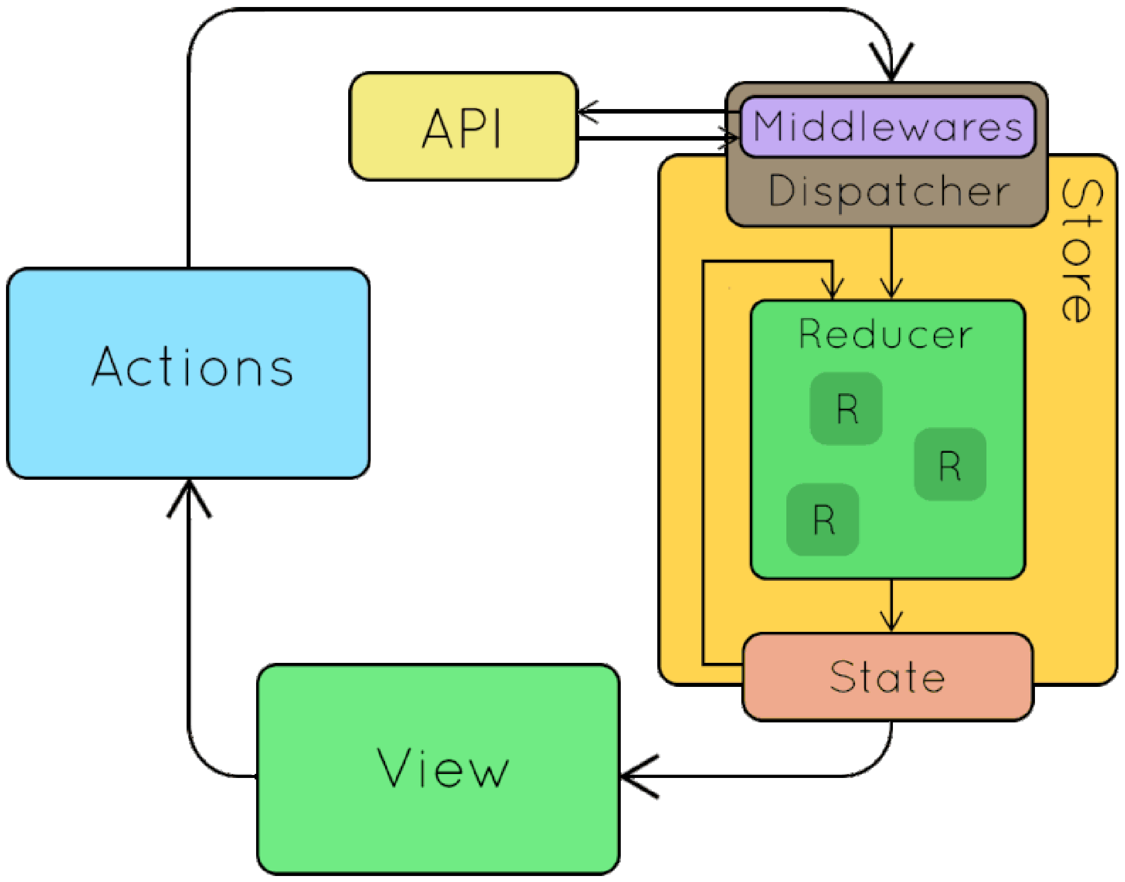
10.how to prevent infinite loops in useEffect?
Ans: Passing an empty array as the second argument to **useEffect** makes it only run on mount and unmount, thus **stopping** any **infinite loops**.

11.React hoc?
Ans:hoc is a javascript function which takes a component as an argument and returns a new component .
Ex:with Router , connect are higher order components.

12.Redux life cycle?
Ans:if a state change occur in a UI it dispatches action .action will be taken by the store then store executes a reduser .A reduser takes action and object as a parameter and returns new object because reduser is a pure function is does’t modify existing object .It creates the new objects accordingly UI is updated.

