REACT JS

NEW CLASS NOTES

BY

Mr.SAMBA SIR

sri raghavendra Xerox

All software language materials available

beside sathyam theatre line balkampetrcad ameerpet Hyderabad

cell :9951596199

Installation of Softwares to build the JS appl^n

1. Download and Install NodeJS
   - "NodeJS" is the serverside scripting language.
   - "NodeJS" used to develop the servers.
     - Http servers
     - Tcp "
   - Node Servers helps to execute the JS appl^n

website : https://nodejs.oreg/en/download/
file : node-v10.16.3-x64.msi

2. Download and Install Visual studio code
   - "VisualStudio code" is the IDE develop the UI Appl^n like angular, React, Node, JS TS..
   - "Visual Studio code" IDE given by Microsoft.
   - "Visualstudio code" IDE is the open source IDE

website :- https://code.visualstudio.com/docs/?dv=win.
file : VScode usersetup-x64-1.39.2.exe.

Execution of JS files :-

→ node sample.JS

→ automatically we can execute the JS files.

Variables :-

→ Variables are used to store the data.

→ By using variables we can store any type of data.

→ we will define variable by using "var/let/const" keyword.

→ "let" & "const" keywords introduced in ES5.

→ Variables should contain [a-z] [A-Z] [0-9] and special characters like "$" and "_" symbol

→ Variable should not start with "digit".

Syntax :- var/let/const variable name = value;

Numbers :- var decimal = 100;

var double = 100.12345;

var hexadecimal = 0XABC123 → starts with 0x.

var octal = 00123 → starts with 0o.

var binary = 0b1010; → binary starts with 0b.

console.log (decimal, double, hexadecimal, octal, binary);

String :- we can declare string in 3 ways.

- " ", ' ' and ` ` (backtick) operator.
- ` ` (backtick) introduces in ES6.
- ` ` (backtick) used to define the multiline strings.

```
Var sub-one = "ReactJS";
Var sub-two = 'NodeJS';
Var sub-three = 'MongoDB';
Console.log (sub-one, sub-two, sub-three);

Var tbl-name = "employees";
Var sal = 50000;
Var sqlquery = 'select * from ${tbl_name} where
    esal > ${sal}';
Console.log (sqlquery);
```

Boolean :- `Var flag = true;`
`Console.log (flag);`

```
Var arr = ["ReactJS",
           "NodeJS",
           "MongoDB",
           "CassandraDB",
           "VueJS"];
```

// for Loop

```
for (var i=0; i<arr.length; i++);
    console.log (arr[i]);
};
```

// for each()

```
arr.forEach ( () => {} )
```

(callback func" — argument)

```
arrc.forEach((element, index) => {
    console.log(element, index);
});
```

// for ... of

```
for (var value of arrc) {
    console.log(value);
};
```

// Cursors

```
var cursor = arrc[Symbol.iterator]();
console.log(cursor.next());
console.log(cursor.next());
console.log(cursor.next());
```

Diff bet^n Var and let keywords :-

→ "var" keyword allows the duplicate variables.
"let" keyword won't allows the duplicate variables.

Ex : 1

```
var data = 100;
var data = 200;
console.log(data); // 200
```

Ex : 2

```
let data = 100;
let data = 200;
console.log(data); // Syntax error: Identifier 'data' has already been declared.
```

→ Variable hoisting raised because of "var" keyword. We can overcome variable hoisting by using "let" keyword.

Ex:1 console.log(data); // error ⇒ undefined ⇒

Javascript ⇒ ⇒ variable hoisting

var data = 100;

Ex:2 console.log(data); // Reference error: data is not defined.

let data = 100;

→ Because of "var" keyword global polluting issue raised. If "block" effecting global member then corresponding issue called as global polluting issue.

→ We can overcome global polluting issue by using "let" keyword.

Ex.1 : // global variable

var data = 100;

{

// local variable

var data = 200;

}

console.log(data); // 200

BY If we take let then o/p will be 100.

EX:2 : let data = 100

```
{ let data = 200;
}
console.log(data); // 100
```

→ "var" keyword breaks the scope rule,
"Let" " obeys the scope rule.

```
for (var i=0; i<10; i++) {
};
console.log(i); // 10
for (let i=0; i<10; i++) {
};
console.log(i); // i is not defined.
```

→ "var" keyword introduced in ES1.
"Let" keyword introduced in ES5.

→ "var" members are not the "blocked scope" members.

→ "Let" members are block scoped members.

Const :-

→ const is the keyword introduced in ES5.

→ "Const keyword used to define the variables.

→ We can't reassign constent members.

```
const data = 100;
console.log(data); // 100
data = 200;
console.log(data); // Type error : Assigment to contant varciable.
```

Functions :-

→ Perticulare business logic called as fenc°

→ function are used to reuse the business logic.

→ We will define functions by using "function" keyword.

→ As per ESC Standards we have following types of function

1) Named functions
2) Anonymous / Arrow function / Callback func°
3) Rest parameters in func°
4) Default " func°.
5) optional " func°

Named functions :- The function with the name called as named function

Syntax :- function definition

```
function functionname (arguments) {
    // business logic
};
// call the function
functionname (parameters);
```

1) function fun_one() {
return "welcome to reactjs";
};

```
console.log(fun_one); // [function: fun_one]
console.log(fun_one()); // welcome to reactjs.
```

2) function fun_one(arg1, arg2, arg3) {
return arg1 + "<=>" + arg2 + "<=>" + arg3;
};

```
console.log(fun_one("ReactJs", "NodeJs", "MongoDB"));
console.log(fun_one("Angular", "NodeJs", "MYSQL"));
console.log(fun_one("VueJs", "NodeJs", "SQLServer"));
```

3) Create the function :-

```
function fun_one() {
    return fun_two();
};
function fun_two() {
    return "welcome";
}
console.log(fun_one()); // welcome.
```

4) Take one empty Array :-

```
let arr = [];
```

```
function fun_one() {
    return "Hello";
};
for (var i=0; i<5; i++) {
    arr.push(fun_one());      // push data to array.
};
arr.forEach((element, index) => {
    console.log(element, index);
});
```

5)

```
function fun_one() {
    return fun_two,
};
function fun_two() {
    return "Hello";
};
console.log(fun_one);      → fun_one body
console.log(fun_one());    → fun_two body
                           → fun_two o/p.
console.log(fun_one()());  // Hello
```

6)

```
let arr = [];
function fun_one() {
    return "welcome";
};
for (let i=0; i<5; i++){
    arr.push(fun_one);
};
```

```
for (let i=0; i<arrr.length; i++){
    console.log (arrr[i]());
}
```

7) Create function

```
function fun_one(){
    console.log (arg1[0](), arg1[1]());
};
function fun_two(){
    return "Hello_1";
}
function fun_three(){
    return "Hello_2";
};
let arrr = [fun_two, fun_three];
  fun_one(arrr);
```

Anonymous function :- 21.10.19

- A function without name called as anonymous function.
- Anonymous function also called arrow functions
- Arrow functions introduced in ES6.
- Arrow functions behaves like callback functions.
- we will represent Arrow functions by using "=>" symbol.

Syntax :-

```
// function definition
var/let/const variablename = (arguments) => {
        // business logic
    };

// call the function
variablename (parameteres);
```

o/p - node demo.js

1. /* create the arrow function with the help of following variable @ fun_one
   arrow function will return "welcome" message */

```
let fun_one = () => {
    return "welcome";
}
console.log (fun_one);        [function-body]
console.log (fun_one());      //o/p: welcome
```

/* Ex2: Create the arrow function with three arguments */

```
let fun_one = (arg1, arg2, arg3) => {
    return arg1 + "---" + arg2 + "---" + arg3;
};

console.log(fun_one("ReactJs", "NodeJs", "MongoDB"));
console.log(fun_one("Angular", "React", "MySQL"));
console.log(fun_one("VueJs", "NodeJs", "SQLServer"));
```

3. /* one arrow function return another arrow function */

```
let fun_one = () => {
    return fun_two();
};

let fun_two = () => {
    return "welcome";
};

console.log(fun_one()); // welcome.
```

4. /* Pass one arrow function as argument to another arrow function */

```
let fun_one = (arg1, arg2, arg3) => {
    console.log(arg1, arg2, arg3);
};

let sub_one = () => {
    return "ReactJs";
};
```

```
let sceb_two = () => {
    return "Nodejs";
};
let sceb_three = () => {
    return "MongoDB";
};
fun_one( sceb_one(), sceb_two(), sceb_three() );
```

O/P = Reactjs Nodejs MongoDB

/* Ex5: Push arrow function to array upto 5 times */

```
let arr = [];
let fun_one = () => {
    return "welcome";
};
for(let i=0; i<10; i++) {
    arr.push(fun_one());
};
console.log(arr);
```

O/P:- 10 times welcome,

/* Ex6: One arrow function return another arrow function definition */

```
// outer arrow function
let fun_one = () => {
// returning inner arrow function
    return () => {
        return "welcome";
    };
};
// welcome
```

EX:7 /* Pass arrow function defination as argument to another arrow function " */

```
let serverCall = (PosRes, errorRes) => {
    console.log (PosRes(), errorRes());
    };

let - successCallback = () => {
    return "success --- ! ";
    };

let - errorCallback = () => {
    return "Error --- ! ";
    };

ServerCall (-SuccessCallBack, -ErrorCallBack);
```

~~EX:8 /* Rest Parameters in function~~

Inrhythm

Rest Parameters in function :-

- Rest parameters introduced in ES6 .-
- we will represent rest parameters by using "..." symbol.
- "..." also called as spread operator.
- "Rest Parameters" have the capability to store multiple values.
- the default value of rest parameter is empty "[]".
- we can't supply more than one rest parameter.
- rest parameter position should be the last position in arguements.

EX 1 : /* create the Named function with spread operator */

```
function fun_one (...arg1) {
    console.log (arg1);
};

fun_one();          // [ ]
fun_one (10);       // [10]
fun_one( 10 ,20, 30, 40, 50); // [10 ,20, 30, 40, 50 ]
fun_one (undefined);  // [undefined]
fun_one (null);       // [null]
fun_one(undefined , null , undefined); [undefined, null,
                                        undefined ]
```

```
/* function fun_one (...arg1, ...arg2) {}
// SyntaxError: Rest Parameter must be last formal Parameter */

/* function fun_one (...arg1, arg2) {}
SyntaxError: Rest_Parameter must be last formal Parameter. */

/* function fun_one (arg1, ...arg2) {
    console.log (arg1, arg2);
};
fun_one ();                    // undefined []
fun_one (10,10);               // 10 [10]
fun_one (10,20,30,40,50);      // 10 [20, 30, 40, 50]
fun_one (undefined, undefined); // undefined [undefined]
fun_one (null, null);          // null [null]
```

29/10/19

Default Parameters

1) Default Parameters in function :-

→ While defining the functions, few parameters are initialized with default values.

```
function fun_one(arg1 = "ReactJS",
                 arg2 = "NodeJS",
                 arg3 = "MongoDB") {
    console.log(arg1, arg2, arg3);
};
  fun_one();
fun_one("Angular", "AWS", "firebase");
fun_one(undefined
        undefined
        undefined); // ReactJS, NodeJS MongoDB
fun_one(undefined, "Gitserver", "SQLserver");
                // ReactJS Gitserver SQLserver
fun_one(null, null, null); // null null null
```

original value never be disturbed

original value deleted

2)
```
function fun_one(arg1,
                 arg2 = "NodeJS",
                 arg3) {
    console.log(arg1, arg2, arg3);
};
```

```
fun_one();          // undefined 'NodeJS' undefined.
fun_one("ReactJS", undefined, "MongoDB")
                    // ReactJS, NodeJS, MongoDB.
fun_one("Angular", "AWS", "MySQL"); // Angular AWS MySQL.
fun_one(null, null, null)  // null null null
```

3) function fun_one(arg1, → normal Parameter
arg2 = "Hello", → default parameter
...arg3) { → Rest API Parameter.

```
    3;
    fun_one(); // undefined 'Hello' []
fun_one("Hello", undefined, "Hello");
                    // Hello Hello ['Hello']
fun_one("Hello", null, null);
                    // Hello null [null]
```

Optional Parameters in functions ;..?

→ "optional parameters" introduced in ES6.

→ "optional parameter" we will represent with "?"

→ while calling functions few parameters are optional.

- We can execute "optional parameters" application on typescript environment.
- we can't execute "optional parameters" on Javascript environment.
- we will install Typescript by using following command.
- npm install -g typescript@latest
- "npm" stands for node Package Manager.
- "npm" is inbuilt tool of nodeJs.
- "-g" stands for global installation.
- we will execute Typescript Applications by using following commands.
- tsc demo.ts
- node demo.js

demo.ts :(epm

1)
```
function funone(arg1?:string,
                arg2?:string,
                arg3?:string):void {
    console.log(arg1, arg2, arg3);
};
funone();  // undefined undefined undefined
funone("ReactJs"); // ReactJs undefined undefined
funone(null, null, null); // null null null
```

ex 2)

```
function fun_one (arg1 : String,          // data type
                  arg2 ?: String) : Void {
    console.log(arg1, arg2);
};
fun_one(); // Expected 1-2 arguments but got 0.
fun_one("Hello"); // Hello, undefined
fun_one("Hello," "Hello",) // Hello Hello
fun_one(undefined, "Hello"); // undefined 'Hello'
```

ex 3)

```
function fun_one (arg1 : String,
                  arg2 : String = "Hello",
                  arg3 ?: String ) : Void {
    console.log (arg1, arg2, arg3);
};
fun_one(); // Expected 1-3 arguments, but got 0.
fun_one("Hello"); // Hello, Hello, undefined.
fun_one ("Hello", undefined, "Hello"); // Hello, Hello, Hello
```

ex 4)

```
function fun_one (arg1 : String,
                  arg2 : String = "ReactJs",
                  arg3 ?: String,
                  ...arg4 : String[]) : Void {
    console.log(arg1, arg2, arg3, arg4);
};
```

fun_one(); // Expected at least 1 arguments but got 0.

fun_one("ReactJs"); // ReactJs ReactJs undefined []

fun_one("ReactJs", "undefined", "ReactJs", "ReactJs");

// ReactJs ReactJs ReactJs ['ReactJs']

fun_one("ReactJs", null, "ReactJs", "ReactJs");

// ReactJs null ReactJs ['ReactJs']

Ex: console.log(undefined == undefined); // true

console.log(undefined === undefined); // true.

console.log(null == null); // true

console.log(null === null); // true.

console.log(undefined == null); // true.

console.log(undefined === null); // false.

23.10.19 :- Array Manipulations :

1) map()
2) filter()
3) reduce()
4) reduceRight()
5) forEach()
6) for ... of
7) for ... in
8) Push()
9) POP()
10) unshift()
11) Shift()
12) splice()
13) slice()
14) Delete
15) length
16) indexOf()
17) findIndex()
18) find()
19) lastIndexOf()
20) Include()
21) every()
22) Some()
23) fill()
24) Reverse()
25) Join()
26) Sort()
27) toString()
28) Flat()
29) flatMap()
30) CopyWithin()
31) Concat()
32) trim()
33) StartTrim()
34) endTrim()
35) Split()
36) Substr().

1. Map() :- This method used to manipulate the array elements.

Ex 1:

```
Console.log(
    [1, 2, 3, 4, 5].map((element, index) => {
        return "$" + element;
    })
);
```

append $

o/p :- ['$1', '$2', '$3', '$4', '$5']

Ex 2 :-

```
Console.log(
    [1, 2, 3, 4, 5].map((element, index) => {
        return element * 100
    })
);
```

o/p [100, 200, 300, 400, 500]

2. filter() : This function used to create the new array based on condition.

Ex-1

```
Console.log(
    [10, 20, 30, 40, 50].filter((element, index) => {
        return element >= 30;
    })
); // [30, 40, 50]
```

3. reduce() :- This function used to find the sum of array elements from left to right.

```
Console.log(
    [1, 2, 3, 4, 5].reduce((firstvalue, nextvalue) => {
        return firstvalue + nextvalue;
    })
); o/p : -15
```

[1,2,3,4,5]

firstvalue	nextvalue	
1	2	1+2=3
3	3	3+3=6
6	4	6+4=10
10	5	10+5=15

4. reduceRight() :- This function used to find the sum of array elements from right to left.

```
Console.log(
    ["ES6", "to", "welcome"].reduceRight((firstvalue, nextvalue) => {
        return firstvalue + " " + nextvalue;
    })
); // welcome to ES6.
```

5) forEach() :- It is used to iterate the array of elements.

```
[10, 20, 30, 40, 50].forEach((element, index) => {
        console.log(element, index);
});  // o/p:- 10 - 0      40 - 3
                20 - 1      50 - 4
                30 - 2
```

6) for ... of :- This loop is the advanced loop to iterate the array of elements.

usage of for...of loop is simple compared to other loops.

```
for (let value of [10, 20, 30, 40, 50]) {
        console.log(value);
}   // 10, 20, 30, 40, 50
```

7) for ... in :- This loop used to iterate the Json object.

```
for (let key in {P_id: 111, P_name: "P_one", P_cost: 1000}) {
        console.log(key, typeof(key));
});  // o/p:  P_id    String
              P_name  String
              P_cost  String
```

8) Push() :- It is used to add the element at the end of the array.

