## How the Process would be fast while we enter any url (google.com) in browser?

|  |
| --- |
|  |

## Full Stack Developer flow:-

|  |
| --- |
|  |

## WWW and Internet

WWW (World Wide Web) – Is the common language that computer can speak.

## Developer should check the code should support all browsers, mobile etc.

## LAMP Stack – (Linux, Apache, MySQL and PHP)

## Header Text – h1 to h6

## Strong Tag <strong> was understand by all browser instead of <b> and <em> instead of <i> reason is due to semantic html

## Order List <ol><li></li> </ol> tag which will show as 1, 2, 3 and <ul><li></li><ul> will show as …

## In Form name attribute is used to send data to backend in query parameter

|  |
| --- |
| <form method="GET">  First Name: <input type="text" name="firstname"> <br>  Last Name: <input type="text" name="lastname"> <br>  Email: <input type="email" name="email" required><br>  Password: <input type="password" name="password" min="5"><br>  Birthday: <input type="date" name="birthday"> <br>  Gender: <br>  <input type="radio" name="gender"> Male <br>  <input type="radio" name="gender"> Female <br>  <input type="radio" name="gender"> Other <br>  Pets: <br>  <input type="checkbox" name="cat"> Cat <br>  <input type="checkbox" name="dog"> Dog <br>  Cars: <br>  <select name="car">  <option value="volvo" value="volvo">volvo</option>  <option value="Audi" value="audi">Audi</option>  </select><br>  <input type="submit" value="Register">  <input type="reset">  </form> |

GET will send data in query string but POST will send data in http body.

## Div tag are used to divide content <div></div> , and also allows to add styles into each different sections.

## Span is same things as div but it is used in inline elements. <span></span>

## Head tag is basically used to declare what that particular html page needs like (external css)

## Html5

* + It is latest version of Html which supports all browsers including mobile devices.
  + It introduces new semantic elements (they want to make website more descripted.

|  |
| --- |
|  |

* Header tag for declaring all the headers tags(h1…h6)
* Nav tag for declaring all the navigation link (<a href=””>)
* Footer tag for adding some footer paragraph (<p></p>)

Overall html5 tag tries to improve the performance of the web

## CSS

**Check this site for any lookup** [**https://css-tricks.com/almanac/**](https://css-tricks.com/almanac/)

**For selecting colors** <http://www.paletton.com/>

### Adding external css files in html page

|  |
| --- |
| <head>  <title>Register</title>  <link rel="stylesheet" type="text/css" href="style.css">      </head> |

* + If any property is repeated in the css file, it will take the last input for that particular property.
  + **Background-size: cover** is used to cover the background properly with full image
  + **List-style:none** is used to remove the bullet or number from list.
  + **Cursor:pointer** to change the cursor of mouse to pointer.
  + **Display: inline-block** to make all the elements in single straight line
  + **Class** (.classname) is used to select the group of elements to use the same design and also we can use multiple classes to a same element.

|  |
| --- |
| <h2 class="Webtext Active">Hello</h2> |

* + Id(#id) is similar to class but it can used only once for same id
  + \* is used to assign the design to whole page
  + Element, elemet (e.g: - h1,p1{color:red} )
  + Element element (eg:- h1 p {color: red} means I want to select all p inside h1.
  + Element > element (eg:- h1 > p ) all paragraph that have parent of h1
  + Element + Element (eg:- h1 + p) select any p element right after h1.
  + :hover is used when we want apply the style when hover your mouse **(eg :- p:hover {color: red}**
  + :last-child suppose you apply multiple class to different element but you want to apply for only last child ( eg:- .webtext:last-child {….})
  + :first-child suppose you apply multiple class to different element but you want to apply for only first child ( eg:- .webtext:first-child {….})
  + !important it overrides any of the cascading styles rules and says.(eg:- p{color:pink !important})
  + What selectors win out in the cascade depends on:-
    - Specificity is higher it is good.( <https://specificity.keegan.st/>)
    - Important
    - Source Order: - means if you attached you style sheet in a html file, the stylesheet present appears at bottom got more dominant.
  + Text-decoration eg( p {text-decoration : underline(line-through)} );
  + Text-transform eg(p{text-transform: uppercase} )
  + Line-height (eg: p {line-height: 20px} )
  + Font-style : italic, font-weight: bold, font-size: 80%, font-family: Georgia;

Font-family : “Times New Roman” , Georgia – this means any system doesnot support Times New Roman than show the text in Georgia

* + Google fonts we can only font directly the url.
  + Float – is used to float some image or div to left or right ( img {float: left} )
  + Box-Model:

|  |
| --- |
|  |

|  |
| --- |
|  |

* + Pm vs em vs rem
  + Always minify the css (<https://cssminifier.com/>) so that load of the server can be minimized for that webpage which make faster.

## Flex Box

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

<https://flexboxfroggy.com/>

<https://darekkay.com/dev/flexbox-cheatsheet.html>

Flex is very easy to implement and it makes the website really nice, we can position the content very easily.