After the action is dispatch before it reaches the store if we want modify an action based on the sutten condition or to make API calls a middleware like thunk is used .

First we will create actions based on actions reducers were created. By passing reducer as an argument to createStore we can get store object by passing store as a provider attribute in the root we can get the store.this store object can be accessed in the component using connect hoc by passing mapStateToProps and mapDispatchToProps we can obtine store object in the component in the form of props.



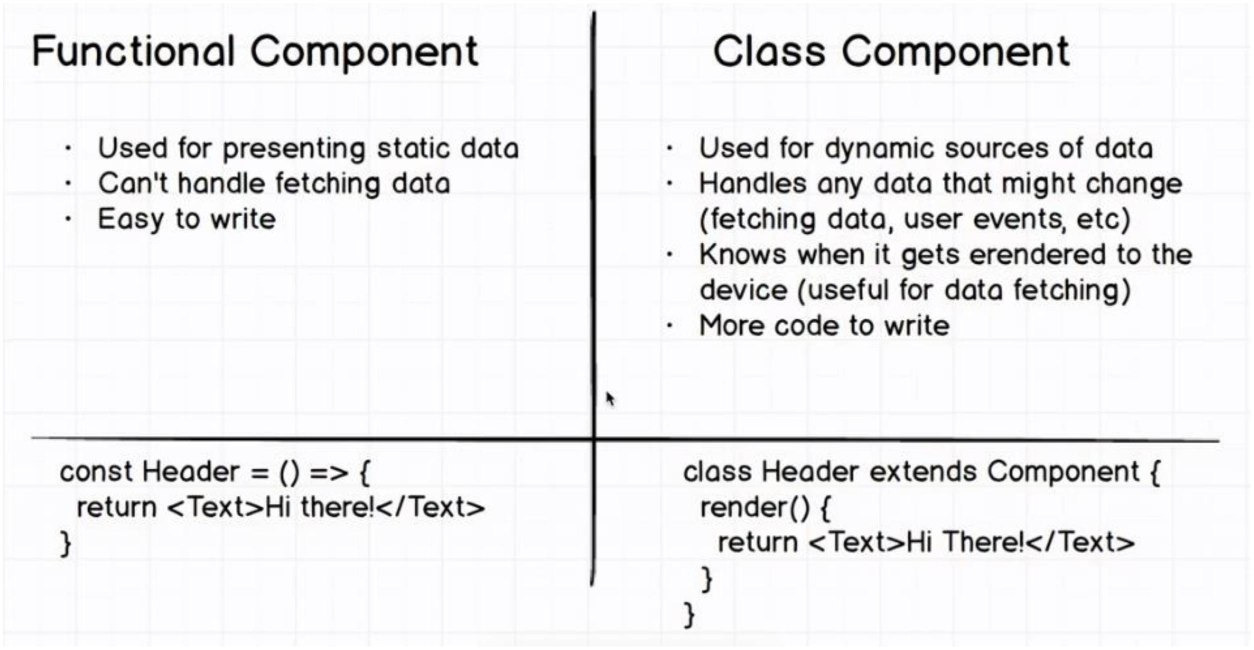
13.Redux thunk vs saga?

Ans: **B**oth Redux Thunk and Redux Saga take care of dealing with side effects. In very simple terms, applied to the most common scenario (async functions, specifically AJAX calls) Thunk allows ``Promises'' to deal with them, Saga uses Generators.
The idea is that a saga is similar to a separate thread in your application that's solely responsible for side effects. However, unlike Redux-Thunk, which utilizes callback functions, a Redux-Saga thread can be started, paused and cancelled from the main application with normal Redux actions.
Redux-Saga middleware allows you to express complex application logic as pure functions called sagas. Pure functions are desirable from a testing standpoint because they are predictable and repeatable, which makes them relatively easy to test.

14.Ref?

Ans: Refs are created using React. createRef() , and are assigned to class properties. In the above example the ref is named myRef , which is then attached to an <input> DOM element. Once a ref is attached to an element, that element can then be accessed and modified through the ref.