9) unshift() :- It is used to add the element to at the begining of array

10) Pop() :- function is used to remove the end element

11) Shift() :- function is used to remove the begining element.

```
let arr = [20, 30, 40];
Console.log(arr);   // [20, 30, 40]
arr.Push(50);
Console.log(arr);   // [20, 30, 40, 50]
arr.unshift(10);
Console.log(arr);   // [10, 20, 30, 40, 50]
arr.Pop(50);
Console.log(arr);   // [10, 20, 30, 40]
arr.Shift();
Console.log(arr);   // [20, 30, 40].
```

12) Splice() :- It is used to remove/add the elements from array.

```
let arr = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];
arr.splice(5, 2);
Console.log(arr) // [0, 1, 2, 3, 4, 7, 8, 9]
```

arrr.splice(4, 2); → index

Console.log(arrr); // [0, 1, 2, 3, 8, 9]

arrr.splice(2,1); → index

Console.log(arrr); // [0, 1, 3, 8, 9]

arrr.splice(2,0,2);

Console.log(arrr); // [0, 1, 2, 3, 8, 9]

arrr.splice(4, 1, 4, 5, 6, 7, 8); (4 → index no, 1 → no of deleted & Remove e, 4,5,6,7,8 → add no)

Console.log(arrr); // [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

24.10.19 **Slice()** :- It is used to extract

1) the Particulare Part of arrray.

arrr1 = [10, 20, 30, 40, 50];

arrr1.slice(2,4);

Console.log(arrr1.slice(1,4)); // [20, 30, 40]

2) arrr2 = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

Console.log(arrr2.slice(7)); // [80, 90, 100]

Console.log(arrr2.slice(5, -2)); // [60, 70, 80]

Delete() It is used to delete the Particulare element in arrray based on index.

let arrr = [10, 20, 30, 40, 50];

delete arrr[2];

Console.log(arrr); // [10, 20, <1 empty item> 40, 50]

Console.log(arrr.length); // 5

Length :- This Property used to find the length of arrray.

```
let arrr = [10, 20, 30, 40, 50]
console.log (arrr.length); // 5

arrr.length = 7;
console.log (arrr[5], arrr[6]); // undefined undefined
console.log (arrr[7]); // undefined
```

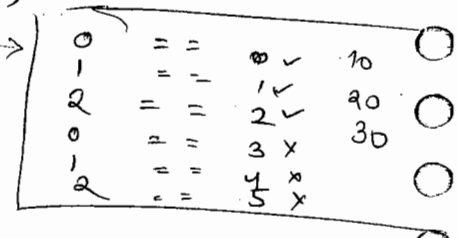
Indexof() :- indexof() won't creates indexes for duplicate elements.

```
let arrr = [0, 1, 2, 3, 0, 0, 1, 1, 3, 4]
           0  1  2  3  0  0  1  1  3  9  ← Index

arrr.foreach ((element, index) => {
    console.log (arrr.indexof (element));
3); // o/p :- 0 1 2 3 0 0 11 3 9
```

* `let arrr1 = [10, 20, 30, 20, 10, 40, 50, 10, 20, 60];`

```
arrr1.foreach ((element, index) => {
    console.log (arrr1.indexof (element));
3); // o/p :- 0 1 2 1 0 5, 6 0 1 9
```

```
let arr2 = [10, 20, 30, 10, 20, 30]
arr2.filter((element, index) => {
return arr2.indexOf(element) == index;
});   // 10, 20, 30
```



findIndex() :- To find the index of perticular element in array.

1)
```
let arr = [20, 100, 40, 50, 10, 1000];
let i = arr.findIndex((element, index) => {
    return element == 50;
});
arr.splice(i, 1);
console.log(arr);   // [20, 100, 40, 10, 1000]
```

2)
```
let arr1 = ["hello", "h1", "welcome"];
let j = arr1.findIndex((element, index) => {
    return element == "h1";
});
arr1.splice(j, 1);
console.log(arr1);   // ['hello', 'welcome']
```

3) let arrc3 = [{eno:111},
{eno:222},
{eno:333},
{eno:444},
{eno:555}];

let K = arrc3.findIndex ((element, index) => {
return element.eno == 222;
});
arrc3.splice (K, 1); // [{eno:111}, {eno:333},
{eno:444}, {eno:555}]

find() :- To check whether the element is present or not

let arrc = [10, 20, 30];
console.log (
arrc.find ((element, index) => {
return element == 20;
})
); // o/p:- 20

25/10/19

LastIndexof ( ) :- This function used to find of last index of repeated elements in array.

→ let arrc = [10, 20, 30, 10, 10, 20, 30];

console.log(arrc.lastIndexof (10)); // o/p :/→ 4

console.log(arrc.lastIndexof(20)); // 5.

Console.log (arrc.lastIndexof(30)); // 6.

)

Includes( ) :- It is a boolean method, and used to such the perticular element in array.

Console.log (
[10, 20, 30, 40, 50].includes (30)
); //true.

every ( ) :- To check each and every element in array with particular condition.

Console.log (
[10, 20, 30, 40, 50].every ((element, index) => {
return element >= 10;
})
); // true.

Some() :- It will check atleast one element in array with condition.

```
Console.log(
    [10,20,30,40,50].some((element,index) => {
        return element <=10;
    })
); // true
```

Fill() :- This function used to replace the array elements with new values.

```
arrr = ["Angular", "ReactJS", "Node"];
arrr.fill("VueJS");
console.log(arrr); // ["VueJS", "VueJS", "VueJS"]
arrr.fill("MERN", 1);
Console.log(arrr); // ['VueJS', 'MERN', 'MERN']
arrr.fill("MEAN", 0, 2);
Console.log(arrr); // ['MEAN', 'MEAN', 'MERN']
```

Reverse() :- It is used to reverse the array of elements.

```
let arrr1 = [10, 20, 30, 40, 50];
console.log(arrr1.reverse()); // [50, 40, 30, 20, 10]
let arrr2 = ["Hello", "Hi", "welcome"];
Console.log(arrr2.reverse()); ['welcome', 'Hi', 'Hello']
```

```
Console.log(Array.from("welcome").reverse().join(""));
```

welcome → emocleW

O/P: // emocleW

```
Console.log(
  ["Hello1", "Hello2"].map((element, index) => {
    return Array.from(element).reverse().join("");
  }).reverse()
); // 2olleH , 1olleH
```

1 % 2 == 0

```
Console.log(
  ["Angular", "React", "Node"].map((element, index) => {
    if (index % 2 == 0) {
      return Array.from(element).reverse().
                                    join("");
    } else {
      return element + "Naresh IT";
    }
  }).reverse();
);
```

O/P: // ['edoN', 'ReactNareshIT', 'ralugnA']

Join() - converting array to string.

→ This function used to join the sequence of Array of elements.

```
Console.log (
    ["welcome", "to", "reactJS"].join(" ")
); // welcome to reactJS
```

✱ Diff join() and toString()

→ Concantination operators is Possible in join()

" " " not Possible in toString().

Sort() :- It is used to sort the array of elements.

→ default sorting is ascending order.

```
Console.log([1, 5, 2, 4, 3].sort());
                    // 1, 2, 3, 4, 5

Console.log (
    [90, 80, 7, 6].sort ((num1, num2) => {
        return num1 - num2;
    })
); // 6, 7, 80, 90 → asending order.
```

26/10/19 :-

*
```
Console.log (
    [10, 50, 20, 40, 30].sort((num1, num2) => {
        return num2 - num1;
    }) // [50, 40, 30, 20, 10]
```

*
```
Console.log (
    [10, 50, 20, 40, 30].sort((num1, num2) => {
        return num2 - num1;
    }) [1]
); // 40 (2nd max element)
```

```
console.log (
    [10, 50, 20, 40, 30].sort((num1, num2) => {
        return num1 - num2;
    }) [1]
); // 20 (2nd min element)
```

toString() :- This method used to convert the array to equalent string.

```
Console.log (
    ["h", 'e', "l", "l", 'o'].tostring()
); // h,e,l,l,o
```

flat() :- This function used to make the multi dimensional arrays to single dimensional arrays.

```
console.log(
    [ [1,2,3], [4,5,6], [7,8,9] ].flat(1)
); // [1,2,3,4,5,6]
```

→ To convert multidimensional array to single dimensional array.

→ This is suppoted in <html> </html> <script> </script>

```
console.log(
    [ [1], [1, [[[[[[[[[1]]]]]]]]] ].flat(Infinity)
); // [1,1,1]
```

→ flat() function will work only in HTML environment.

flatMap() :- This function is the advanced version of map().

```
let arr1 = [1,2,3];
let arr2 = ["one", "two", "three"]
console.log(
    arr1.map((element, index) => {
        return [element, arr2[index]];
    })
); // [[1,'one'], [2,'two'], [3,'three']]
```

<html> <script>

```
let arr1 = [1, 2, 3];
let arr2 = ["one", "two", "three"];

console.log(
    arr1.flatMap((element, index) => {
        return [element, arr2[index]]
    })
); // [1 "one", 2, "two", 3, "three"]
```

</script>
</html>

<html>
<script>

<script>

```
let arr1 = ["MEAN", "MERN", "MEVN"];
let arr2 = ["Angulare", "Reactjs", "vuejs"];
console.log(
    arr1.map((element, index) => {
        return [element, arr2[index]];
    }).
);
console.log(
    arr2.flatMap((element, index) => {
        return [element, arr2[index]];
    })
```

);

</script>

Imp. copyWithin() :- This function used to shift the array indexes.

→ console.log(
[0,1,2,3,4,5,6,7,8,9].copyWithin(1)
); //[0,0,1,2,3,4,5,6,7,8]

→ console.log(
[0,1,2,3,4,5,6,7,8,9].copyWithin(9)
); //[0,1,2,3,4,5,6,7,8,0]

→ console.log(
[0,1,2,3,4,5,6,7,8,9].copyWithin(2,5)
); [0,1,5,6,7,8,9,7,8,9].

→ console.log(
[0,1,2,3,4,5,6,7,8,9].copyWithin(3,8)
); //[0,1,2,8,9,5,6,7,8,9]

→

```
console.log(
    [0,1,2,3,4,5,6,7,8,9].copyWithin(3,3)
); // [0,1,2,7,8,9,6,7,8,9]

console.log(
    [0,1,2,3,4,5,6,7,8,9].copyWithin(5,2)
); // [0,1,2,3,4,2,3,4,5,6]

console.log(
    [0,1,2,3,4,5,6,7,8,9].copyWithin(2,5,7)
); // [0,1,2,5,6,4,5,6,7,8,9]
```

29/10/19 :- trim() :- It is used to remove the white spaces at begining and end of array.

trimStart() : This function introduced on ES9.

→ This function used to remove the white spaces at the begining of string.

trimEnd() :- This function also introduced in ES9. This function used to remove the white spaces at the end of string.

```
let str = "welcome to reactjs";
console.log(str);
console.log(str.trim());
console.log(str.trimStart());
console.log(str.trimEnd());
```

Concat() :- This function used to concat the arrays.

```
console.log([1,2,3,4,5].concat([6,7,8,9,10]));
        o/p:-[1,2,3,4,5,6,7,8,9,10]
console.log([1,2].concat([3,4],[5,6]));
                    // [1,2,3,4,5,6]
```

Split() :- This function applicable on strings. This function used to split the string to equalent array based on separator.

```
console.log("welcome to reactjs".split(" "));
        o/p :- ["welcome", "to" "reactjs"]
```

Substring() :- This function applicable on strings.

→ This function used to extract the perticular portion of string.

→

```
console.log("welcome to reactjs".substr(11,18);
                    // reactjs.
```

// Removing duplicates By using underscore JS :-

Need html environment.

→ underscore.js is the Javascript library.

→ " helps to Javascript to overcome the complex code

→ We will represent underscore.js by using "_"

→ How to remove duplicat element in single Array ?

-.uniq ;

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/underscore.js/1.9.1/underscore-min.js"></script>
    </head>
    <body>
        <script>
            console.log(_.uniq([1,2,3,1,2,3]));
                                    // [1,2,3]
        </script>
    </body>
</html>
```

entries() :-

→ entries function used to split the array to equalent objects.

```
let cursor = ["ReactJs", "VueJs", "Angular"].entries();
console.log(cursor.next());
console.log(cursor.next());
console.log(cursor.next());
console.log(cursor.next());
```

o/p :- { value : [0, 'ReactJs'], done : false }
{ value : [1, 'VueJs'], done : false }
{ value : [2, 'Angular'], done : false }
{ value : undefined, done : true }

JSON :- JSON used to transfer the data over the network.

→ JSON is lightweight compared to XML.

→ Parsing of JSON is Eazy compared to XML. (Reading)

→ JSON is network friendly format.

→ Syntax

1) Objects ⇒ { } → curley braces

2) Arrays ⇒ [ ] → square bracket.

3) data ⇒ Key & value Pairs.

Key & value separated by using ":"

Key & value pairs separated by using ","

JSON.JS

1)
```
let obj = {
    Sub_one : "ReactJS",
    Sub_two : "NodeJS",
    Sub_three : "MongoDB"
};

console.log( obj.sub_one , obj.sub_two , obj.sub_three);
```

o/p : ReactJS NodeJS MongoDB

2)
```
let obj = {
    Sub_one : "ReactJS",
    Sub_two : "NodeJS",
    Sub_three : "MongoDB"
};
```

for in loops iterate of JSON objects

```
for (let key in obj) {
    console.log (key, typeof (key), obj[key]);
};
```

value.

= obj of key

o/p :

Sub_one	String	ReactJS
Sub-two	String	NodeJS
Sub-three	string	MongoDB

30/10/19 :-

1)
```
let obj = {
    Sub_one : ["ReactJS"],
    Sub_two : ["NodeJS"],
    Sub-three: ["MongoDB"]
};
Console.log (obj. sub_one[0],
             obj. Sub-two[0],
             obj. Sub-three[0]); //ReactJS, NodeJS, MongoDB.
```

2)
```
let obj = {
    Sub_one : {
        Value : 'ReactJS',
        Sub_two : {
            Value : 'NodeJS',
            Sub_three: {
                Value : 'MongoDB'
            }
        }
    }
};
```

```
Console.log (obj.sub_one.value, ⊗
             obj.sub_one.sub_two.value,
             obj.sub_one.sub_two.sub_three.value);

// ReactJs NodeJs MongoDB.
```

3)
```
let obj = {
    OracleDB: ( ) => { return "oracle comm" soon !!! "},
    mysqlDB: ( ) => { return "mysql connection soon !!! "},
    MongoDB: ( ) => { return "mongodb connection soon !!! "},
    Cassandra: ( ) => { return " Cassandra connection soon !!! "},
};

Console.log ( obj.OracleDB(),
              obj.mysqlDB(),
              obj.mongoDB(),
              obj.cassandra() );  //
```

4)
```
let obj = {
    sub_one : [ {value : "Angular"},
                { value : { sub_two : "NodeJS"}} ]
};

Console.log (obj.sub_one [0].value,
             obj.sub_one [1].value.sub_two);

// Angular NodeJS.
```

5) let Products = [

```
{P_cid : 111, P_name : 'P_one', P_cost : 10000},
{P_cid : 222, P_name : 'P_two', P_cost : 20000},
{P_cid : 333, P_name : 'P_three', P_cost : 30000},
{P_cid : 444, P_name : 'P_four', P_cost : 40000},
{P_cid : 555, P_name : 'P_five', P_cost : 50000}
```

];

```
Products.forEach ((element, index) => {
console.log (index,
             element.P_cid,
             element.P_name,
             element.P_cost)
});
```

//

0	111	P_one	10000
1	222	P_two	20000
2	333	P_three	30000
3	444	P_four	40000
4	555	P_five	50000

6) How to create function ?

```
function myfun () {
    return {
        wish : wish
    }
};
function wish() {
    return {
        wish : "welcome to reactjs"
    }
```

```
};
console.log(myfun().wish().wish);
// Welcome to reacts.
```

7) Create the callback function.

```
let callback = () => {
    return {
        mern: () => {
            return "welcome to mern Development"
        }
    }
};
console.log(callback()); // Json object
console.log(callback().mern()); // welcome to mern development.
```

return Json object

Deep cloning :-

```
let obj1 = { P_id : 111 };
let obj2 = obj1;
console.log(obj1);      // { P_id : 111 }
console.log(obj2);      // { P_id : 111 }
obj1.P_id = 222;
console.log(obj1);      // { P_id : 222 }
console.log(obj2);      // { P_id : 222 }
obj2.P_id = 333;
console.log(obj1);      // { P_id : 333 }
console.log(obj2);      // { P_id : 333 }
```

Shallow cloning :- Object concatenation → Interview ques

```
let obj1 = { P_cid : 111 };
let obj2 = { P_name : 'P_one' };
let obj3 = { P_cost : 10000 };
let obj4 = { ...obj1, ...obj2, ...obj3, ...obj4 };

    console.log (obj4);

// { P_cid : 111, P_name : 'P_one', P_cost : 10000 }
```

31/10/18

In shallow cloning object occupies diff memory location. we will implement shallow cloning by using "..." spread operator → Virtual Dom concept.

```
let obj1 = { P_cid : 111 }
let obj2 = { ...obj1 }
console.log (obj1); // { P_cid : 111 }
console.log (obj2); // { P_cid : 111 }

Obj1.P_cid = 222;
console.log (obj1); // { P_cid = 222 }
console.log (obj2); // { P_cid = 111 }

Obj2.P_cid = 333;
console.log (obj1); // { P_cid = 222 }
console.log (obj2); // { P_cid = 333 }
```

EX2 :- create a html file like index.html.

<!--
Read the JSON Data from following URL
URL : https://restcountries.all/rest/v2/all.
- above rest api URL gives the JSON as response.

→ To know the structure of complex JSON by visiting following :- Jsonviewer, Stack.bm (3rd party website)

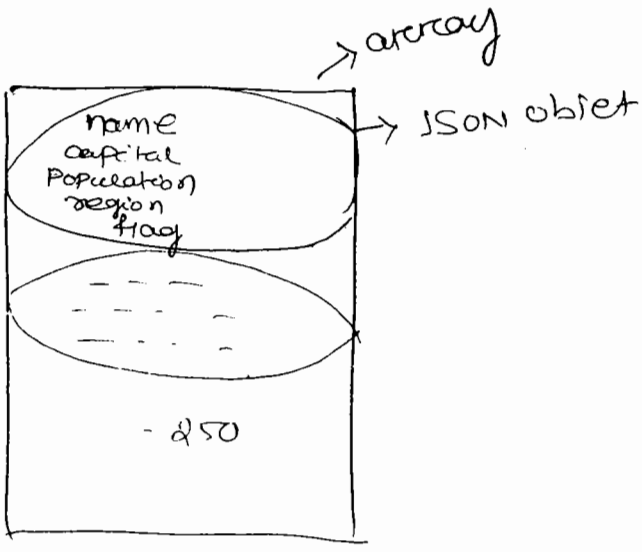
URL : http://Jsonviewer.Stock

copy code from previous website click viewer button.

Structure of JSON :-

→ Initially we have JSON array
→ we will iterate JSON array by using for(), foreach() or for ... of loop.
→ JSON Array elements are JSON objects.
→ Each JSON objects contain following keys.

@ name
@ capital
@ Population
@ region
@ flag



```
<!DOCTYPE html>
<html>
    <script>
        let arrc = "copy the data from URL1";
        document.write (
<Body>
            <table border = "1"
                    cell padding = "10px"
                    cell spacing = "10px"
                    align = "center" >
    <thead style = "background-color : grey" >
        <tr>
            <th> SNO </th>
            <th> NAME </th>
            <th> CAPITAL </th>
            <th> POPULATION </th>
            <th> REGION </th>
            <th> FLAG </th>
        </tr>
    </thead>
    <tbody>
);
        arrc. foreach ((element, index) => {
            document.write (<tr>
            <td> ${index+1} </td>
            <td> ${element.name} </td>
            <td> ${element.capital} </td>
            <td> ${element.population} </td>
            <td> ${element.region} </td>
```

```
<td> <img width = "100px"
          height = "50px"
     src = "${element.flag}"> </td>
</tr>;
});
   document.write('</tbody></table>')
</script>
</html>
```

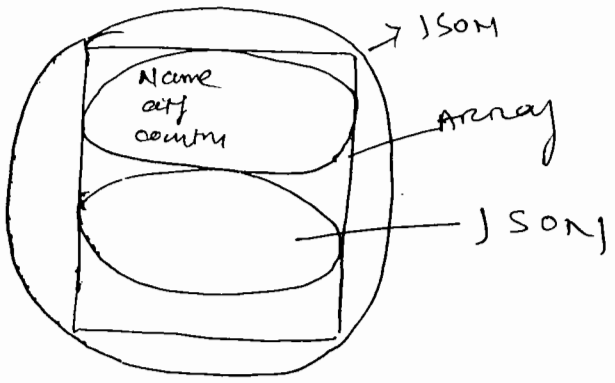
Read the JSON data from following rest api call.

URL : https://www.w3schools.com/angular/customers.php.

Structure :- Initially we have JSON object

- JSON object contains "records" Array.
- records Array contains 15 JSON object.
- Each JSON objects contain following keys.
  - @ Name
  - @ city
  - @ country.

→ we can iterate only array not JSON objects.



2.11.19.

consume the promise

```
let Promise1 = new Promise ((resolve, reject) => {
    setTimeout(() => {
        resolve ("Promise1");
    },0);
});
let Promise2 = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve ("Promise2");
    }, 5000);
});

let Promise3 = new Promise ((resolve, reject) => {
    setTimeout (() => {
        resolve ("Promise3");
    }, 10000);
});
```

Consume the Promise

```
/*
Promise1.then ((posRes) => {
    console.log (posRes); // Promise 1 immediately
});

Promise2.then ((posRes) => { // Promise 2 after 5 sec
    conside.log (posRes);
});

Promise3.then ((posRes) => { // Promise 3 after 10 sec.
    console.log (posRes);
});
```

```
Promise.all([Promise1, Promise2, Promise3]).
                                    then((POSRes) => {
    Console.log(POSRes[0]);
    Console.log(POSRes[1]);
    Console.log(POSRes[2]);
});

// Promise 1
// Promise 2
// Promise 3
// Above result will show after 10sec.
// all Promises will execute after 10 sec.
```


JSON SERVER :-

→ Json servers used to develop the rest apis.

→ By using JSON servers we can create our own rest apis.

→ Json servers supports only JSON files (data).

→ Json servers develops Rest API's based on JSON data.

→ JSON servers is light weight servers.

→ JSON Servers by default running on Port no. 3000

→ We will install JSON Servers by using following command.

> npm install -g json-server@latest (Run : cmd)

> "npm" stands for node Package manager.

> "npm" is inbuilt tool of nodejs.

> "-g" stands for global installation.

> latest version of json server is 0.15.1

* We will deploy json files by using following command.

> json-server --watch demo.json (How to deploy file into server ?)

We will test rest api calls by using Postman.

→ "Postman" is the software used to test the rest api calls.

→ website : https://www.getpostman.com/downloads/

file : Postman-win64-7.10.0-Setup.exe.

Steps to develop the rest APIs :-

1) Create the demo.json file on Desktop.