* + How to flex box inside a class

|  |
| --- |
| .container {  display: flex;  } |

* + How to wrap inside the class

|  |
| --- |
| .container {  display: flex;  flex-wrap: wrap;  } |

* + How to make the content to center

|  |
| --- |
| .container {  display: flex;  flex-wrap: wrap;  justify-content: center;  } |

## CSS3

CSS3 is the latest version of CSS.

Check which all browser support:

<https://www.w3schools.com/cssref/css3_browsersupport.asp>

<https://caniuse.com/>

Some of the new properties in CSS3

* + Transition – We want to image to load big on mouse hover.

|  |
| --- |
| img {  width: 450px;  height: 300px;  margin: 10px;  transition: all 4s;  }  img:hover{  transform: scale(1.1);  } |

## Bootstrap

Bootstrap is nothing but inbuilt CSS with JavaScript which we can use in our website, instead of writing the website from stretch.

For including Bootstrap we can download the version else we can use the CDN version.

* + Layout in Bootstrap and Grid

Bootstrap has idea of grid system and this grid will have 12 lines which we can divide.

|  |
| --- |
| <div class="container">  <div class="row">  <div class="col col-sm-6">  1 of 2  </div>  <div class="col col-sm-3">  2 of 2  </div>  <div class="col col-sm-3">  extra  </div>  </div>  </div> |

We have different size and pixel.

|  |
| --- |
|  |

* + Blow Meta tag is important if you want the page to be responsive

|  |
| --- |
| <!-- Required meta tags -->  <meta charset="utf-8">  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no"> |

* + Background image adjustment in css

<https://css-tricks.com/perfect-full-page-background-image/>

* + Bootstrap class to make text upper case

|  |
| --- |
| <h1 class="text-uppercase"><strong>The biggest startup event of the year</strong></h1> |

* + Bootstrap button classes

|  |
| --- |
|  |

* + Css class to make the horizontal line smaller in size

|  |
| --- |
| hr{  border-color: #f05f44;  border-width: 3px;  max-width: 65px;  } |

* + Bootstrap class to make text center

|  |
| --- |
| <header class="text-center col-12">  <h1 class="text-uppercase"><strong>The biggest startup event of the year</strong></h1>  </header> |

* + Bootstrap class to enable flexes box functionality and align item center

|  |
| --- |
| <body>  <div class="container d-flex align-items-center">  <div class="row">  <header class="text-center col-12">  <h1 class="text-uppercase"><strong>The biggest startup event of the year</strong></h1>  </header>  <section class="text-center col-12">  <hr>  <button class="btn btn-primary btn-xl">Find out more</button>  </section>  </div>  </div>  </body> |

* + Bootstrap class to make height 100 %

|  |
| --- |
| <body>  <div class="container d-flex align-items-center h-100">  <div class="row">  <header class="text-center col-12">  <h1 class="text-uppercase"><strong>The biggest startup event of the year</strong></h1>  </header>  <section class="text-center col-12">  <hr>  <button class="btn btn-primary btn-xl">Find out more</button>  </section>  </div>  </div>  </body> |

## MailChimp

Is an email marketing service, there is other tools like mailerlight etc… That allows you to user click on a button submit a form and save the information somewhere to use later.

## Animate

<https://daneden.github.io/animate.css/>

Very useful if you want to animate elements.

## Creative Tim

They create bootstrap themes, they are using the bootstrap tool kits to build their own styles and little customizes actions to make even better.

<https://www.creative-tim.com/>

## Free Html/CSS templates

<http://www.mashup-template.com/templates.html>

## CSS Grid

CSS Grid Vs Flexbox

* Ideally Flexbox is used to make One dimension like columns and how to do layout based on columns
* CSS Grid is more powerful for two dimension layout i.e. both columns and rows like image gallery, portfolio project.
* Both we can do all in bootstrap by using Grid system, then why CSS grid because if you know the basis if will give you extra points.

We don’t use either Flexbox or CSS Grid, ideally we can combine together to make a better layout page

* + Enable grid for a class/container (which will wrap all other element)

|  |
| --- |
| .container{  display: grid;  } |

* + How to set the column size of a page inside CSS grid ?

It will take 300px of two grid/columns. We can also use % instead of px which is definitely better.

Instead of all % or px or em or rem. CSS grid gives a really good tool called fraction (fr) fr means if you do 1fr will be get one column and so one.

|  |
| --- |
| .container{  display: grid;  grid-template-columns: 300px 300px;  } |

It means three columns first two columns with 1fr and third one with 2fr (half the size of first two columns)

|  |
| --- |
| .container{  display: grid;  grid-gap: 20px;  grid-template-columns: 1fr 1fr 2fr;  } |

* + How to set the column/grip gap?

|  |
| --- |
| .container{  display: grid;  grid-gap: 20px;  grid-template-columns: 300px 300px;  } |

* + How to set the rows?

The first row set to 1fr and second set to twice big a first row

|  |
| --- |
| .container{  display: grid;  grid-gap: 20px;  grid-template-columns: 1fr 1fr 2fr;  grid-template-rows: 1fr 2fr;  } |

Grid-template-columns : 1fr 1fr 1 fr can written as repeat(3, 1fr) as well from grid-template-rows

* + How to set auto size to columns and rows?