15.class components vs functional components or

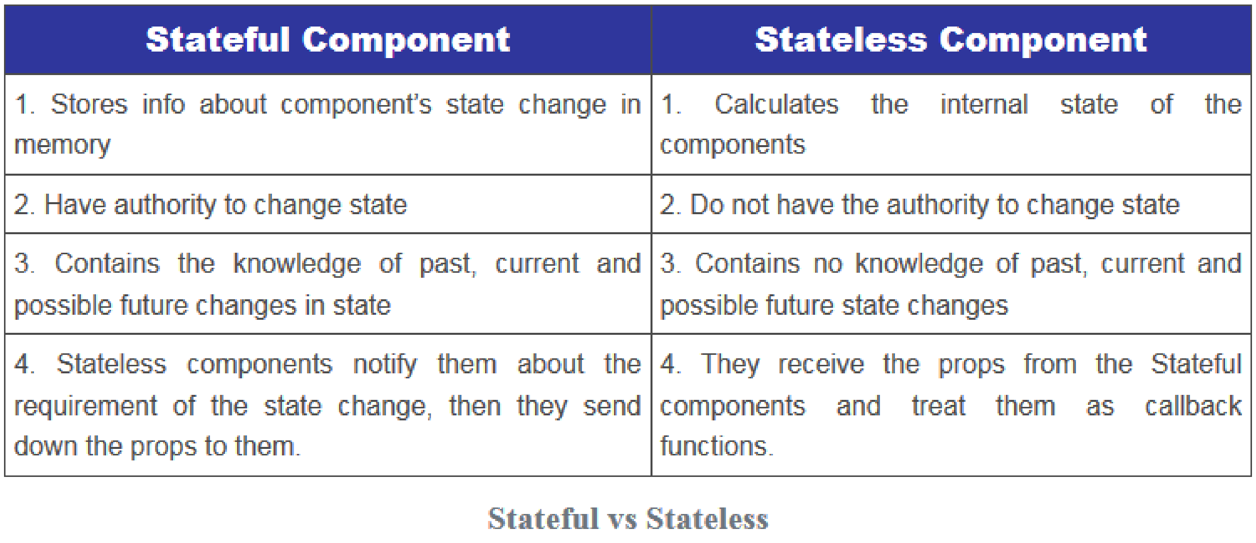


Functional Component	Class Component
• Used for presenting static data • Can't handle fetching data • Easy to write	• Used for dynamic sources of data • Handles any data that might change (fetching data, user events, etc) • Knows when it gets erendered to the device (useful for data fetching) • More code to write

```
const Header = () => {
  return <Text>Hi there!</Text>
}
```

```
class Header extends Component {
  render() {
    return <Text>Hi There!</Text>
  }
}
```

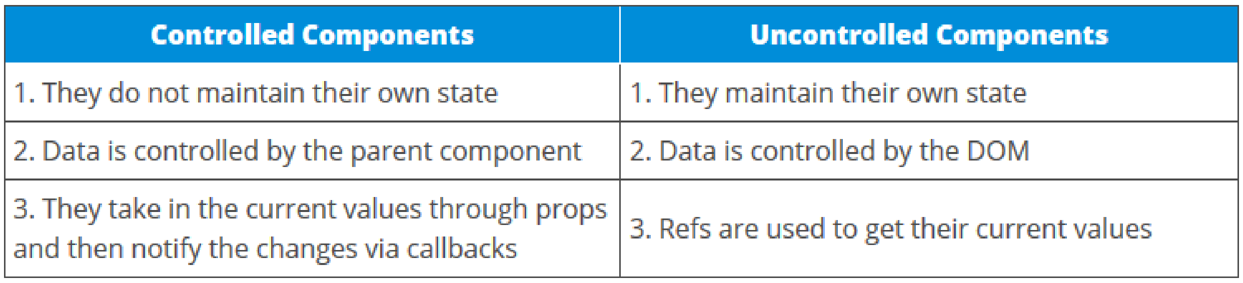
stateful vs stateless components ?