```
{
  "Products" : [
    {"id": 1, "name": "laptop", "cost": 10000, "qty": 10},
    {"id": 5, "name": "macbook", "cost": 50000, "qty": 50},
    {"id": 2, "name": "watch", "cost": 20000, "qty": 20},
    {"id": 4, "name": "Smart watch", "cost": 40000, "qty": 40},
    {"id": 3, "name": "desktop", "cost": 30000, "qty": 30}
  ]
}
```

2) deploy the demo.json in Json server.

> cd desktop

> Json-server --watch demo.json.

3) Test the following rest apis by using Postman.

⇒ http://localhost:3000/Products

- above rest api call representing "GET" Request
- above rest api call gives the all Products in the form of a array.

⇒ http://localhost:3000/Products?_sort=id&_order=asc

- above rest api call also GET Request.
- above rest api call sort the data based on "ID" and arranges in ascending orders.

⇒ http://localhost:3000/Products?cost_gte=20000&cost_lte=40000

- above rest api call also representing GET Request
- gte stands for ">="
- lte stands for "<="
- above rest api call shows the result bet^n 20000 to 40000 cost.

⇒ http://localhost:3000/products?cost_ne=30000

- above rest api calls also GET Request, It will excludes the "30000" cost record.
- "ne" stands for not equal.
- If we want to delete a perticular record

⇒ http://localhost:3000/products?q=macbook

- above rest api calls searches perticular record form JSON Server.
- "q" stands for search.

⇒ http://localhost:3000/products?_limit=3.

- above GET rest api calls gives the first 3 records.

⇒ http://localhost:3000/products?_start=1&_end=5

→ "-start & _end" depending on indexes of JSON objects.

→ Indexes start with 0.

→ In above case 1, 2, 3, 4 indexed record will be displayed.

⇒ http://localhost:3000/products?name_like=watch

"_like" Just nothing but filter.

- It will filter matched word in all records.

```
let class_one = function() { };
// add function dynamically as per adv javascript.
class_one.prototype.getSubone = function() {
        return "ReactJS";
    };
// child class
let class_two = function() { };
// Make the parent child relationship.
class_two.prototype =
        object.create(class_one.prototype);
// add function dynamically.
class_two.prototype.getSubTwo = function() {
        return "NodeJS";
    }
// - Sub child class
let class_three = function() { };
// Make the parent child and subchild relationship.
class_three.prototype = object.create
    (class_two.prototype);
// add the function dynamicaley
class_three.Prototype.getSubThree = function() {
        return "MongoDB";
    };
```

Prototype chaining :- 6/11/19 :-

→ Inhericitance is nothing but Prototype chaining.

→ ~~Inheritance~~ → getting the Properties from parent class to child class called as Prototype chaining.

→ create() is used to Prototype chaining.

```
let class_one = function() { };
// add function dynamically
class_one.Prototype.fun_one = function() {
    return " i am from class one !!! ";
};
let class_two = function() {
};
// Make the class_two as child class of class_one.
class_two.Prototype = object.create
    (class_one.prototype);
// add function dynamically.
class_two.Prototype.fun_two = function() {
        return "i am from class two !!! ";
};
    let obj = new class_two();
console.log(obj.fun_one(), obj.fun_two());
// i am from classone !!! I am from class two !!!
```

⇒ http://localhost:3000/Products (POST)

- above request is the Post Request.
- above Request add the record to JSON Server.
- above request carrying the JSON object as body Parameters.

⇒ http://localhost:3000/Products/6 (PUT)

- above request is Put request.
- above request used to update the 6th record.
- above request takes the JSON object as body Parameters.

⇒ http://localhost:3000/Products/6 . (DELETE)

- above Request is DELETE ~~Request~~ Record.
- above Request Deletes the 6th record from JSON Servers.

Assignment :-

* Develop the rest api by using employee.json file
* Consume the rest api calls by using ajax Calls.

Ajax calls :- 4/11/18

→ Ajax calls also called Asynchronous calls.

→ Making the rest api calls without affecting the other Network called as Ajax call.

→ Ajax calls are slower in processing the requests

→ Data accuracy is high in Ajax calls.

→ We will make Ajax calls by using JQuery, AngularJS, Angular, ReactJS, VueJS, ...

Ex 1 : https://restcountries.eu/rest/v2/all

```
<!DOCTYPE html>
<html>
    <head>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
    </head>
    <body>
        <script>
            $.ajax({
                Method: "GET",
                Url: "https://restcountries.eu/rest/v2/all",
                Success: (PosRes) => {
                    console.log(PosRes);
                },
                error: (errRes) => {
                    console.log(errRes);
                }
```

```
});
</script>
</body>
</html>
```

2)

```
<script>
    // URL: https://www.w3schools.com/angularjs/
      customers.php
    $.ajax({
        Method :"GET",
        Url :"https://www.w3schools.com/angularjs/
          customers.php",
      Success:(POSRes)=>{
      // Parse method used to convert the string
  response to object response
          Console.log(JSON.Parse(POSRes));
    },
        Error:(ErrRes)=>{
            console.log(errRes);
        }
    });
</script>
```

// Series of Ajax calls

// https://restcountries.eu/rest/v2/all

// https://www.w3schools.com/angular/customers.php

```
$.ajax({
    Method : "GET",
    Url : "https://restcountries.eu/rest/v2/all",
Success : (posRes) => {
    Console.log(posRes);
```

⇒

```
$.ajax({                    → update the data to server.
    Method : "GET",
    Url : "https/www.w3schools.com/angular/
    customers.php",
  Success : (posRes) => {
        Console.log(posRes);
    },
  error : (errRes) => {
        Console.log(errRes);
    });
},
error : (errRes) => {
    Console.log(errRes);
}
```

4) * POST Request :-

URL: http://test-routes.herokuapp.com/test/uppercase

```
$.ajax({                                  send the data to server.
    Method : "POST",
    Url : "http://test-routes.herokuapp.com/test/uppercase",
    data : {"message" : "facebook reactJS"},
    Success : (PosRes) => {
        Console.log(PosRes);
    },
    Error : (errRes) => {
        console.log(errRes);
    }
});
```

5) Ajax-Calls, Promises and async & await :-

```
let Promise =
    new Promise((resolve, reject) => {
    $.ajax({
        Method : "GET",
        Url : "https://restcountries.eu/rest/v2/all",
    Success : (PosRes) => {
        resolve(PosRes);
    },
```

```
        error: (errRes) => {
            reject (errRes);
        }
    });
});
    async function myfunc() {
        let result = await promise;
        console.log (result);
    };
    myfunc();
</script>
</body>
</html>
```

Assignment :-

1) Make the Ajax call with the help of Promises and Async & await.

Url : https://www.w3schools.com/angularjs/customers.php

2) Make the ~~Box~~ Series call with the help of Promises ?

URL-1 : ~~https://www.w3schools.com/angular/customers.php~~
https://restcountries.eu/rest/v2/all

URL-2 :

OOPS in JS

demo.js → file. 5/11/19

→ Javascript Supports the few oops concept.
→ JS is not the object oriented Scripting/language
→ Javascript is the object based scripting language.
→ Constructors

Constructors :-

→ It is used to creat the "objects."
→ In order to create the objects, we need "Constructor functions".
→ In Constructor functions everything should start with "this" keyword
→ "this" keyword used to refers the current class members.
→ We will create the object to constructors functions by using "new" keyword.

1)
```
let class_one = function() {
    this.sub_one = "ReactJS";
    this.sub_two = "NodeJS";
    this.sub_three = "MongoDB";

    this.getsubone = function() {
        return this.sub_one;
    };
```

```
this.getSubTwo = function() {
        return this.sub_two;
};

this.getSubThree = function() {
        return this.sub_three;
    };
};

let obj = new class_one();
Console.log( obj.getSubOne(),
             obj.getSubTwo(),
             obj.getSubThree());
// Reacts Nodejs MongoDB.
```

2) If we use "use strict" it will throws manual error.

```
let class_one = function() {
        this.wish = "good evening...!";
    };
    let obj = new class_one();
        Console.log(obj);
// class_one { wish : 'good evening...!' }
```

3) let class_two = function() {
    this.wish = "hello";
};
let obj = class_two();
console.log(obj); // undefined.

throw the error

"use strict"

let class_three = function() {
    this.wish = "good evening...!";
};

let obj = class_three();
console.log(obj);
// TypeError: Cannot set property 'wish' of undefined.

* "Use Strict"

let class_four = function() {
    if (!new.target) throw "my new custom exception";
    this.wish = "good evening...!";
};

(Predefined API used to throw the custom exception)

let obj = class_four();
console.log(obj); // my new custom exception.

(index.html)

```
<script>
let class_five = function () {
    this.color = "green";
};
let obj = class_five();
console.log (window.color); // green
```

→ ~~In constructor functions we will access~~
too

```
</script>
</html>
```

→ In constructor function we will access properties with the help of window object.

→ this example need HTML environment.

→ window is the predefined is object

→ window is the global object.

Parameterized Constructor :-

```
let my_class = function (arg1, arg2, arg3) {
    this.sub_one = arg1;
    this.sub_two = arg2;
    this.sub_three = arg3;
};
let obj1 = new my_class ("Angular", "NodeJs",
                         "mysql");
console.log (obj1.sub_one, obj1.sub_two,
             obj1.sub_three);
let obj2 = new my_class ("Reactjs", "NodeJs", "mongodb");
```

```
Console.log (obj2.sub_one, obj2.sub_two, obj2.sub_three)
```

Adding the functions dynamically :-

```
let my_class = function (arg1, arg2, arg3) {
        this.var_one = arg1;
        this.var_two = arg2;
        this.var_three = arg3;
    };

// Prototype is the predefined property on constructor functions.
// Prototype used to add the dynamic properties (functions)

my_class.prototype.getvarone = function() {
                                  (function.)
        return this.var_one;
    };

my_class.Prototype.getvartwo = function() {
        return this.var_two;
    };

my_class.Prototype.getvarThree = function() {
        return this.var_three;
    };

let obj = new my_class ("ReactJs", "NodeJs", "mongoDB");
        Console.log (obj.getvarone(),
                     obj.getvartwo(),
                     obj.getvarthree());
```

Overriding the Predefined toString :-

Interview Question ~~let obj = {~~ I dnt want to see object object I want to see custom message.

```
let obj = {
    wish: "nareshit"
};
console.log(obj);  { wish: 'nareshit' }
console.log(obj.toString());  // [object object]
Object.prototype.toString = function(){
    return 'welcome to ${obj.wish}';
};
console.log(obj.toString());  // welcome to nareshit
```

```
// create the object to Subchild
let obj1 = new class_three();
Console.log(obj1.getSubOne(),
            obj1.getSubTwo(),
            obj1.getSubThree());
        // ReactJs NodeJS MongoDB
// create the object to child class
let obj2 = new class_two();
Console.log(obj2.getSubOne(), obj2.getSubTwo());
// ReactJs NodeJS
// create the object to Parent class
let obj3 = new class_one();
Console.log(obj3.getSubOne()); // ReactJs
```

Function overridding :-

```
let class_one = function() {
        this.dbFun = function() {
            return "Oracle data soon---!";
        };
let class_two = function() { };
class_two.prototype = object.create
        (class_one.prototype);
Class_two.prototype.dbFun = Function() {
        return "Firebase Data soon---!";
};
```

```
let obj = new class_two();
console.log(obj.dbFun());
// Firebase Data soon --- !
}
```

→ Calling Superclass constructor on adv javascript.

```
let class_one = function(arg1) {
        this.arg1 = arg1;
};
let class_two = function(arg1, arg2) {
        object.call(this.arg1);
        this.arg2 = arg2;
};
class_two.Prototype = object.create
        (class_one.Prototype);
let obj = new class_two("Hello_1", "Hello_2");
console.log(obj); // class_one {arg2: 'Hello_2'}
```

7/11/19 :-

→ Callbacks ✓
→ callback hell ✓
→ IIFE (Immidiate Invokable functional Expressions)
→ Generators & Iterators ✓
→ Event bubbling & Event capturing
→ closures ✓
→ Call, apply, bind methods ✓
→ Debugging JS Appl^n ✓
→ Data structures (Map, weakmap, set, weakset) ✓
→ ES2020

React JS

→ Introduction
→ Components in ReactJS ✓
→ Middlewares
  - Redux ✓
  - SAGA
  - Thunk
→ SPA
→ MERN Development
→ Databases
  - MYSQL
  - SQLSERVER
  - MongoDB
→

demo.JS → file

Generators :-

dynamically I want to produce.

- Generators are introduced in ES6
- Generators are used to produce the values dynamically.
- Generators place the important role while implementing the middleware (ex: SAGA, Redux, Thunk) in ReactJs.
- Generators fun^n should start with "*"
- In general Generators functions will return "cursor" Mechanism (cursor are used to read the dynamically generated values)

false → next value present.
true → I am a last.

```
function *fun_one() {
    Yield 1;
    Yield 2;
    Yield 3;
    Yield 4;
    Yield 5;
};
let cursor = fun_one();
Console.log(cursor.next());
        // { value: 1, done: false }
Console.log(cursor.next());
        // { value: 2, done: false }
```

```
Console.log( cursor.next() );
                // { value : 3, done : false }
Console.log ( cursor.next() );
                // { value : 4, done : false }
Console.log ( cursor.next() );
                // { value : undefined, done : true }
```

2) ~~Hoo~~ function *fun_one ( ) {

```
        Yield 1 ;
        Yield * fun_two ();
        Yield 3;
function *fun_two ( ) {
        Yield 2;
};
let cursor = fun_one ( );
Console.log ( cursor.next ( ) );
                // { value : 1 , done : false }
Console.log ( cursor.next ( ) );
                // { value : 2, done : false }
Console.log ( cursor.next ( ) );
                // { value : 3, done : false }
Console.log ( cursor.next ( ) );
                // { value : undefined, done : true } ?
```

3)

```
function *fun_one() {
    yield 1;
    return "Hello";
    yield 2;
};
let cursor = fun_one();
console.log(cursor.next());
    // { value: 1, done: false }
console.log(cursor.next());
    // { value: 'Hello', done: true }
console.log(cursor.next());
    // { value: undefined, done: true }.
```

4)

```
function *fun_one() {
    for (let i=0; i<5; i++) {
        yield i;
    };
};
let cursor = fun_one();
for (let i=0; i<6; i++){
    console.log(cursor.next());
}

// { value:0, done: false }
// { value: 1, done: false }
// { value: 2, done: false }
// { value: 3, done: false }
```

// { value : 4 , done : false }
// { value : undefined , done : true }

Iterators

// for loop
// for each()
// for ... of
// for ... in

→ As per ES6 standard we have following iterators

```
let arr = ['Angular', 'NodeJs', 'ReactJs']

// Iterate by using for loop.
for(let i=0; i<arr.length; i++) {
    console.log(arr[i]);
};
                    // Angular
                       NodeJs
                       ReactJs

// Iterate by using forEach loop
arr.forEach(element => {
    console.log(element);
});
                    // Angular
                       NodeJs
                       ReactJs
```

// for --- of loop

```
for(let value of arr) {
    console.log(value);
};      // Angularjs
        // NodeJs
        // ReactJs
```

// for --- In loop is used

→ To iterate Json object

```
let obj = {
        key1 : "welcome",
        Key 2 : "to",
        Key 3 : "reactJs"
};
    for (let key in obj) {
            console.log(key); // key1 key2 key3
            console.log(typeof(key)); // String
                                        String
                                        String
        console.log(obj[key]);
                        // welcome to reactJs
    };
```

8/11/19

Callbacks :- Passing one function defination as argument to another function called as callback function.

→ we will run JS appln in watch mode.

→ to run JS appln in watch mode we need following tool "nodemon".

→ we will install "nodeman" tool by using following command

→ npm installs -g nodemon@latest.

→ We will execute JS appln in watch mode by using following comand.

→ nodemon demo.js.

①
```
function fun_one(){
    console.log("hello, welcome to callbacks");
};
function fun_two(callback){
    callback();
};
fun_two(fun_one);
```

② Create the four functions :-
```
function add(num1, num2){
    return num1+num2;
};
function sub(num1, num2){
    return num1-num2;
};
```

```
function mul (num1, num2) {
    return num1 * num2 ;
};
function div (num1, num2) {
    return num1/num2;
};
function calc (arg1, arg2, callback) {
    switch (callback) {
    case add :
        console.log ("Addition ---", callback (arg1, arg2));
        break;
    case sub :
        console.log ("substraction --", callback(arg1, arg2));
        break;
    case mul :
        conse.log ("multiplication ---", callback(arg1, arg2));
        break;
    case div :
        console.log ("division ---", callback(arg1, arg2));
        break;
    }
};
        calc (10,10, add);
        calc (10, 10, sub);
        calc (10, 10, mul);
        calc (10, 10, sub);
```

③
```
Set timeout( function myfun() {
        Console.log ("hello");
},5000);          // Hello.
```

④
```
Set Timeout (function () {
        Console.log( "Hello");          // Hello
},5000);
```

⑤
```
let demo = function() {
        Console.log ("hello");
};
Set Timeout (demo, 5000);          // hello.
```

⑥
```
function fun_one (callback, time) {
    Set Timeout (callback, time);
     Console.log ("welcome");
};
function fun_two () {
        Console.log ("hello");
};
fun_one (fun_two, 5000);          // welcome
                                  // hello
```

fs module :-

→ fs is the predefined ES6 module.

→ fs module used to read the data from files.

→ fs module entirely developed by using callbacks.

→ we will import any predefined module in ES6 by using following function require().

```
let fs = require("fs");
fs.readfile("sample.txt", function(err, res) {
        if (!err)
            console.log(res.tostring());
});
```

(2 read the data)

```
function myFun() {
    $.ajax({
        Method : "GET",
        Url : "URL",
        Success : function (posRes) { },
    // If ajax call Success response will be stored
    into callback function first position.
        Error : function (err) {
    // If ajax call fails Error response will
    be stored into callback function first
    position.
        }
    });
}
```

Data Structure :- 11.11.2019

→ Data Structure introduced in ES6.

→ We have following Data structures.

- @ Map
- @ weakmap
- @ Set
- @ weakset

→ Data Structure allows us to Persist the data in key & value pairs.

→ in Data structure we can use keys as arbitory values.

→ By using Datastructure, we can add "JSON ~~keys~~ objects" as keys in "JSON objects".

```
<!DOCTYPE. html>
<html>
    <script>
        let obj = {};
        let key1 = {};
        let key2 = {};
    obj [key1] = "Hello_1";
    obj [key2] = "Hello_2";
      console.log (obj);
</script>  // {[object object] : "Hello_2"}
</html>
```

→ In the above example we added key1 & key2.
→ but JS allows us add only one key
→ always Javascript shows the latest value.
→ In order to overcome above problem in ES6 they introduced Data structures.

2)

```
<!DOCTYPE html>
<html>
  <script>
    let key1 = {};
    let key2 = {num:1};
    let map = new map();
    map.set(key1, "Hello_1");
    map.set(key2, "Hello_2");
    console.log(map);
    // {...} => "Hello_1", {-} => "Hello_2"}
  </script>
</html>
```

3)

```
<!DOCTYPE html>
<html>
  <script>
    let map = new Map();
    map.set("key1", "a").
      set("key2", "b")
      set("key3", "c");
```

```
        console.log(map);
// {"key1" => "a", "key2" => "b", "key3" => "c"}
    map.set("key1", "d");
    console.log(map);
// {"key1" => "d", "key2" => "b", "key3" => "c"}
// Map won't allows duplicate keys
    console.log(map.size);
    map.delete("key1");
    console.log(map);
// {"key2" => "b", "key3" => "c"}
    for(let [k,v] of map.entries()){
        console.log(k,v);
    }
    // key2 b
    // key3 c
    console.log(map.keys());
    // MapIterator {"key2", "key3"}
</script>
</html>
```

4)

```
<!DOCTYPE Html>
<html>
    <Script>
        let arcy = [1,2,3]
        let map = new Map();
        map.set("key1", "Hello_1");
        console.log(map);
                // {"key1" => "Hello_1"}
    Console.log(map.get("key1"));
                // Hello_1
// Conversion from map to array
// to convert map to array we will use
 Spread operator
        let arcy = [...map];
        Console.log(arcy);
                // ["key1", "Hello_1"]
</Script>
</html>
```

5)

```
<!DOCTYPE html>
<html>
  <script>
    // Javascript block
      let obj = {
          a: [1, 2]
      };
    var weakmap = new weakmap();
  weakmap[obj] = "Hello_1";
  }
   Console.log(weakmap);
```

→ Weakmap won't allows the scoped member to outside.

Set :- Set is the data structures introduced in ES6.

→ Set allows us to add ~~Key & Value pairs~~ the arbitory values.

→ Set won't allows duplicate values.

```
<!DOCTYPE html>
  <html>
   <script>
      let set = new set();
        Set.add (1);
        Set.add(2);
        Set.add(3);
        Set.add(4);
        Set.add(1);
        Set.add(3);
       Console.log(set);    // {1, 2, 3}
   </script>
```

13/11/19 :-

→ Set is the data structures.
→ Set introduced in ES6.
→ Set won't allows the duplicate values.
→ Set allows us to add only orderless values.
→ We can create set by using "Set class" constructor.

// add - used to add the element to set.
clear() - used to delete (clear) the data from set.
constructor() - used to create the set.
delete - used to delete the particulare item from set.
entries() - used to iterate the set.
has() - used to check the particulare element in set
keys() - used to get the keys.
size() - used to know the size of set.
values() - used to get the values from set.

```
<!DOCTYPE html>
<html>
  <script>
      let set = new set();
        set.add(1);
        set.add(2);
        set.add(3);
      Console.log(set);   //{1,2,3}
```

```
        console.log(set.has(3)); // true

    set.delete(3);
    console.log(set); // {1, 2}

    for (let value of set) {
        console.log(value); // 1, 2
    };

    set.clear();
    console.log(set); // {}
</script>
</html>
```

2)

```
<!DOCTYPE html>
<html>
    <script>
        var set = new set();
        set.add(1);
        set.add(2);
        set.add(3);
        console.log(set); // {1, 2, 3}
    // Convert set to array.
        var ary = [...set];
        console.log(ary); // [1, 2, 3]
    var ary1 = [Array.from(set)];
        console.log(ary1[0]); // [1, 2, 3].
</script>
</html>
```

3)
```
<!DOCTYPE html>
<html>
  <Script>
    let arcy = [1,2,3,4,5,1,2,3];
    let set = new set(arcy);
    Console.log(set); // {1,2,3,4,5}
    Console.log [... new set(arcy)]); // [1,2,3,4,5]
// removes duplicate elements and returning the
equalent arcray.
  </Script>
</html>.
```

4)
```
<!DOCTYPE html>
<html>
  <script>
    let set = new Set();
    Set.add({a:1}).add({b:2});
    Console.log(set) // {{...} {...}}
    Set.add(3);
    for (let value of set) {
      Console.log(value); // {a:1}
                          // {b:2}
    };
    Console.log(set.has({a:1})); // false.
// has function works with only primitives.
// has function won't throws the appropriate.
results for referenced data types.
```

```
        console.log (set.has(3)); // true.
    </Script>
    </html>
```

Weakset :-

```
<!DOCTYPE html>
<html>
    <Script>
        Var ws = new weakset();
        Console.log (ws);
        ws.add ({a:13});
        Console.log (ws); // {{...}}
        Console.log (ws.has({a:13})); // false.
    </Script>
```

Github :-

→ Setting → Settings → git enabled

→ Source control go →

https://github.com/ajsnaresh/ReactJS

→ github.com

↓

New

↓

Repository name (ReactJS_classes

↓

Create_repository

↓

https://gitub.com/naveshajs/React_JS_classes.git.

→ How to Push the code ?

VS → drag and drop folder → file create

↓

Push the file to cloude

↓

settings

↓

git enable

↓

enabled

↓

Source control Go

↓

+

↓

test Commit

↓

Yes

↓

Go → Push.

↓

View → Command → Previous. Url.

↓

View → terminal

↓

git init

↓

git config --global user.email Suresh@gmail.com

↓

git config --global user.name Ram

Callback Hell :-

→ Writing one callback in another callback called as Callback hell.

→ Callback is the big issue in development.

→ Never use callback hell while developing the appl^n

→ Debugging of callback appl^n are difficult.

// create the addition function.

```
function addition (val, callback) {
    return callback ((val+5), false);
};

addition (5, function (addRes, err) {
    if (!err) {
        console.log (addRes);
    }
});
```

2)

```
function addition (val, callback) {
    return callback ((val+5), false);
};

function Substraction (val, callback) {
    return callback ((val-3), false);
};

function multiplication (val, callback) {
    return callback ((val*1), false);
};
```

```
function division (val, callback) {
    return callback ((val/2), false);
};

addition (5, function (addRes, err) {
    if (!err) {
        Substraction (addRes, function (subRes, err) {
        if (!err) {
            multiplication (SubRes, function (mulRes, err) {
            if (!err) {
                division (mulRes, function (divRes, err) {
                if (!err) {
                    console.log (divRes);
                }
                });
            }
            });
        }
    });
```

Solution for callback :-

→ To overcome "callback hell" we have Promises, async & await.

→
```
let promise = new Promise ((resolve, reject) => {
        resolve (10);
});

async function myfun () {
        let res = await promise;
        console.log (res);
};
```

```
function addition(val) {
  return new Promise((resolve, reject) => {
      resolve((val+5));
  });
};
  async function consume() {
      let res = await addition(5);
      console.log(res);
  };
  consume();

/*  Consume two promises :-

function addition(val) {
  return new promise((resolve, reject) => {
      resolve(val+5);
  });
};
function Substraction(val) {
  return new Promise((resolve, reject) => {
      resolve(val-3);
  });
};
async function consume() {
    let addRes = await addition(5);
    let SubRes = await Substraction(addRes);
    console.log(SubRes);
};
consume();
```

→ debug - ctrl+B → sources

15|11|19 :- Call(), Apply(), Bind() :-

→ call(), apply(), & Bind() functions are used to establish the relationship bet^n objects and functions temporarily.

1)
```
let obj = {
    num: 1
};
let myfun = function (arg1) {
    return this.num + arg1;
};
console.log (myfun.call (obj, 1)); //2
```

2)
```
let obj = {
    a: 1,
    b: 2
};
let myfun = function (c, d, e) {
    console.log (this.a + this.b + c + d + e);
};
myfun.call (obj, 3, 4, 5); //15
```

→ // To establish rel^n bet^n object & function temporarily and passing parameter independently called as call().

3)
```
function myfun() {
    console.log(arguements);
        // {'0':1, '1':2, '2':3}
    // convert to Array
    console.log(Array.from(arguements));
                                // [1,2,3].
};
  myfun(1,2,3);
```

4)
```
function class_one (arg1) {
        this.arg1 = arg1;
}

function class_two (Param1, Param2) {
        class_one.call (this, Param1);
          this.Param2 = Param2;
};
let obj = new class_two ("Hello_1", "Hello_2");
      console.log (obj);
```

5)
```
let obj = {
      num:1
};
let arr = [1,2,3];
let myfun = function (arg1) {
        console.log (this.num + arg[0]
};
```

5)

```
let obj = {
    num : 1
};
function myfun (arg1, arg2, arg3) {
    console.log (this.num + arg1 + arg2 + arg3);
};
myfun.call (obj, 1, 2, 3); // 7
myfun.apply (obj, [1, 2, 3]); // 7
```

// by using apply () function we can supply array of values.

6)

```
let obj = {
    a : 1,
    b : 2
};
let myfun = function (arg1, arg2, arg3) {
    console.log (this.a + this.b + arg1 + arg2 + arg3);
};
myfun.call (obj, 1, 2, 3); // 9
let arrr = [1, 2, 3];
myfun.apply (obj, arrr);
```

7)

```
let obj = {
    num : 1
};
let myfun = function (arg1) {
    console.log (this.num + arg1);
};
```

```
let myfun1=myfun.bind (obj);
    myfun1(1); //2
```

Bind()

→ to add object to function by default we will use bind().

8)
```
let obj = {
    a: 1,
    b: 2,
    c: 3
};
let myfun = function (arg1, arg2, arg3) {
        console.log(this.a + this.b + this.c +
                    arg1 + arg2 + arg3);
};

myfun.bind(obj) (1,2,3); // 12
```

IIFE :-

Immediate Invokable functional Expressions.

→ IIFE Introduced in ES9.

→ IIFE overcomes the global Polluting Issues.

→ IIFE functions are self Invokable functions.

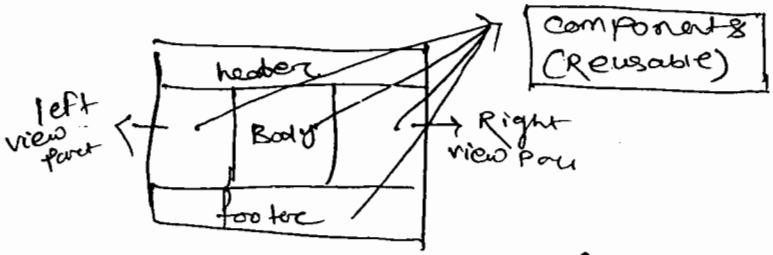
→ ~~(){}()~~. (syntax) (() => { })()

```
(() => {
    console.log ("Hello");
})(); // Hello
```

Introduction to ReactJS :-

29/11/19

→ ReactJs is the library.
→ ReactJs is the Javascript library.
→ ReactJs is given by facebooks.
→ ReactJs is used to develop the web applications.
→ ReactJs is used to simplyfy the complex UI.
→ We will develop React appln by using either JSx or typescript.
→ JSX Stands for Javascript + XML.
→ In general XML used understable to browsers.
→ So, we must convert XML to equivalent Javascript.
→ Babel is the integrated tool used to convert the XML to equivalent JS.
→ ReactJs simplifies the complex UI.
→ ReactJs Splits the complex UI to multiple Reusable UI Components.
→ Part of complex UI called as component.
→ Components are reusable in ReactJs.
→ React applicatoon are component based applications.



Environmental Setup for ReactJs application :-

→ Download and install node js.
→ node js server side scripting language.
→ Node JS used to develop the server.
→ React applications development depending on node server.
→ website : https://nodejs.org/en/download
file : node -v0.12.1-x64.msi

Download and Install Git :-

→ Git is the open source Repositary maintained by Microsoft company.

→ Git Repositary contains so many open source libraries related to React.

→ ~~Git Repositary contains~~.

https://git-scm.com/download/win.

Git-2.24.0.2-64-bit.exe ; file download.

Install "create-react-app" tool :-

→ "create-react-app" is the tool given by facebooks.

→ "create-react-app" used to create the React applications.

→ we will install "create-react-app" tool by using following command.

→ npm install -g create-react-app@latest.

→ npm stands for node packeng manager.

→ npm is the integrated tool of nodejs.

→ "-g" stands for global installation.

Creating the React Applications :-

1) create the react app.

→ create-react-app first-app.

→ where "first-app" is the react app.

→ switch to react app

>cd first-app.

→ execute the react app

>npm start (or) >yarn start.

→ by default react app running on port no 3000.

Installation

1) Install yarn tool :

→ "yarn" is the tool given by facebook.

→ "yarn" tool used to download the libraries from Git in faster manner.

→ We will install yarn tool by using following command.

> npm install -g yarn@latest.

30/11/19

Component :-

→ Part of reusable complex UI called as component.

→ React appl^n are component based applications

→

```
first-app
    src
      first.js
      index.js
```

→ "index.js" is the registration file.

→ We must register all our component in index.js file. once if we register then only it will execute

first.js :-

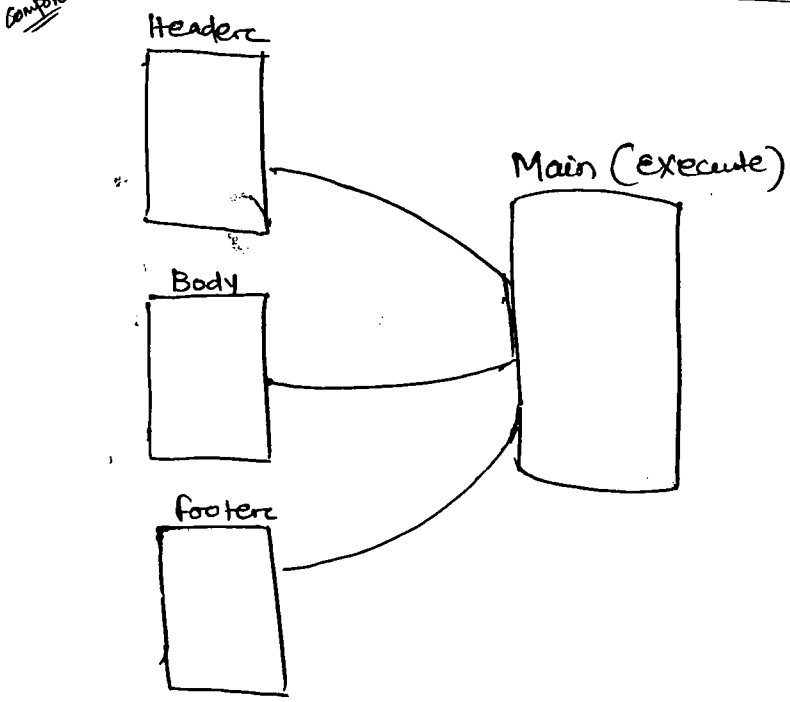
```
// import React class
// React is the predefined class, used to create the
   component environment.

// Import Component
// Component is the predefined class, used to create the
   component
   Import React, {Component} from "react";

// create and export the component.
   export default class First extends React.Component {
     // Render()
   // render() called as main method in React Component.
   // render() used to render the view.
     render() {
         return (
           <div>
             <h1> Welcome </h1>
```

```
    </div>
    )
  };
};
```

Index.JS :-

```
import React from 'react';
import ReactDom from 'react-dom';
import './index.css';
import App from './App';
import first from "./first";
import * as serviceworker from './serviceworker';

ReactDom.render(<first />,
                document.getElementById('root'));
serviceworker.unregister();
```

Two or more components onto one ?



Second-app

src

Header.JS

Body.JS

Footer.JS

Main.JS

Index.JS

Header.JS :-

```
import React, {component} from "react";
export default class Header extends React.Component {
  render() {
    return (
      <div>
        <h1 style = {{ color : 'red'}} > Header Soon </h1
      </div>
    )
  }
};
```

Body.JS :-

```
import React, {component} from "react";
export default class Second extends React.component {
  render() {
    return (
      <div>
        <h1 style = {{color : 'green'}} >
          Content Soon... !
        </h1>
      </div>
    )
  }
};
```

Footer.JS :-

```
import React, {component} from "react";
export default class Footer extends React.component{
    render() {
        return (
            <div>
                <h1 style = {{color : 'blue'}}>
                    Footer Soon ---!
                </h1>
            </div>
        )
    };
};
```

Main Component Main.JS :-

```
import React, {component} from "react";
import Header from "./Header";
import Body from "./Body";
import footer from "./footer";
export default class main extends React.Component{
    render() {
        return (
            <div>
                <Header />
                <Body />
                <Footer />
            </div>
        )
    };
};
```

Index.JS

```
Import Main from "./main";

ReactDom.render(<Main/>, document.getElementById
                                    ("root"));
```

→ View → terminal

→ @ Yarn start

O/P :- Header Soon---!
Content Soon---!
Footer Soon---!

State :- "state" is the predefined functionality in Reactjs.

- "State" used to maintain the component data.
- We will define "state" in two Places.
  1) Inside the constructor
  2) Outside the constructor.
- Defining the State inside the constructor is recommended
- We will change the "State" by using "setState()" function.

2/12/19

Ex:
```
constructor() {
  - super();
    this.state = {
      uname : "admin",
      upwd : "admin"
    }
  }
  render() {
    return (
      <div>
        {this.state.uname}
        <br>
        {this.state.upwd}
      </div>
    )
  }
};
```

Rule of State

→ Boolean value can't be display directly.
→ Map() → Iterate the arrays.
→ " ReactJs we can loop through map().