Stateful Component	Stateless Component
1. Stores info about component's state change in memory	1. Calculates the internal state of the components
2. Have authority to change state	2. Do not have the authority to change state
3. Contains the knowledge of past, current and possible future changes in state	3. Contains no knowledge of past, current and possible future state changes
4. Stateless components notify them about the requirement of the state change, then they send down the props to them.	4. They receive the props from the Stateful components and treat them as callback functions.

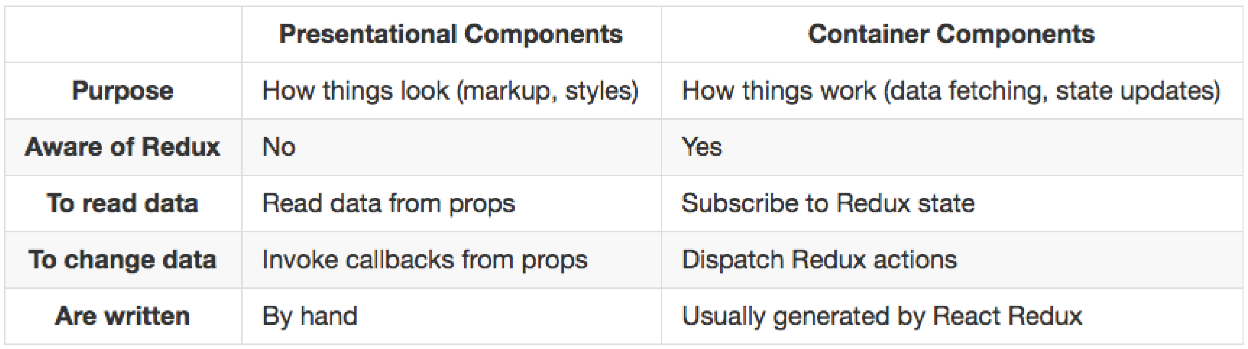
Stateful vs Stateless

Controlled vs uncontrolled



Controlled Components	Uncontrolled Components
1. They do not maintain their own state	1. They maintain their own state
2. Data is controlled by the parent component	2. Data is controlled by the DOM
3. They take in the current values through props and then notify the changes via callbacks	3. Refs are used to get their current values

Presentational Vs Container Components



	Presentational Components	Container Components
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
Aware of Redux	No	Yes
To read data	Read data from props	Subscribe to Redux state
To change data	Invoke callbacks from props	Dispatch Redux actions
Are written	By hand	Usually generated by React Redux

16.React hoc ?

Ans: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.

17. How to prevent components from re-rendering?

Ans: That's where you can use the more broad yet simpler solution for preventing the rerender: React's PureComponent. React's PureComponent does a shallow compare on the component's props and state. If nothing has changed, it prevents the rerender of the component. If something has changed, it rerenders the component.

18.what is the pure component?

Ans: A React component can be considered pure if it renders the same output for the same state and props. For class components like this, React provides the PureComponent base class. Class components that extend the React.PureComponent class are treated as pure components.
Pure components have some performance improvements and render optimizations since React implements the shouldComponentUpdate() method for them with a shallow comparison for props and state.

19. What is React.memo?

Ans: React. memo is a higher order component. If your component renders the same result1 reuse the last rendered result.

20. Why is setState asynchronous in react?

Ans: This is because setState alters the state and causes rerendering. This can be an expensive operation and making it synchronous might leave the browser unresponsive. Thus the setState calls are asynchronous as well as batched for better UI experience and performance.

21. What is the context API in react?

Ans: React Context API is a way to essentially create global variables that can be passed around in a React app. This is the alternative to "prop drilling", or passing props from grandparent to parent to child, and so on. Context is often touted as a simpler, lighter solution to using Redux for state management.