```
Import React, {Component} from "react";
export default class App extends React.Component {
  constructor(){
    super();
    this.state = {
      Sub : "ReactJs",
      version : 1.4,
      flag : true,
      Subs : ['Angular',
              'ReactJs',
              'NodeJs' ],
```

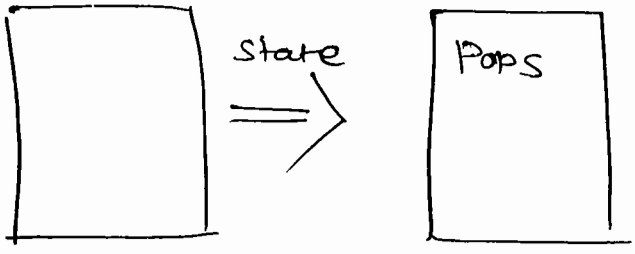
```
products :{
    P_id : 111,
    P_name : "P_one",
    P_cost : 10000
  }
};
render() {
    return (
      <div>
        <h1> {this.state.subs} </h1>
        <h1> {this.state.version} </h1>
        <h1> {JSON.stringify (this.state.flag)} </h1>
        <P> {this.state.subs.map ((element, index) => {
          <h1> {element} - {index} </h1>
        })} </P>
      <h1> {JSON.stringify (this.state.products)} </h1>
    </div>
```

Display Array in Table format :-

```
import React, {Component} from "react";
export default class Table extends React.Component{
    state = {
        Products : [
            {P_id : 111, P_name: 'P_one', P_cost : 10000},
            {P_id : 222, P_name : 'P_two', P_cost : 20000}
            {P_id : 333, P_name : 'P_three', P_cost : 30000}
            {P_id : 444, P_name : 'P_four', P_cost : 40000}
            {P_id : 555, P_name : 'P_five', P_cost : 50000}
        ]
    };
    render(){
        return(
            <div>
                <table border="1"
                    cellPadding = "10px"
                    cellSpacing = "10px"
                    align = "center">
                <thead style={{ backgroundColor : 'grey'}}>
                <tr>
                    <th> SNo</th>
                    <th> ID</th>
                    <th> Name</th>
                    <th> Cost </th>
                    <th>  </th>
```

```
        </tr>
      </thead>
      <tbody>
          {this.state.Products.map((element, index) => {
        <tr>
          <td>{index+1}</td>
          <td>{element.P_id}</td>
          <td>{element.P_name}</td>
          <td>{element.P_cost}</td>
        </tr>
      ))}
      </tbody>
    </table>
    </div>
      )
  }
};
```


Passing the data ~~beginning~~ between Components



5/12/19

→ Component receives the data from another component by using "Props"

→ "Props" is the predefined functionality used to receive the data from another component.

→ We can share the data bet^n components by using State and Props.

→ Component one sending the data by using "State"

→ Component two receiving the data by using "Props"

```
data-app
    src
      Parent.js
      Child.js
      index.js
```

→ "Parent" component want to send the data to child component.

Parent.JS :-

```
import React, {Component} from "react";
import Child from "./Child";
export default class Parent extends React.Component {
    constructor(){
      super();
      this.state = {
            wish : "welcome to ReactJS",
            version : 4.0,
            subs : [ 'Angular',
                     'React',
                     'NodeJS',
                     'VueJS',
                     'React Native' ]
```

```
    }
};
  render() {
      return (
        <div>
          <child arg1 = {this.state.wish}
                 arg2 = {this.state.version}
                 arg3 = {this.state.subs} />
        </div>
      )
    };
};
```

Child.JS :-

```
import React, {Component} from "react";
export default class Child extends React.Component {
    render() {
        return(
            <div>
                <h1> {this.Poops.arg1} </h1>
                <h1> {this.Poops.arg2} </h1>
                <p> {this.Pops.subs.map((e,i) => (
                    <h1> {e} ... {i+1} </h1>
                ))} </p>
            </div>
        )
    }
};
```

first component to third Componat through Second Component :-

First.js :- Import React, {component} from "react";
Import Second from "./second";
export default class First extends React.component {
    State = {
        Sub: "ReactJs"
    };
    render() {
        return (

                <Second arg1 = {this.state.sub}/>

)
    };
};

Second.js :- Import React, {component} from "react";
Import Third from "./Third";
export default class second extends React.component {
    render() {
        return (

                <Third {...this.props} />

        );
    };
};

Third.JS :-

```
import React, {Component} from "react";
export default class Third extends React.Component{
    render() {
        return (
            <div>
                <h1> {this.Props.arg1} </h1>
            </div>
        )
    }
};
```

Button.JS :- Import React, {Component} from "react";

```
export default class Button extends React.Component {
    myfun = () => {
        alert ("Hello");
    };
    render(){
        return (
            <div>
                <button onClick={this.myfun}> Click me
                </button>
            </div>
        )
    };
};
```

4/12/19 :- cd events-app → yarn start

~~import React, {Component} from~~

Ex:- import React {component} from "React";
export default class Events2 extends React.component {
  render() {
    return (

        <button onClick={() => {this.myfun({'uname':'admin', 'upwd':'admin'})}}> login </button>

)
  };
  myFun = (obj) => {
    if (obj.uname === "admin" && obj.upwd === "admin")
      alert ("Login success");
    else
      alert ("Login fail");
  };

Ex:- import React {component} from "react";
export default class Events3 extends React.
                                    component.
  render() {
    return (

```
<input type="text" placeholder="username" ref="uname"></input>
<br></br>
<button onClick={this.myFun}>read</button>
</div>
)
};
myFun = () => {
alert(this.refs.uname.value);
};
};
```

Login Form :-

```
import React,{Component} from "react";
export default class Events4 extends React.Component{
render(){
return(
<div className="App">
<fieldset>
<legend>login</legend>
<input type="text" placeholder="username" ref="uname">
</input>
<br></br><br></br>
<input type="password" placeholder="password"
ref="upwd"></input>
<br></br><br></br>
<button onClick={this.login}>Login</button>
</fieldset>
```

```
    >
  3
  login=() => {
      If(this.refs.uname.value === 'admin' && this.
        refs.upwd.value === "admin")
        alert("Login Success ----!");
      else
          alert("Login fail ---!");
    }
};
```

to access the data

6/12/19 :-

To change the State : SetState().

State-change-app.

```
Import React, {Component} from "react";
export default class APP extends React.Component {
    constructor() {
        super();
        this.state = {
            sub: "ReactJs"
        };
    };
    render() {
        return(
            <div>
                <h1> {this.state.sub} </h1>
                <button onclick={this.myfun}> change </button>
            </div>
        )
    };
    myfun = () => {
        this.setState({
            sub: "React native"
        })
    };
};
```

Middleware

→ React supports the following Middlewares.

1) Redux
2) Saga
3) Thunk.

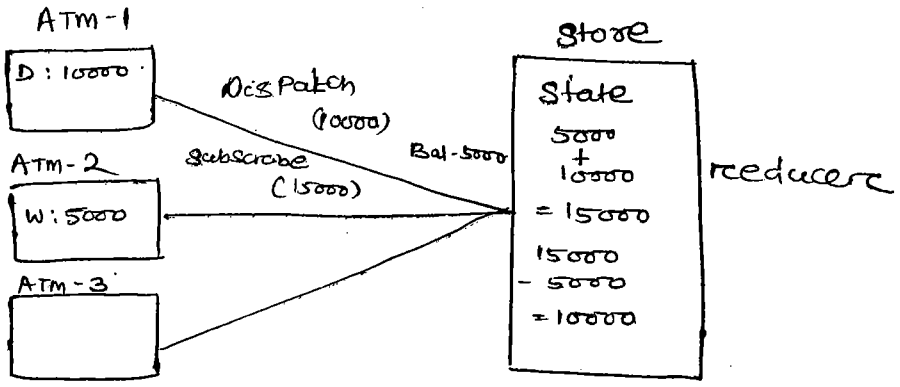
→ Middlewares used to make the "application logic" as global.

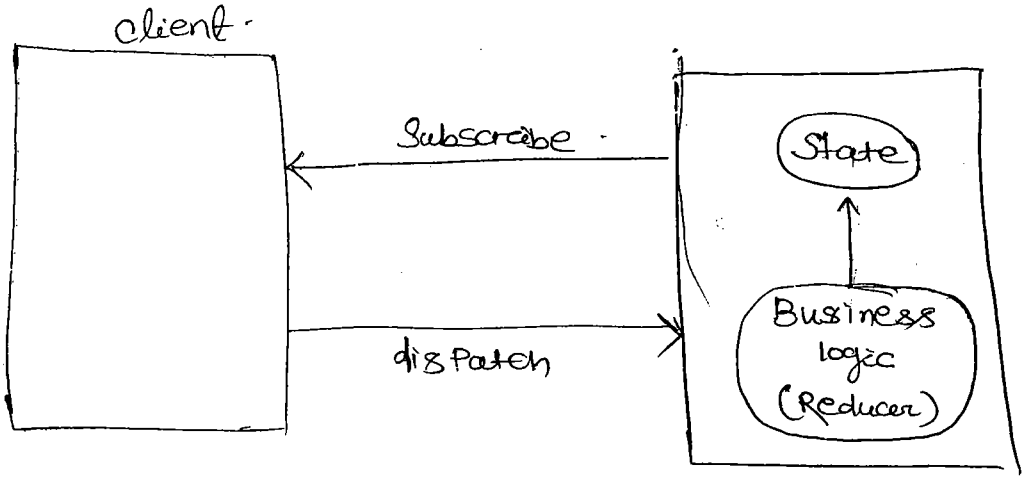
→ If we make application logic as global, everyone can access the bussiness logic.

Terminology in Middleware:

1. Store
   - Store is the common place, where we are deploying appl^n logic.
2. Reducers
   - "application logic called" as reducers.
3. State
   - reducer o/p called as state.
4. dispatch
   - requesting reducer technically called as dispatch.
5. Subscribe
   - Making the "result of reducer" (State) available to clients called as Subscribe.

→ get the result.


ATM-1
D : 10000
ATM-2
W : 5000
ATM-3
Dispatch
(10000)
Subscribe
(15000)
Bal-5000
Store
State
5000
+
10000
= 15000
15000
- 5000
= 10000
reducers


client
Subscribe
dispatch
State
Business
logic
(Reducer)

Node Introduction :- 7/12/19

→ Node JS is the server side scripting language.

→ NodeJS is used to develop the servers.

- → Http Server
- → TCP Server.

→ We will develop node applications by using Javascript.

→ We will start the node server by using either node or nodemon tools.

→ Both node and nodemon tools provided by NodeJS.

→ "Nodemon" tool used to start the server on watch mode. (live mode).

→ We will install nodemon tool by using following command.

```
> npm install -g nodemon@latest.
```

→ "npm" stands for node package manager.

→ "npm" is the integrated tool of NodeJS.

→ "-g" stands for global installation.

→ Node supports the Modules.

→ Predefined libraries called as modules.

Module in NodeJS :-

1) MySQL -
   - This module used to connect to mysql database.

2) MongoDB — Port no. - 27017
   - This module used to connect to mongoDB database.

- The Stable version of mongodb module is 2.2.32.

3) mssql

- mssql module used to connect to SQL Servers.

4) express

- express module used to develop the rest apis (web services)

→ Providing comm^n bet^n different technologies called as web services.

5) body-parser

- "body-parser" module used to read the POST parameters.

6) cors

- this module used to enable the communication between ~~cors~~ ports.

7) multer

- multer module used to upload images to the server.

8) Socket.io

- this module used to implement the chat applications.

9) jwt-simple

- this module used to generate the tokens.
- In general we can use these tokens for authentication purpose.
- This authentication system technically called as token based authentication system.

- "jwt" stands for json web token.

→ we will download above modules by using either "npm" or "yarn" tool.

→ "yarn" is the tool given by facebook.

→ "yarn" used to download the libraries in faster manner.

→ we will install "yarn" tool by using following command.

> npm install -g yarn@latest.

→ "Postman" is the software used to test the rest apis.

website :- https://www.getpostman.com/downloads/

file: Postman-win64-7.11.0-Setup.exe.

EX1:-

Develop the rest apis by using node js.

1) Create the directory (folder)

Ex.

RestAPI

2) Download the "express" module

- "express" module used to develop the restAPIs.

- we will download "express" module by using yarn tool.

> Yarn add express --save.

3) Develop the rest api by using nodeJS.

Ex

Server.JS

Server.JS :-

```
// Import the express module.
// require() is the predefined function in Nodejs, used to
   import the modules.
// require() function will return object.
   let express = require("express");

Put() - update the data in server is Put().
get() - get the data from server.
Post() - Send the data to server.

// Create the rest object by using express module.
// rest object used to create the rest apis.
   let app = express();
// "where" app" is the rest object.
// where "app" object used to create the rest api's.

// Create the get request:
   app.get("/", (req, res) => {
   // both req & res objects provided by node engine.
   // req object used to receive the data from client.
   // res object used to give the reply to client.
      res.send("mysql connection Soon ---!");
   });

   app.get("/mongodb", (req, res) => {
      res.send("mongodb connection Soon ---!");
   });
```

// create the Post request &

```
app.post("/", (req, res) => {
    res.send("sqlserver connection soon ---!");
});

app.post("/firebase", (req, res) => {
    res.send("firebase connection soon ---!");
});
```

// assign the port no

```
app.listen(8080);
console.log("server listening the Port no. 8080");
```

4) Start the server

```
>nodemon server
```

5) test the rest api's by using Postman.

⇒ http://localhost:8080/ (GET)

⇒ http://localhost:8080/mongodb (GET)

⇒ http://localhost:8080/ (POST)

⇒ http://localhost:8080/firebase (POST)

9/12/19 :- Yarn add redux --save → Download redux library

Demo.js

```
// import createstore
// createStore is the Predefined class.
// create Store class used to create the "Store" in redux architecture.
const {createstore} = require("redux");
// create initialstate
const initialState = {
    bal : 5000
};

// create the Store
const Store = createstore

// create the reducers
const my_reducer = (State = initialstate, action) => {
    // create the virtual Dom
// virtual Dom increases the appln Performance
    const newstate = {...State};
    switch (action.type) {
        case : "DEPOSIT" :
            newstate.bal += action.value;
            break;
        case : "withDRAWN":
            newstate.bal -= action.value;
            break;
    }
```

```
// ~~create~~ the return virtual DOM
    return newstate;
};

// create the store
const store = createstore(myReducer);
// subscribe
store.subscribe(() => {
    console.log(store.getstate());
});

// dispatch
store.dispatch({type:"DEPOSIT", value: 10000});
store.dispatch({type: "WITHDRAW", value: 7000});
store.dispatch({type: "DEPOSIT", value: 5000});
```

Assignment :-

1) create the state with inital age as 20.
2) Increment the age 3 times with value as 1.
3) decrement the age 3 times with value as 2.
4) Show the dispatch and subscribe actions

10/12/19

```
const {createStore} = require("redux");
const initialState = {
        age : 20;
      };
const myReducer = (state = initialState, action) => {
        const newstate = {...state};
       switch (action.type) {
         case "AGE_UP":
             newstate.age += action.value;
             break;
         case "AGE_DOWN":
             newstate.age -= action.value;
             break;
    }
     return newstate;
};
 const store = createStore(myReducer);
    store.subscribe(() => {
            console.log(store.getState());
    });

    store.dispatch({type : "AGE_UP", value: 1});
    store.dispatch({type: "AGE_UP", value: 1});
    store.dispatch({type: "AGE_UP", value: 1});
```

```
store.dispatch({type:"AGE_DOWN", value:2});
store.dispatch({type:"AGE_DOWN", value:2});
store.dispatch({type:"AGE_DOWN", value:2});
```

Integrate Redux with React :-

~~[redacted]~~

Yarn add redux react-redux --save.

→ React-redux library used to integrate the redux to react JS.

→ src → Newfolder → reducer → newfile → reducer.js.

reducer.js :-

```
// we will create reducer and we will export it.
// initial balance.
const initialState = {
    bal : 5000
};

// create the reducer
const reducer = (state = initialState, action) => {
    const newstate = {...state};
    switch(action.type){
        case "DEPOSIT":
            newstate.bal += action.value;
            break;
```

```
    case "WITHDRAW":
        newstate.bal -= action.value;
        break;
    }
    return newstate;
};

// export the reducers
export default reducers;
```

index.js

```
import React from 'react';
import ReactDom from 'react-Dom';
import './index.css';
import App from './App';

// import reducers
import reducer from "./reducer/reducer";

// import createstore
import { createStore } from "redux";

// import Provider
// Provider makes the Store as global
import { Provider } from "react-redux";
const store = createStore(reducer);
import * as serviceWorker from './serviceWorker';
ReactDom.render(<Provider store={store}><App/>
</Provider>,
```

document.getElementById('root'));
serviceWorker.unregister();

App.js :-

```
import React, { component } from "react";
import "./App.css".

// connect is the helper class
// connect class used to "use" dispatch and subscribe
   on React component.
// helper class also called as "Higher order component".

import { connect } from "react-redux";
class App extends React.component {
    render(){
        return(
            <div className ="App">
            <h1> Bal : {this.Props.bal} </h1>
    <button onClick = {this.Props.onDeposit} > Deposit </button>
<button onClick = {this.Props.onWithDraw} > withDraw
                                            </button>
        </div>
        )
  };
};
```

```
// subscribe
let receive = (state) => {
    return {
        bal: state.bal
    }
};
// dispatch
let send = (dispatch) => {
    return {
        onDeposit: () => dispatch({type: "DEPOSIT", value: 10000}),
        onWithdraw: () => dispatch({type: "WITHDRAW", value: 5000})
    }
};
export default connect(receive, send)(App);
```

11/12/19 :- React redux example :-

reducers.js :- cd agg-app.

Yarn add redux react-redux --save.

```
Const InitialState = {
    age : 20.
};

Const reducer = (State = initialstate, action) => {
    Const newstate = {... State};
    Switch (action.type) {
        Case "AGE_UP":
            newstate.age += action.value;
            break;
        Case "AGE-Down":
            newstate.age -= action.value;
            break;
    };
    return newstate;
};

export default reducer;
```

We are deploying reducer to store

index.js

```
import reducer from "./reducer/reducer";
Import { createStore } from "redux";
Import { Provider } from "react-redux";
  Const Store = Createstore(reducer);
ReactDom.render (<Provider store = {Store} ><APP/>
    </Provider>, document.getElementById('root'));
```

APP.js

Store availability into app.

```
import React, {Component} from "react";
import "./App.css";
import {connect} from "react-redux";
class App extends React.Component {
    render() {
        return (
            <div className="App">
            <h1> Age:
            <span style={{color:'red'}}>{this.props.age}</span>
            </h1>
            <br> <br>

    <button onClick={this.props.onAgeUp}><b> AgeUp </b>
                                        </button>
<button onClick={this.props.onAgeDown}><b> AgeDown </b></button>

            </div>
        )
    };
};

// Sub Scribe
const receiveData = (state) => {
    return {
        age: state.age
    }
};
```

```
// dispatch
const SendRequest = (dispatch) => {
  return {
    OnAgeUp: () => dispatch({ type: "AGE_UP", value: 1 }),
    OnAgeDown: () => {dispatch({ type: "AGE_DOWN", value: 2 })}
  }
};

// Hoc
export default connect(receiveData, SendRequest)(APP);
```

12/12/19 :-

Rest API calls :- "axios" is the 3rd party module

→ 'axios' module used to make the rest API calls.

→ we will download "axios" module by using yarn tool.

→ Yarn add axios --save.

→ automatically axios module doownloads to "node_modules" folder.

→ ComponentDidMount() is the react life cycle function, used to make the rest API calls.

→ never make the rest API calls in render()

→ URL : https://restcountries.eu/rest/v2/all.

```
rest-api-APP
    src
        APP.js
        index.js
```

```
import React, { Component } from "react";
import "./App.css";
// Import axios class
// axios used to make the rest api calls.
import axios from "axios";
export default class App extends React.Component {
    constructor() {
        super();
        this.state = {
            countries : [ ]
        };
    };
    componentDidMount() {
        axios.get("https://restcountries.eu/rest/v2/all")
            .then((posRes) => {
                this.setState({
                    countries : posRes.data
                })
            }, (errRes) => {
                console.log(errRes);
            });
    };
    render() {
        return (
            <div className="App">
            <table border="1"
                cellpadding="10px"
                cellspacing="10px"
                align="center">
```

hosted on the server

```
<thead style={{backgroundColor:"grey"}}>
  <tr>
    <th>SNo</th>
    <th>Name</th>
    <th>Capital</th>
    <th>Population</th>
    <th>flag</th>
    <th>code</th>
  </tr>
</thead>
<tbody>
{this.state.countries.map((element, index) => (
    <tr>
      <td>{index+1}</td>
      <td>{element.name}</td>
      <td>{element.capital}</td>
    <td>{element.population}</td>
    <td><img width="100px", height="50px"
        src={element.flag}></img></td>
  <td>{element.currencies[0].code}</td>
  </tr>
))}
</tbody>
</table>
</div>
);
}
```

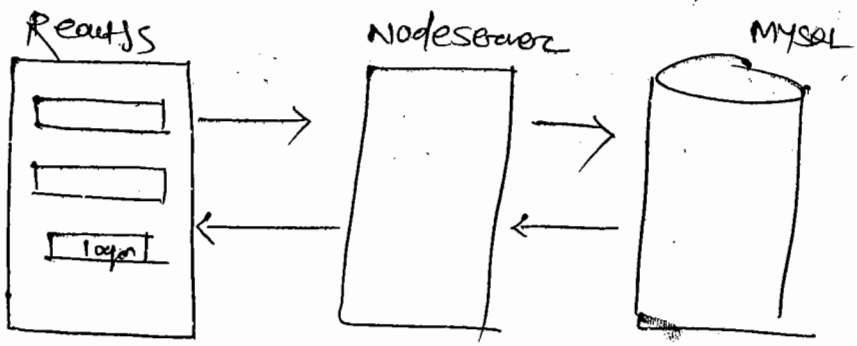
2.

Assignment :- Read the rest api data from following URLs.

1) https://www.w3schools.com/angular/customers.php
2) http://test-routes.herokuapp.com/test/hello
3) http://httpbon.org/get?param1=hello

```
const initalstate = {
        countries : [ ]
};
let reducer = (state = initialstate, action => {
      const newstate = {...state};
          switch (action.type) {
              case "COUNTRIES":
                  axios.get("URL").then((posRes)=> {
                      newstate.countries = posRes.data
                  }, (errRes) => {
              })
          }
          return newstate;
};
```

PostRequest :-



13/12/19 Login Authentication :-

1) Create the react application.

> create-react-app demo-app.

Where "demo-app" is the react application.

2) Switch to react application.

> cd demo-app

3) download the libraries

⇒ express

⇒ mysql

⇒ body-parser

⇒ cors

• – "express" module used to develop the rest apis.

• – "mysql" module used to interact with the mysql database.

• – "body-parser" module used to read the Post Parameters.

• – "cors" module used to enable the ports communication

> Yarn add express mysql body-parser cors --save

4) Develop the rest api by using nodejs

demo-app

Servere

Server.js

Server.js

```
// Import the modules
// @express @mysql @cors @body-parser
// require() is the predefined function used to import
   the modules.

let express = require("express");
let mysql = require("mysql");
let bodyParser = require("body-parser");
let cors = require("cors");

// Create the rest object.
let app = express();

// where "app" is the rest object.
// where "app" object used to develop the rest api's.

// enable the cors
→ Providing the comm" bet" reactJS & nodeJS.

    app.use(cors());

// Set the JSON as MIME TYPE.
app.use(bodyParser.json());

// read the Post parameteres.
app.use(bodyparser.urlencoded({extended: false}));

// create the connection object
let connection = mysql.createconnection({
                    host: "localhost",
                    User: "root",
                    password: "root",
```

```
        database = "testdb"
    });

// Connection to database.
Connection.connect();

// Create the rest api
app.post("/login", (req, res) => {
    // execute the SQL query.
    connection.query('select * from login_details
    where uname = '${req.body.uname}' and upwd = '${
    req.body.upwd}'', (err, records, fields) => {
        if (records.length > 0) {
            res.send({ login : "success" });
        } else {
            res.send ({ login : "fail" });
        }
    });
});
// assign the port
app.listen (8080);
Console.log ("Server listening the Port no. 8080");
```

5) Start the server

```
> cd Server
> node Server
```

6) test the following rest api by using Postman

⇒ http://localhost:8080/login (POST)

→ Post → url → Body → raw → JSON

7) download the axios Module.

> Yarn add axios --save.

8) Create the react application.

Ex. App.js

```
import react, {component} from "react";
import axios from "axios";
import "./App.css";
export default class APP extends React.Component{
        constructor(){
          super();
         this.state = {
            status :{}
          };
      }
  login =()=>{
        axios.POST ("http://localhost:8080/login",
            {"uname": this.refs.uname.value,
            "upwd": this.refs.upwd.value})
          .then ((posRes)=>{
                this.setstate({
                    status : posRes.data
              })
    }, (errRes)=>{
```

```
            console.log(errRes);
        });
    };
    render(){
        return(
            <div className="App">
            <fieldset>
                <legend>Login</legend>
    <input type="text" Placeholder="username"
                ref="uname"></input>
            <br></br>
    input type="Password" Placeholder="Password"
                ref="Upwd"></input>
        <br></br><br></br>
<button onClick={this.Login}>Login</button>
        <h1>{this.state.status.login}</h1>
    </fieldset>
</div>
    )
};
```

16.12.19

```
const initialState = {
        age : 20,
        arrr : []
};

const reducerr = (state = initialState , action) => {
    const newstate = {...state};
    switch (action.type) {
      case "AGE-UP";
          return {
              newstate,
              age:state.age + action.value,
          arrr : state.arrr.concat({id : Math.random(),
                                    age :state.age + action.value})
      }

    case "AGE-DOWN":
      return {
            ... state,
          age : state.age - action.value
          arrr: state.arrr.concat ({id:Math.random(),
           age : state.age - action.value})
      }

    case "DEL_AGE":
        return {
              ... state,
            arrr.State , arrr.filter((element, index) => {
```

```
        return element.id != action.value;
      });
    }
  }
    return newState;
};
export default reducer;
```

index.js :-

```
import Reducer from "./reducer/reducer";
import {createStore} from "redux";
import {Provider} from "react-redux";
const Store = createStore(reducer);
ReactDom.render(<Provider store={Store}><APP/>
        </Provider>, document.getElementById('root'));
```

APP.js :-

```
import React, {component} from "react";
import "./APP.css";
import {connect} from "react-redux";
class APP extends React.component {
    render() {
        return(
            <div className = "APP">
        <h1> Age :
```

```
<span style={{color:'red'}}>{this.Props.age}</span>
</h1>
<button onClick={this.Props.onAgeUp}>Age UP</button>
<button onClick={this.Props.onAgeDown}>Age Down</button>
<br><br><br>
<h1>
{this.Props.arrr.map((el)=>(
<p onClick={()=>this.Props.delAge(el.id)}>{el.age}
</p>
))}
</h1>
</div>
);
};

const receive = state => {
return {
age: state.age,
arrr: state.arrr
}
};

const send = dispatch => {
return {
onAgeUp: ()=> dispatch({type:"AGE_UP", value:1}),
onAgeDown: ()=> dispatch({type:"AGE_DOWN", value:1}),
delAge: (id)=> dispatch({type:"DEL_AGE", value:id})
```

```
    }
};
export default connect(receive, send)(APP);
```

17/12/19 :-

```
const initialState = {
    bal : 5000,
    Statement : [ ]
};
const reducerc = ( state = initialstate , action) => {
    const newstate = {... state};
    switch(action.type) {
        case "DEPOSIT" :
            return {
                newstate,
                bal
                ~~amount~~ : State.bal + action.value,
                Statement
                ~~ministatement~~ : State . statement
                    .concat({amt : State.bal + action.value})
            }
        break;
        case "withDRAW" :
            return {
                newstate,
                bal
                ~~amount~~ : State.bal - action.value
                ~~Mini~~Statement : State . Statement
                    .concat({amt : State.bal - action.value})
            }
        break;
    }
    return newstate ;
};
```

export default reducer;

Index.js

```
import reducer from "./reducer/reducer";
import {createstore} from "redux";
import {Provider} from "react-redux";
const store = createstore(reducer);
ReactDom.render(<Provider store={store}>
                    <App/>
                </Provider>
document.getElementById('root'));
```

App.js

```
import React, {component} from "react";
import "./App.css";
import {connect} from "react-redux";
class App extends React.component {
    render() {
        return(
            <div classname="App">
            <h1> Balance :
                <span style={{color: 'green'}}>
                    {this.Props.amount}
                </span>
            </h1>
            <br> </br> <br> </br>
```

```
<button onclick={this.Props.onDeposit}>
    Deposit
</button>
<button onclick={this.Props.onDeposit}>
    Deposit
</button>
<button onclick={this.Props.onWithDraw}>
    WithDraw
</button>
<br></br><br></br>
<h1>
    {this.Props.ministatement.map(el => (
        <p>{el.amt}</p>
    ))}
</h1>
</div>
)
}
};
const receive = (state) => {
    console.log(state);
return {
    amount: state.bal,
    ministatement: state.statement
}
};
const send = dispatch => {
    return {
```

onDeposit : () => dispatch ({type : "DEPOSIT", value : 5000})

onWithDraw : () => DISPATH ({type : "WITHDRAW", value : 10000})

3.

};

export default connect (receive, send) (APP)

18/12/19

Multiple Reducer :-

→ connect is the helper class.

Index.js

```
import reducerA from "./reducers/reducerA";
import reducerB from "./reducers/reducerB";
import {createStore, combineReducers} from "redux";
import {Provider} from 'react-redux';
  const obj = combineReducers({
        rA : reducerA,
        rB : reducerB
  });
  const store = createStore (obj);
```

App.js

```
import React, {component} from "react";
import "./APP.css";
import {connect} from "react-redux";
 class App extends React.component {
```

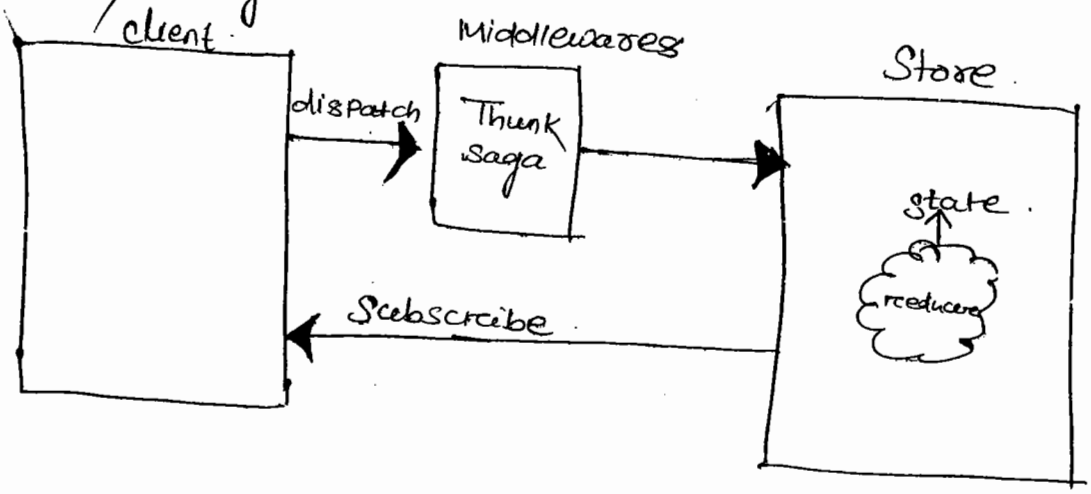
```
        render() {
          return (
            <div className = "App" >
              <div className = "col" >
            <h1> A: <Span className = "font-color" > {this.Props.a}
                                    </Span> </h1>
            <button onClick = { () => this.Props.UpdateA (this.Props.b) } >
                    UPdate A </button>
            </div>
        <div className = "col" >
          <h1> B: <Span className = "font-color" > {this.Props.b} </Span> </h1>
        <button onClick = { () => this.Props.UpdateB (this.Props.a) } > UpdateB
              </button>
          </div>
          </div>
        )
      };
    };
        const receive = (State) => {
              return {
                a: State.rcA.a,
                b: State.rcB.b
              }
            };
          const Send = (dispatch) => {
              return {
```

```
    updateA : (b) => dispatch ({ type : "UPDATE_A", value : b }),
    updateB : (a) => dispatch ({ type : "UPDATE_B", value : a })
  }
};
export default connect (receive, send) (APP);
```

Middlewares

→ redux supports the following middlewares.

1) Thunk.
2) Saga.



→ Middlewares monitors the "dispatch" from client-application.

→ We have two types of middleware as listed above, but majority of projects using "thunk".

→ Thunk is the capability to separate actions.

Ex. AGE_UP
DEPOSIT
WITHDRAW
UPDATE_A ... ...

→ In general dispatch() takes the "Json object".

→ Json object technically called as action.

Ex: {type: "AGE_UP", value: 1}

```
const initialState = {
    age: 20
};
const reducer = (state = initialState, action) => {
    const newstate = {...state};
    switch (action.type) {
    case "AGE_UP":
        newstate.age += action.value
        newstate.loading = false;
        break;
    case "AGE_DOWN":
        newstate.age -= action.value;
        break;
    case "LOADING":
        newstate.loading = true;
    }
    return newstate;
};
export default reducer;
```

actions.js

```
export const loading = () => {
    return {
        type: "LOADING"
    };
};
```

```
export const ageUpAsync = val => {
    return { type: "AGE_UP", value: val };
};

export const ageup = val => {
  return dispatch => {
    dispatch(loading());
    setTimeout(() => {
      dispatch(ageUpAsync(val));
    }, 5000);
  };
};

export const agedown = val => {
    return { type: "AGE_DOWN", value: val };
};
```

App.js :-