22. Arrow function vs normal function

Ans: "this" binding

Unlike regular functions, arrow functions don't have their own this or arguments binding. Instead, those identifiers are resolved in the lexical scope like any other variable.

"arguments" binding

Arguments objects are not available in arrow functions, but are available in regular functions.

Arrow functions cannot be called with "new":

If a function is constructible, it can be called with new, i.e. new User(). If a function is callable, it can be called without new (i.e. normal function call).

Regular functions created through function declarations / expressions are both constructible and callable.

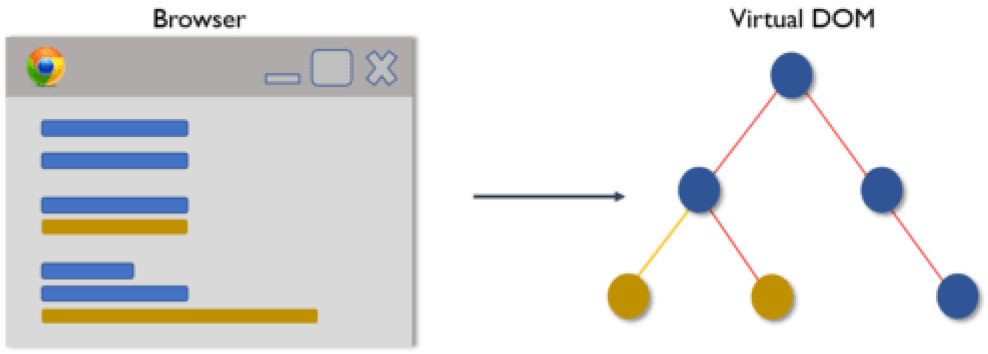
Arrow functions (and methods) are only callable i.e arrow functions can never be used as constructor functions. Hence, they can never be invoked with the new keyword.

23. Real DOM Vs Virtual DOM

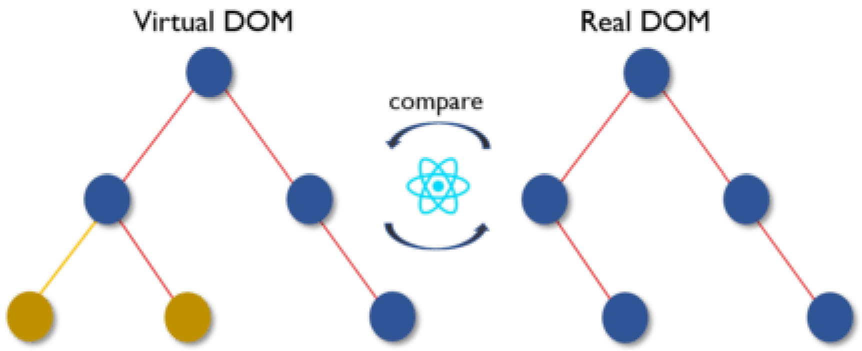
Ans: A virtual DOM is a lightweight JavaScript object which originally is just the copy of the real DOM. It is a node tree that lists the elements, their attributes and content as Objects and their properties. React's render function creates a node tree out of the React components. It then updates this tree in response to the mutations in the data model which is caused by various actions done by the user or by the system.

This Virtual DOM works in three simple steps.

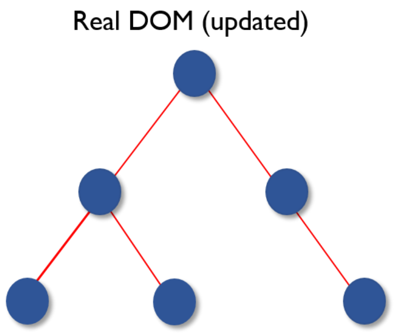
1. Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.



2. Then the difference between the previous DOM representation and the new one is calculated.



3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.



24. XML Vs JSON

- JSON object has a type whereas XML data is typeless.
- JSON does not provide namespace support while XML provides namespaces support.
- JSON has no display capabilities whereas XML offers the capability to display data.
- JSON is less secured whereas XML is more secure compared to JSON.
- JSON supports only UTF-8 encoding whereas XML supports various encoding formats.