```
import React, { Component } from "react";
import "./App.css";
import { connect } from "react-redux";
import logo from "./logo.svg";
import * as actionCreator from "./store/actions/actions";

class App extends Component {
    render(){
      return (
        <div className = "App">
        <div className = "Age-label">
```

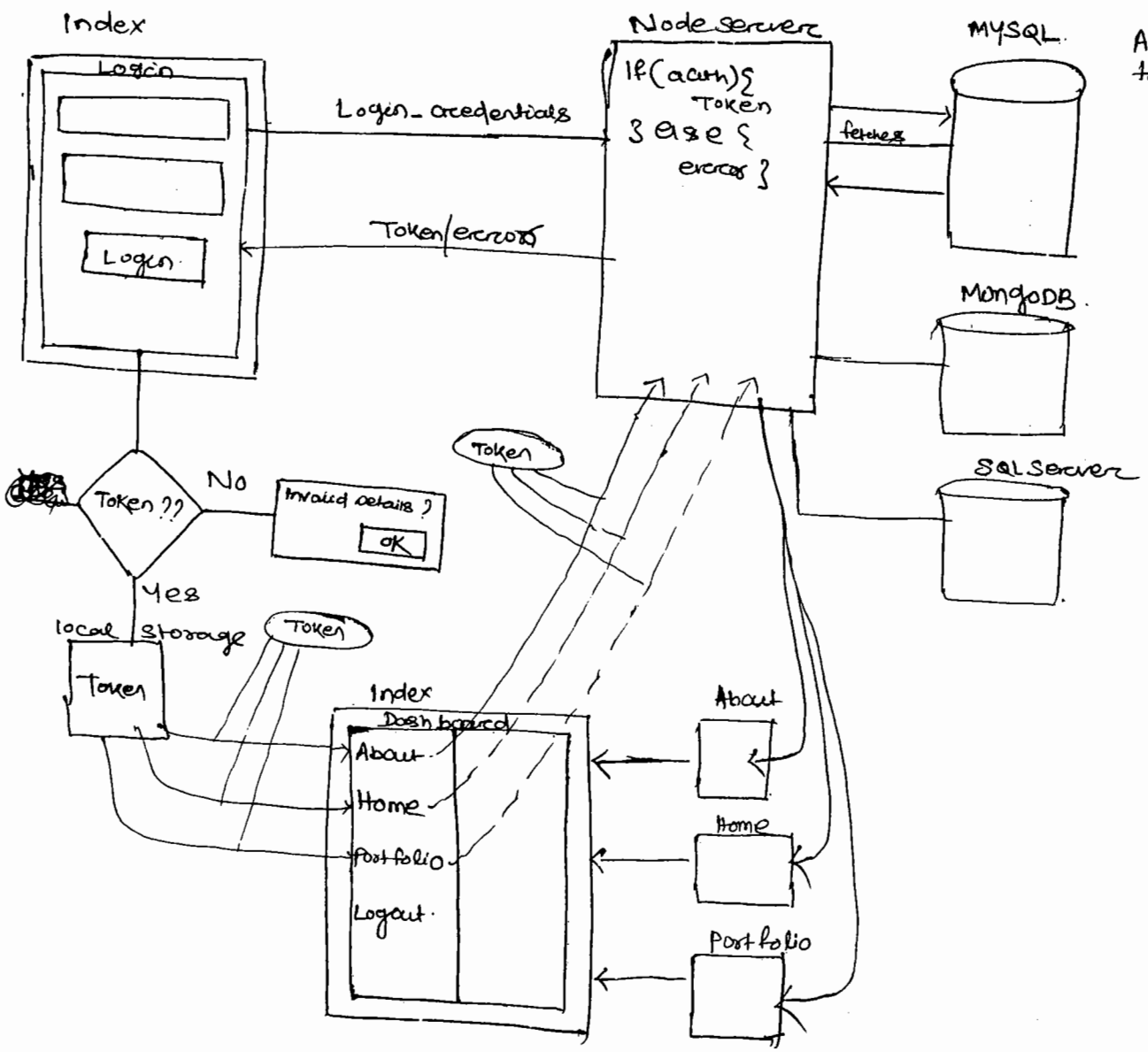
```
        Youre age : <span> {this.Props.age} </span>
      </div>
<button onClick={this.Props.onAgeup}>Age up</button>
<button onClick={this.Props.onAgeDown}>Age Down</button>
  {this.Props.loading && <img src={logo}
      className="App-logo" />}
    </div>
  );
 }
}

const receive = state => {
  return {
    age : state.age
    loading : state.loading
  };
};

const Send = dispatch => {
  return {
    onAgeup : () => dispatch(actionCreator.ageup(1)),
    onAgeDown : () => dispatch(actionCreator.ageDown(1))
  };
};

export default connect(
    receive,
    Send ) (App);
```

Assignment:

→ display the Progress bar while Performing the actions.
→ Pass the dynamic data to actions from reactJS
→ reducers should return objects instead of Primitive data.

23/12/19 :-

→ In this miniproject (PoC) we have following databases.
→ MYSQL
MongoDB
SQLServer.
→ MYSQL database for both login authentication as well as about module in miniproject.
→ MongoDB database for "home" module Presenting in Miniproject.
→ SQL Server for "Portfolio" module Present in Miniproject.
→ We will Develop rest apis by using nodeJs
→ In the above mini project we are maintaining token based authentication System.
→ Coming to the reactJS we are deviding reactJS app^n into multiple modules.
- Index
- Login
- Dashboard
@ About
@ Home
@ Portfolio



About → fetch the token from local storage.

After fetching token Send the same token to node server.

Node server compares two tokens.

If two tokens same → MySQL. MongoDB. SQLServer fetches the data from

→ In Reactjs, we are focusing on Single Page Apps.

→ Loading one component to another component without refreshing whole web project called as "Single Page Application".

Implementation :- 24|12|19

1) Download and install MYSQL Database.
mysql-essential-5.2.3-falcon-alpha-win32.msi

2) create the tables in MYSQL databases
- Default Password : root

> create schema reactjs;
- Automatically "Reactjs" database will ready.

> use reactjs;
- We can switch to react JS database.

> create table login_details(uname varchar(20), Upwd varchar(20));
- Automatically login_details table will be created.

> insert into login_details values ("admin", "admin");
- We can insert the record into login_details table

> create table about (Sno integer, Name varchar(20);
- We can create the "about" table.

> insert into about values (1, "MERN Stack");
- We can insert the record into about table.

> Select * from login_details;
Select * from about;

- we can fetch the data from tables

host : localhost

user : root

Password : root

database : reactjs

tables : Login_details

3) Make the MongoDB database ready for mini project implementation.

> mongod
- Automatically mongo Server will start.

> mongo
- we can connect to mongo Server

> use reactjs;
- Automatically reactjs database will be created. and switches also.

> db.createCollection("home");
- Automatically "home" Collection will be created (table)

> db.home.insert({Sno:1, name:"reactjs"});
- we can insert the record into home collection.

> db.home.find();
- we can fetch the data from "home" collection

Protocol : mongodb

Port no : 27017

host : Local host

database : reactjs

collection : home

4) Make the SQL Server ready for mini Project Implementation.

Name : SQL server 2014 management studio.

User : Sa

Password : 123

Server : Localhost

database : reactjs

table : portfolio

5) create the react Application

> create-react-app miniproject-app.

6) Switch to react application.

> cd miniProject-app.

7) download the node modules.

=> express

=> MySql

=> Mongodb @2.2.32

=> Mssql.

=> body parser

=> cors

=> jwt-simple.

- "express" module used to develop the rest API.
- "mysql" used to connect to mysql database.
- "Mongodb@2.2.32" used to connect to mongodb database.
- "mssql" module used to interact with the SQL server.
- "body-parser" module used to read the Post parameters -
- "cors" module used to enable the ports communication.
- "jwt-simple" module used to generate the tokens.

> Yarn add express mysql mongodb@2.2.32 mssql body-parser cors jwt-simple --save.

8) Develop the rest apis for mini Project by using nodejs.

Miniproject-app

Server

Config

To maintain mysql database properties to create connection object ← mysql_properties.js

← mysql_connection.js

To maintain sql server properties ← mssql_properties.js

To maintain the server side token. ← token.js

generateToken.js

Compare the client side token to server side token ← auth.js

Login

login.js → To compare reactJS details.

about

about.js → fetch the data from about table to mysql database.

home

home.js → fetch the data from home collection to mongodb database.

Portfolio

Portfolio.js → fetch the data from portfolio table to Sql server database.

logout

logout.js → To delete the token from token.js file.

Server.js

→ "mysql_properties.js" file used to maintain the mysql database details.

→ "Mysql_connection.js" file used to create the mysql connection object.

→ MS.sql_~~Connection~~ Properties.js" file used to maintain the SqlServer details.

→ token.js file used to ~~no~~ save the Server ~~side~~ token.

→ 'generateToken.js' file used to generate the token by using Jwt-Simple module.

→ ~~Auth~~ "auth.js" file used to compare the client side token with server side token.

- "login.js" file used to compare the react application login credentials with login_details table present in mysql database.
- "about.js" file used to fetch the data from About table present in MYSQL database.
- "Home.js" file used to fetch the data from home collection present in mongodb database.
- "Portfolio.js" file used to fetch the data from Portfolio table present in SQL Server.
- "logout.js" file used to delete the server side token.
- "Server.js" file is the main server file

25/12/19 :

mysql_properties.js

```
// This file used to maintain the mysql database properties.
// We will create and export Json object.
//
    let obj = {
        host : "localhost",
        User : "root",
        Password : "root",
        database : "reactjs"
    };
    module.exports = obj;
```

my_sql_connection.js

```
// Import mysql module.
    let mysql = require("mysql");
// Import db details.
    let obj = require("./mysql_properties");
//  Create and export Json object
    let conn = {
        getConnection : () => {
            return mysql.createConnection(obj);
        }
    };
//  export conn
    module.exports = conn;
```

mysql_properties.js :-

```
// this file used to maintain the SQL Server Properties
    let obj = {
        server : "localhost",
        user : "sa",
        password : "123",
        database : "reactJS"
    };
    module.exports = obj;
```

token.js :-

```
// This file used to maintain the server side token.
    let obj = {
            token : ""
        };
    module.exports = obj;
```

generateToken.js :-

```
// This file used to generate the token by using
   jwt-simple module.
// import jwt-simple module.
    let jwt = require("jwt-simple");
// converting readable data to unreadable data with
    custom password called as token.
        let my_fun = (data, password) => {
            return jwt.encode(data, password);
        };
```

```
module.exports = my_fun;
```

auth.js

```
// This file used to compare the client-side token with
   Server side token.
// Client Sending the token through headers.
// Serverside token Presenting ".token.js" file.
// comparing tokens before processing requests called
   as "middleware."
//
   let auth = (req, res, next) => {
     let allHeaders = req.headers;
     let c_token = allHeaders.token;
     let s.token = require(".|token").token;
     if (c_token === s_token) {
         next();
     } else {
         res.send("unauthorized user ... !");
     }
   };
   module.exports = auth;
```

login.js :-

```
// This file used to create the module.
// This module used to create the login rest api.
// login rest api used to compare the reactJS application
   "login credentials" with "login_details" table present in
   mysql database.
// get the connection object.
   let conn = require("../config/mysql_connection");
   let connection = conn.getConnection();
// import token.js file used to store the token
   let obj = require("../config/token");
// import generateToken.js file, used to generate token based
   on sucessful authentication.
      let my_fun = require("../config/generateToken");
// create and export the module
   module.exports = require("express").Router().post
      ("/", (req, res) => {
   connection.query('select * from login_details where uname=
      ${req.body.uname}' and upwd = '${req.body.upwd}',
                                    (err, records, fields) => {
      if (records.length) {
         let token = my_fun({
            uname: req.body.uname,
            upwd: req.body.upwd
```

```
        }, "hit@tcs.in");
        obj.token = token;
        res.send({login:"success", token:token});
    }else{
        res.send({login:"fail"});
    }
    });
});
```

about.js :-

```
let conn = require("../config/mysql_connection");
let connection = conn.getConnection();
let auth = require("../config/auth");
module.exports = require("express").Router().get
("/", [auth], (req,res) => {
connection.query('select * from about', (err,records,fields)=>{
    res.send(records);
});
});
```

26/12/19

Home.js :-

```
// This file used to fetch the data from home collection,
   present in ReactJS database.

let mongodb = require("mongodb");
let auth = require("../config/auth");
module.exports = require("express").Router().get("/",
    [auth], (req, res) => {
  let narcesh1T = mongodb.MongoClient;
narcesh1T.connect("mongodb://localhost:27017/reactjs",
        (err, db) => {
    db.collection("home").find().toArray((err, array) => {
        res.send(array);
    });
  });
});
```

Portfolio.js :-

```
// This file used to fetch the data from Portfolio
   table present in ReactJS database in SQL Server.

let mssql = require("mssql");
let obj = require("../config/mssql_properties");
let auth = require("../config/auth");
module.exports = require("express").Router()
    .get("/", [auth], (req, res) => {
      mssql.connect(obj, (err) => {
```

```
    if(error) {
        console.log("error....!");
    } else {
        let request = new mssql.Request();
        request.query('select * from Portfolio', (err, records) => {
            if (err) {
                console.log("error...!");
            }
            else
                res.send(records);
            mssql.close();
        });
    }
});
});
```

logout.JS :- This file used to delete the server side token.

```
let obj = require("../config/token");
let auth = require("../config/auth");
module.exports = require("express").Router().get("/", [auth],
    (req, res) => {
        obj.token = "";
        res.send({logout : "success"});
});
```

Server.JS :- Node starts the execution from server.JS file.

```
let express = require("express");
let bodyParser = require("body-parser");
let cors = require('cors');
let app = express();
app.use(bodyParser.json());
```

```
app.use(bodyParser.urlencoded({extended: false}));
app.use(cors());
app.use("/logen", require("./logen/logen"));
app.use("/about", require("./about/about"));
app.use("/home", require("./home/home"));
app.use("/Portfolio", require("./Portfolio/Portfolio"));
app.use("/logout", require("./logout/logout"));
app.listen(8080);
console.log("Server listening the port no. 8080");
```

9) Start the node server.

```
> cd MiniProject.app
> cd server
> node server.
```

automatically node server will start on Port no 8080.

10) Test the following Rest apis by using Postman.

⇒ http://localhost:8080/logen (POST)
⇒ http://localhost:8080/about (GET)
⇒ http://localhost:8080/home (GET)
⇒ http://localhost:8080/Portfolio (GET)
⇒ http://localhost:8080/logout (GET)

11) Download the axios Module

→ "axios" module used to make the rest api calls.

```
> cd miniproject-app
> npm add axios --save.
```

12) Implement the components with the Rest Api calls.

Login.JS

```
import React, {component} from "react";
import "./App.css";
import axios from "axios";
export default class logen extends React.component {
  render() {
    return (
      <div classname = "App">
       <fieldset>
         <legend> logen </legend>
         <label> uname </label>
        <input type = "text"
            ref = "uname"
          Placeholder = "user name"> </input>
        <br> <br> <br> <br>
<label style = {{ marginRight: 40}}> upwd: </label>
 <input type = "password"
      ref = "upwd"
     Placeholder = "Password"> </input>
   <br> <br> <br> </br>
  <button onclick = {this.logen}> logen </button>
  </fieldset>
 </div>
 )
}
logen = () => {
   axios.post ("http://localhost:8080/login",
       {
          'uname': this.ref.uname.value,
          'upwd': this.ref.upwd.value
       }).then ((Response) => {
```

```
    let str = JSON.stringfy (PostPos.data);
    window.localstorage.setItem ("login_details", str);
// navigate to Dashboard.



3, (errRes) => {
        console.log (errRes);
    });
  };
};
```

About.js : import React{Component} from 'react';
import axios from [redacted] "axios";

```
export default class About extends React.Component {
    constructor () {
      super();
       this.state = {
          res : "Please wait ... !"
        };
    }
    render () {
      return (
        <div classname = "App">
        <h1 style ={{ color : "red" }} > {this.state.res} </h1>
        </div>
      );
    };
```

```
componentDidMount(){
  let str = window.localStorage.getItem("login_details");
  let obj = JSON.parse(str);
  axios.get("http://localhost:8080/about",{token:obj.token})
  .then((posRes)=>{
    let str = JSON.stringify(posRes.data);
      this.setState({
          res:str
      })
  },(errRes)=>{
        console.log(errRes);
  });
};
```

home.js :-

```
import React {component} from "react";
import axios from "axios";
export default class home extends react.component{
    constructor(){
      super();
    this.state={
        res:"Please wait...!"
    }
};
render(){
  return(
      <div className="App">
  <h1 style={{color:"green"}}>{this.state.res}</h1>
  </div>
  )
};
```

```
componentDidMount() {
    ~~axios.get("http://localhost:8080/home",{token:~~
let token =
    JSON.parse(window.localStorage.getItem("login_details")).
            'token';
  axios.get("http://localhost:8080/home",{token:token}).
    then((posRes)=>{
        this.setState({
          res:JSON.stringfy(posRes.data)
      });
    },(errRes)=>{
          console.log(errRes);
    });
  });
```

Porttfolio.js :-

```
import React{Component} from 'react';
import 'axios' from 'axios';
export default class home extends react.component {
      constructor(){
        super();
    this.State = {
          res: "Please wait...!"
        }
    };
```

20/12/19

Functional Component :- App.js

```
import React, {Component} from "react";
import user from "./user";
export default class App extends React.Component {
    render() {
        return (
            <div>
                <user key1 = "Hello"
                    key2 = {100}/>
            </div>
        );
    }
```

user.js

```
import React from "react";
let user = {Props} => {
    return (<div>
            <h1> {Props.key1} </h1>
            <h1> {Props.key2} </h1>
        </div>);
    };
    export default user;
```

Purejs

```
import React, { Component } from "react";

let Func = (props) => {
    console.log("in Func");
    return (
        <h1> {props.key1} </h1>
    );
};

export default class Purec extends React.Component {
    constructor(){
        super();
        this.state = {
            key1 : 1
        }
    };
    render() {
        console.log("Purec");
        return (
            <div>
                <Func key1 = {this.state.key1}></Func>
            </div>
        )
    }
    componentDidMount() {
        setInterval(() => {
```

```
this.setstate({
    key1: 1
})
}, 1000);
};
};
```

// Hoc
// forms with validation
// refs & keys
// component life cycle hooks
// Saga
// Dom

31.12.19

Hoc :- Reusing the business logic by using component called as ~~called as~~ in React JS called as higher order component.

1.1.2020

→ React content following stages in component life cycle

1) Initialization
2) Mounting
3) updation
4) Unmounting

Component life cycle hooks :-

Constructor :- will execute only once. In general we will use constructor for initialization

→ In general we must define State and Props in constructor.

→ The best Place to define State and Props is constructor.

Component will Mount() :-

→ This life cycle hook will execute immediately after constructor

→ This life cycle hooks will execute only once.

→ This life cycle hook is recommended for change in the State.

Render() :-

→ render function will execute immediately after component will mount.

→ In general we will use render function for UI designers.

→ When ever change detected in state or Props automatically render function will be executed.

→ Never change the State in render function

Component Did mount ( ) :-

If render function executed successfully then immediately component did mount function will execute.

→ In general we will use this life cycle hook for making network calls.

Component will ~~R~~eceive Props :-

→ If component receives data ~~then~~ through Props then, this. lifecycle hook will be executed.

→ Should component update :-

→ This lifecycle hook is the boolean lifecycle hook.

→ Based on boolean value Component may or may not receive Props.

Component will update :-

→ This life cycle hook will execute after Should Component update.

→ This lifecycle hook intimates the changes in data to render function.

componentDidUpdate :-

→ If render function executed successfully with latest data then component did update will execute

componentWillUnmount :-

→ Before killing the component by react library automatically component will unmount will execute.

Saga Middleware 2/1/2020

→ saga middleware maintains the request show on a queue mechanism.

→ Saga won't execute all request at a time.

→ Saga handles the request by request.

→ Because of saga middleware we won't miss data accuracy.

→ Saga middleware working depending on generators

→ generators hv the capability to produce values constantly

→ Generator functions should Prefix with the Star.

→ app.js :-

```
<div className = "Age-label">
Yourage : <span> {this.Props.age} </span>
</div>
<button onClick = {this.Props.onAgeup} Age UP </button>
<button onClick = {this.Props.onAgedown} > Age Down </button>
</div>
);
}
}

const mapstatePoops = State => {
    return {
        age : State.age
    };
};
```

```
const mapDispatchToProps = dispatch => {
  return {
    onAgeUp: () => dispatch({type: 'AGE_UP', value: 1}),
    onAgeDown: () => dispatch({type: 'AGE_DOWN', value: 1})
  };
};

export default connect(
    mapStateProps,
    mapDispatchProps
)(APP);
```

Saga.JS :-

```
import {delay} from "redux-saga";
import {takeLatest, Put} from "redux-saga/effects";

Function* ageUpAsync(){
  Yeild delay(4000);
  Yeild Put({type: "AGE_UP_ASYNC", value: 1});
}
export function* watchAgeUp(){
    Yeild takeLatest("AGE_UP", ageUpAsync);
}
```

```
Const initialState = {
    age: 20
};

Const reducer = ( State = initialState , action) => {
  Const newState = {... State};
 Switch (action.type) {
   Case "AGE_UP_ASYNC":
     newState.age += action.value;
     break;
 Case "AGE_DOWN":
     newstate.age = action.value;
   break;
 }
    return newState;
};
  export default reducer;
```

Index.JS :-

```
// Import reducer
 Import reducer from "./store/reducer";
// Import Provider
 Import {Provider} from "react-redux";
 Import {createstore, applyMiddleware} from "redux";
 Import createSagaMiddleware from "redux-saga";
 Import {watchUp} from "./sagas/saga";
 const store = createstore(reducer, applyMiddleware(
      SagaMiddleware));
```

Sagemiddleware.run(waterAge(p));

1. Difference bet^{n} Real DOM and Virtual DOM.

Real DOM	Virtual DOM
1) It updates slow.	It updates faster.
2) Can directly update HTML	Can't directly update HTML.
3) DOM Manipulation is very expensive.	DOM Manipulation is very easy.
4) Too much of memory wastage.	No memory wastage.

2) What is React ?

→ React is a frontend ~~application~~ JavaScript library developed by facbook in 2011.

→ It follows the component based approach which helps in building reusable UI components.

→ It is used for developing complex and interactive web and Mobile UI.

→ It was open-sourced only in 2015.

3) What are the features of react ?

i) It uses the Virtual Dom instead of the Real Dom.

ii) It uses server-side rendering.

iii) It follows uni-directional data flow or data binding

4) List some of the mayors advantages of React.

i) It increases the appl^{n} Performance.

ii) It can be used on client as well as server side.

iii) Because of JSX code readbility increases.

iv) React is very easy to integrate with other frameworks like Meteor, Angular etc.

v) Using React, writing UI test cases became extremely easy.

5) What are the limitations of React ?

i) React is just a library, not a full-blown framework.

ii) Its library is very large and takes time to understand.

iii) Its can be little diff for the novice programmers to understand.

iv) coding gets complex as it uses inline templating and JSX.

6) What is JSX ?

JSX Stand for Javascript XML. This is a type of file used by React which utilizes the expressiveness of JS along with HTML like Template syntax. This makes the HTML file really easy to understand. This file makes appln robust and boost its performance.

ex:

```
render () {
    return (
        <div>
            <h1> Welcome to React </h1>
        </div>
    );
}
```

7) What do you understand by virtual Dom ? ✓

Virtual Dom is a lightweight JS 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 object and their properties. React's render function creates a node tree out of the react components.

Virtual Dom works three simple steps.

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

2) Then the diff ~~bet~~ of the ~~Previous~~ 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.
   → Virtual Dom can't directly update HTML.
   → Dom manipulation is very easy.
   → No memory wastage

8) Why can't browser read JSX ? ✓

Browser can read only Javascript objects. but JSX is not a regular JS object. Thus to enable a browser to read JSX, first we need to transform JSX file into a JS object using JSX transformers like babel and then pass it to the browser. <01

9) How diff is React's ES6 syntax when compared to ES5. ✓

1) require vs import

```
// ES5
var React = require('react');

// ES6
import React from 'react';
```

ii) exports vs exports

// ES5 - Module.exports = component;

// ES6 - export default component;

iii) Component and function

// ES5

```
var MyComponent = React.createClass({
    render: function() {
        return (
            <h3> Hello </h3>
        );
    }
});
```

// ES6

```
class MyComponent extend React.component {
    render() {
        return (
            <h3> Hello </h3>
        );
    }
}
```

10) How is React diff from Angular ?

React	Angular
→ only the view of MVC.	→ Complete MVC.
→ Server side rendering	→ client side rendering
→ uses virtual Dom	→ uses real Dom
→ one way data binding	→ Two way data binding
→ compile time debugging	→ Runtime debugging.
→ facebook	→ Google.

11) What do you understand from "In React, everything is a component.". ✓

Components are the building blocks of a React app's UI.

→ These components split up the entire UI into small independent and reusable pieces.

→ Then it renders each of these components independent of each other without affecting the rest of the UI.

12) Explain the purpose of render() in React. ✓

Each React component must have a render() mandatorily.

→ It returns a single React element which is the representation of native DOM component.

→ If more than one HTML element needs to be rendered they must grouped together inside tag such as <form>, <group> .

→ This function must be kept pure i.e it must return the same result each time it is invoked.

13) How can you embed two or more components into one ?

14) What is Props ? ✓

Props is the shorthand for Properties in React.

→ They are read-only components which must be kept pure i.e immutable. (can't change the value of Props)

→ component receives data from another component by using "props".

→ It is a Predefined functionality used to receive data from anoter component.

→ They are always Passed down from the Parent to the child components throughout the app's.

→ A child component can never send a Prop back to the Parent component.

15) What is State in React and how is it used ? ✓

→ State is the heart of React components.

→ States are the objects which determine components rendering and behaviour.

→ They are mutable unlike the Props and ~~state~~ create dynamic and interactive components.

→ We will define State in two Places.

i) Inside constructor

ii) outside "

→ We will recommended to use inside the constructor.

→ To change the state by using setState().

16) How can you update the State of a Component?

State of a component can be updated using this.setState().

~~17) Define State and Props ?~~

→ It is used to store the component related data.

~~18) How can you update the state of a component ?~~

18) What are the diff phases of React Components life cycle ? ✓

There are three diff phases of React Component's life cycle

i) Initial Rendering Phase :- This is the phase when the component is about to start its life journey and make its way to the Dom.

ii) Updating Phase :- once the component gets added to the Dom. It can potentially update and re-render only when a Prop or State change occurs. This happens only in this phase.

iii) Unmounting Phase :- This is the final phase of a component's life cycle in which the component is destroyed and removed from the Dom

19) What is an event in React ?

In React, events are the triggered reactions to specific actions like mouse hover, mouse click, key Press etc.

There are some Syntactical diff

→ Events are named using camel case instead of just using the lowercase.

→ Events are passed as functions instead of Strings.

Each Event type contains its own Properties and behaviours which can be accessed via its event handler only.

20) What is React life cycle ?

React is going through a cycle of birth, growth and death.

Mounting - Birth of your component.

Update - Growth of your component.

Unmount - Death of your component.

render() method is the most used lifecycle method.

render is the only required when a class component in React.

→ render happens during the mounting and updating of your component.

→ React requires that your render() is Pure.

Pure func are those that do not have any side effects and will always return the same o/p when the same i/p are passed.

→ This means you can not ~~create~~ setState() within a render().

→ You can't modify the component state when the render().

21) What is arrow function in React ? How is it used ? ✓

Arrow functions are more of brief syntax for writing the function expressions . They are also called "fat arrow (⇒) the functions . Arrow functions are mostly useful while working with the higher order functions.

```
// General Way
render() {
    return (
        <MyInput onChange={this.handleChange.bind(this)} />
    );
}

// With Arrow function
render() {
    return (
        <myInput onChange={(e) => this.handleonChange(e)} />
    );
}
```

22) How do you create an event in React ? ✓

```
class Display extends React.Component({
show(evt) {
        // code
},
render() {
    // render the div with an onClick prop (value of a function)
    return (
<div onClick={this.show}> click me ! </div>
);
```

23) What do you understand by refs in React? ✓

Refs is the short hand for References in React.

→ It is an attribute which helps to store a reference to a particular React element or component which will be returned by the components render configuration function.

→ It is used to return references to a particular element or component returned by render().

24) List of the cases when you should use Refs? ✓

→ When you need to manage focus, select text or media playback.

→ To trigger imperative animations.

→ Integrate with third-party DOM libraries.

25) controlled and uncontrolled components? ✓

controlled components	uncontrolled components
→ They do not maintain their own state.	→ They maintain their own state.
→ Data is controlled by the parent component.	→ Data is controlled by the Dom.
→ They take in the current values through Props and then notify the changes via callbacks.	→ Refs are used to get their current values.

26) What are Heigher order components ? ✓

Hoc is an advanced way of reusing the component logic.

→ Hoc are custom components which wraps another component within it.

→ They can accept any dynamically provided child component but they won't modify or copy any behaviors from their c/p components.

→ Hoc are 'Pure' components.

27) What were the major problems with MVC framework ? ✓

→ Dom manipulation was very expensive.

→ Appl^n were slow and ineffecient.

→ There was huge memory wastage.

→ Because of circular dependencies, a complicated model was created around models and views.

28) What is Redux ? ✓

Redux is one of the hottest libraries for front-end development in todays marketplace.

→ It is a predictable state container for JS appl^n and is used for the entire appl^n state management.

→ Appl^n developed with Redux are easy to test and can run in diff environments showing consistent behaviour.

→ It helps you manage the data you display and how you respond to user actions.

29) What are the three Principles that Redux follows ? ✓

i) Single Source of truth : The state of the entire appn is stored in an object/state tree within a single store. The single state tree makes it easier to keep track of changes over time and debug or inspect the appn.

ii) State is read-only :- The only way to change the state is to trigger an action. An action is a plain js object describing the change. Just like state is the minimal representation of data, the action is minimal representation of the change to that data.

iii) Changes are made with pure functions

Pure functions are those whose return value depends solely on the values of their arguments.

30) List down the components of Redux. ✓

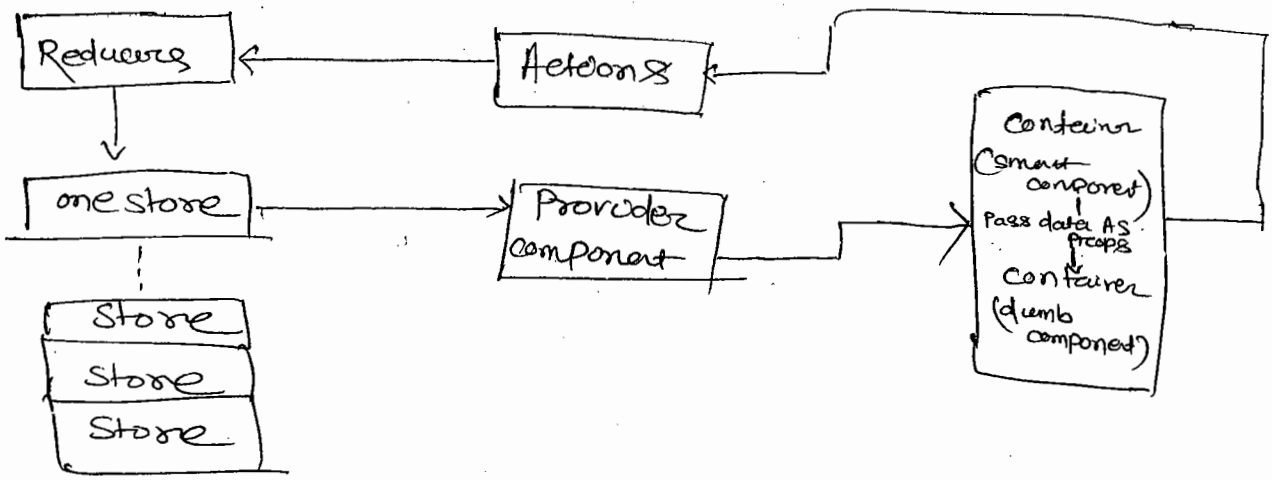
i) Action :- It is an object that describes what happened.

ii) Reducer :- It is a place to determine how the state will change.

iii) Store :- State/object tree of the entire appn is saved in the store.

iv) View :- Simply displays the data provided by the store.

31) Show how the data flows through Redux?



32) Explain the role of Reducer? ✓

→ The reducer is a Pure function that takes the previous state and an action, and returns the next state. It is the type of function you would pass to array

new state = (initial state, action)

→ It "reduce" a collection of actions and an initial state (of the store) on which to perform these actions to get the resulting final state.

33) What is Store? ✓

A Store is basically just a Plain JS object that allows components to share state.

→ A Store holds the whole state tree of your app?

→ The only way to change the state inside it is to dispatch an action on it.

34) What is dispatch? ✓

dispatch is a function of the Redux store.

→ dispatch() is the method used to dispatch actions and trigger state changes to the store.

→ You call Store.dispatch to dispatch an action.
→ This is the only way to trigger a state change.
→ By default a connected component receives a Props, dispatch and can dispatch action of self.

35) What is Subscribe ? ✓

It is a function of redux store.
→ every time the store changes.
→ So it watches for changes and then tells react to redraw.
→ Subscribe() just lets you know every time the store changes.

36) Explain State Management in React js ? ✓

→ State is a interface bet? your ~~data~~ data and react.
→ React Projects break data up into two catagories:
- State : read-write data that lives within a component.
- Props : read-only data sent to a component. may be updated later by the sending component.

37) What is middleware in ReactJS ?

→ middleware is software that lies bet^n an operating system and the appl^n running on it.

→ In ReactJS the middleware sits in between the dispatch and reducers.

→ We can alter ocur dispatched actions before they get to the reducers or execute some code during the dispatch.

38) What is connect in Redux ?

→ This function connects a react component to redux store.

→ connect is used to interact with the store,

→ we can dispatch and subscribe the data by connect() function.

39) How to make store available globally ?

* By using Provider in index.js we can make store available globally.

The <Provider /> makes the redux store globally.

→ ReactDom.render(<Provider store = {store}><APP/></Provider>, document.getElementById('root'));

40) What is Flux ?

Flux is a JS architecture or Pattern for UI which runs on a unidirectional data flow and has a centralized dispatcher.

→ It was created by Facebook, and complements Reacts as view. flux is a Pattern and Redux is a library.

→ In Flux, an action is a simple JS object, and that's the default case in Redux too, but when using Redux middleware, actions can also be functions and promises.

→ Flux is a pattern for managing how data flows through a React app.

→ It is a method of working with React components is through passing data from one parent component to its children components.

Q1) Is render mandatory in ReactJS ?

Yes render() function is mandatory in ReactJS.

→ The render() method is the only required method in a class component.

→ The render() function should be pure.

→ Render() function does not modify component state.

→ Render() function returns the same result each time it's invoked.

→ render() function does nt directly interact with the browser.

42) Stateful components

→ Are always class components.

→ Stateful components have a state that gets initialised in the constructor

→ Stateful component can render both props and state.

→ Stateful components the props and state are rendered like {this.props.name} and {this.props.state}

→ Stateful render depends upon the value

Stateless components

→ We can use either function or class.

→ Stateless components are those components which don't hv any state at all.

→ A stateless components can render. Props.

→ In stateless components the props are displayed like {props.name}

43) explain "axios" module in ReactJS ?

Axios is a Promise based HTTP client that works both in the browsers and in a node.js.

→ Axios is a lightweight HTTP client based similare to a fetch API.

→ Axios is Promise-based ashyn/await library for the readable asynchronous code.

→

49

Q) State

→ State is managed within the component.

→ State is mutable (can change) The value of state properties can be change by using setState

→ Variables declared inside the function body.

→ In class component State can be accessed using this keyword.

→ In functional component State can be used "useState" hook.

Props

→ Props get passed through Parent to child component.

→ Props are immutable.

→ Props are function Parameters.

→ In class component Props can be accessed using "this.Props".

→ In functional component Props can be accessed by using "Props".

Q) What is React ?

→ React is the client side JS library.

→ ReactJS given by facebook

→ ReactJS is simplifies the complex UI.

→ By using reactJS, we can split the complex UI to multiple executable soln.

→ Each executable soln called as component

→ components are building blocks of React App

→ ReactJS is a component based library.

→ we can reuse component in ReactJS.

→ ReactJS is a one way data binding.

→ ReactJS is the View in MVC.

front-end app? → Jo user ko samne dekhayi deta hai

→ ReactJs allows us to create reusable UI components.

→ components → building block of any react app.

45) Will react follows MVC?

No, React won't follows MVC Because it follows unidirectional data flow where ever MVC follows two-directional data flow.

46) In how many ways we can create Component?

Two ways

1) Class component.

```
Ex: import React {Component} from "react";
    Class App extends React.Component {
      Render() {
        return (
            <div>
              <b> Hello </b>
            </div>
        )
      }
      export default App;
```

2) Function Component

```
import React from "react";
const App = () => {
  return
      <h1> Hello </h1>
  }
    export default App;
```

47) What is thunk ?

→ Thunk is a middleware library in React.
→ It supports for asynchronous data flow.
→ It is a one way communication and the flow of comm^n is one way only.
→ Thunk created by using "Promises" functions.
→ It allows to write action creators that return a function instead of an action.

48) What is Saga ?

Sagas created by using generator functions.

48) Diff bet^n thunk and saga ?

→ Thunk and Saga are middleware of redux.
→ Without middleware redux store supports only Synchronous data flow.
→ Sagas and thunk responsible for asynchronous, especially for Ajax calls.

Saga	Thunk
→ Sagas created by using "generator" function.	→ Thunk created by using "Promises" function.
→ Sagas takes advantage of the yield keyword to halt execution with in a function.	→ Thunk is a function that already has everything it needs to execute.
→ Easy to test.	→ Diff to test
→ It is useful to express complex appl^n logic.	→ It is good for small use cases and for beginners.

49) how to Provide the Comm^n bet^n reducers ?
how to combine reducers ?

CombineReducers useful for Provide Comm^n bet^n reducers.

Ex:- If we are taken 2 reducers,

i) reducerA ii) reducerB.

Combine two reducer in index.js like;

```
Const rootReducer = CombineReducers({reducerA, reducerB});
```

50) What is combineReducer ?

→ It is a helper function. It is useful for combine reducers in various Places to create root reducer

→ This func^n helps you organize your reducers to manage their own slices of state.

→ In redux, there is just one store, but combine Reducers helps you keep the same logic bet^n reducers.

51) Explain Yarn tool importance in ReactJS ?

Yarn tool is the Native tool given by facebook.

→ Yarn tool used to download the libraries in faster manner in React.

→ Yarn is very fast when installing dependencies in project and NPM installs Packages sequentially.

→ Slow down the Performance significantly Yarn solves this problem by installing these Packages in Parallel.

52) How to create React App ?

1) Download and install nodeJS.
2) Download and install Git.
3) Install Yarn tool by following command :
> npm install -g yarn@latest.
4) Install "create-react-app" tool by following command.
> npm install -g create-react-app@latest.
5) Create the React App by following command.
> create-react-APP filename.
6) Switch to React app by following command.
> cd filename
7) Execute the React app following command.
> npm start (or) Yarn start.

53) What is create-react-app ?

"create-react-app" is the tool given by facebook developers. and this tool used to create the React app and it saves you from the time consuming setup configuration.

54) What is super() ?

→ "super" keyword is used as a function which calls the parent class.

→ The super() in JS is used to call the methods of the parent class.

→ It is used within a constructor function to call the parent constructor function.

Sri Raghavendra Xerox
All Software Material Available
Opp: Satyam Theater Back Gate,
Ameerpet, Hyderabad.
Ph: 9951596199

Sri Raghavendra Xerox
All Software Material Available
Opp: Satyam Theater Back Gate,
Ameerpet, Hyderabad.
Ph: 9951596199