25. What is react routing ? What are the components used in react routing ? What is its history?

Ans: React Router, and dynamic, client-side routing, allows us to build a single-page web application with navigation without the page refreshing as the user navigates. React Router uses component structure to call components, which display the appropriate information.

https://www.educative.io/blog/react-router-tutorial#what-is

26. Localization

Localizing a react application has become an easy task with i18next's react implementation.

https://medium.com/@jishnu61/6-easy-steps-to-localize-your-react-application-internationalization-with-i18next-8de9cc3a66a1

27. What is the difference between authentication and authorization?

Ans: Authentication is identifying users by confirming who they say they are, while authorization is the process of establishing the rights and privileges of a user. Both processes play equally important roles in securing sensitive data assets from breaches and unauthorized access.

28.Where to make API calls in React?

Ans: As best place and practice for external API calls is React Lifecycle method componentDidMount(), where after the execution of the API call you should update the local state to be triggered new render() method call, then the changes in the updated local state will be applied on the component view.

29.How do you call multiple API in react JS?

Ans: If you want to call multiple API calls simultaneously, there's a better approach using Promise. all() . But if one API calls requires data from another, returning the fetch() method like this provides a simple, readable, flat structure and let's you use a single catch() for all of your API calls.

30.in Package.json diff btw dependencies and dev dependencies?

Ans: The difference between these two, is that devDependencies are modules which are only required during development, while dependencies are modules which are also required at runtime. To save a dependency as a devDependency on installation we need to do an npm install --save-dev , instead of just an npm install --save.

31. fetch Vs axios?

AXIOS	FETCH
Axios has url in request object.	Fetch has no url in request object.
Axios is a stand-alone third party package that can be easily installed.	Fetch is built into most modern browsers; no installation is required as such.
Axios enjoys built-in XSRF protection.	Fetch does not.
Axios uses the data property.	Fetch uses the body property.
Axios' data contains the object.	Fetch's body has to be stringified.
Axios request is ok when status is 200 and statusText is 'OK'.	Fetch request is ok when the response object contains the ok property.
Axios performs automatic transforms of JSON data.	Fetch is a two-step process when handling JSON data- first, to make the actual request; second, to call the .json() method on the response.
Axios allows cancelling request and request timeout.	Fetch does not.
Axios has the ability to intercept HTTP requests.	Fetch, by default, doesn't provide a way to intercept requests

Axios has built-in support for download progress.	Fetch does not support upload progress.
Axios has wide browser support.	Fetch only supports Chrome 42+, Firefox 39+, Edge 14+, and Safari 10.1+ (This is known as Backward Compatibility).

32. In how many ways data is transferred from one component to another component?
Ans:

1. Passing data from parent to child using props
2. Passing data from child to parent employing callbacks
3. Passing data among siblings. This can be achieved by one of the following methods:
   a. Integrating the methods mentioned above
   b. Using Redux
   c. Utilizing React's Context API.

33. ESLint?
Ans:Linting JavaScript With ESLint. ESLint is a popular **linter** for JavaScript. It's primarily used to capture language related issues but can be used to enforce code style and good practices. It can fix many issues automatically, especially code style.
https://medium.com/@RossWhitehouse/setting-up-eslint-in-react-c20015ef35f7

34. Error boundary ?
Ans: 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.
A class component becomes an error boundary if it defines either (or both) of the lifecycle methods static getDerivedStateFromError() or componentDidCatch(). Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown. Use componentDidCatch() to log error information.

35. How to configure HTTPS in a React app on localhost?
Ans: the create-react-app application is ran using npm run start, or simply npm start, because in the package.json file’s scripts section, we have this line:

```
"start": "react-scripts start"
```

change that to:

```
"start": "HTTPS=true react-scripts start"
```

This sets the HTTPS environment variable to the true value.