Its automatically resizes to fit the content.

|  |
| --- |
| .container{  display: grid;  grid-gap: 20px;  grid-template-columns: auto;  grid-template-rows: 1fr 2fr;  } |

.container{

display: grid;

grid-gap: 20px;

grid-template-columns: repeat(auto-fill,minmax(200px,1fr));

grid-template-rows: 1fr 2fr;

}

Just check the cheat sheet: <http://grid.malven.co/>

## JavaScript

It’s a file to write instruction to a computer

### JavaScript Types

1. Number (NaN)

2. String

3. Boolean

4. Undefined – means the variables is not been assigned.

5. Null

6. Symbol (new in ECMAScript 6) - check in ES6 section

7. Object

### JAVASCRIPT VARIABLES

Variable needs to start with letter but can end with number. For coding standard variables should be camel case.

e.g: var firstName = “chiranjit Saha”

var

let (new in ECMAScript 6) check in ES6 section

const (new in ECMAScript 6) check in ES6 section

### How to convert string to number?

Number(variableName)

### JAVASCRIPT CONDITIONALS

* if
* else
* else if
* ternary operator (condition ? exp1 : exp2)
* switch

### JAVASCRIPT LOGICAL OPERATORS

* &&
* ||
* !

### JAVASCRIPT FUNCTIONS

1. var a = function name() {} // Anonymous Function assign function to a variable
2. function name() {}
3. return
4. () => (new in ECMAScript 6) check in ES6 section

### JAVASCRIPT DATA STRUCTURES

1. Array can also have function(), array, number, string, Boolean also mixed but don’t mixed as it leads to performance issue. And objects also

List =[“apple”, “mango”,” banana”];

List.shift();// will remove first element i.e. apple.

List.concat(“watermelon”, “grapes”). // multiple value can be add to end of the array

1. Object – can have array also can have objects(methods).

Var user = {

username : ‘chiranjit’,

password : ‘ssjkdjs’,

favoriteFood : [‘egg’, ‘dryfish’, ‘prawn’]

}

### JAVASCRIPT LOOPING

**Both arrays and strings are iterating in JavaScript**

1. for
2. while
3. do
4. forEach (new in ECMAScript 5)

|  |
| --- |
| basket.forEach(item => {      console.log(item);  }) |

1. for in

|  |
| --- |
| const basket = ['apples','mango','oranges'];  for (item of basket){  console.log(item);  } |

1. for in

**It works for objects to check the object properties. With we are not iterating rather we are enumerating.**

|  |
| --- |
| const detailedBasket = {  apples: 5,  oranges: 10,  grapes: 1000  };  for (item in detailedBasket){  console.log(item);  } |

### Scope

\*\* If statements don’t create new scope

|  |
| --- |
| //Root Scope  var fun = "Hello";  function funnest(){      //child scope      // here it will print Hello      console.log(fun);  } |
|  |

## JavaScript ES5 and ES6 (ECMAScript)

Babel helps to convert ES5-6 syntax to older syntax so that the browser can support <https://babeljs.io/>

### Let and const

Const variable it can’t be update and also if we declare an object as constant but we can still change the property of the object.

Eg:-

const name =”chiranjit”

let id= “001340013”

if(name === “chiranjit”)

{

let id = “00123991” // new scope is present inside any {} bracket

console.log(id) // will print 00123991  
}

console.log(id); // will print 001340013

// **for var it will be**

const name =”chiranjit”

var id= “001340013”

if(name === “chiranjit”)

{

var id = “00123991”

console.log(id) // will print 00123991  
}

console.log(id); // will print 00123991

### Destructuring

**Eg:-**

|  |
| --- |
| const obj = {      player: 'chiranjit',      experience: 5.5,      wizardLevel: false  }  const player = obj.player;  const experience = obj.experience;  let wizardLevel = obj.wizardLevel;  // Instead we can write.  const {player,experience} = obj;  let {wizardLevel} = obj |

### Object Properties

The new ways of declaring properties:-

|  |
| --- |
| const name = "chiranjit Saha";  const obj = {      [name]:'hello',      [1+2]: 'hihi'  } |

|  |
| --- |
| const a ="chiranjit";  const b = true;  const c = {};  const obj = {      a: a,      b: b,      c:c  }  // Instead we write a below if the porerties match the value  const obj = {      a,      b,      c  } |

### Template Strings

|  |
| --- |
| const name = "chiranjit";  const age = 28;  const pet = "dog"  const greeting = "Hello " + name + " your age " + age + " and your pet " + pet;  // alternate we can use the back tick `  const greeting = `Hello ${name} your age ${age} and your pet ${pet}`; |

### Default Arguments

*If nothing is passed while calling this below greet function it will take the default one.*

|  |
| --- |
| function greet(name="", age=30, pet="dog"){      return `Hello ${name} your age ${age} and your pet ${pet}`;  } |

### Symbol

|  |
| --- |
| let sm1 = Symbol();  let sm2 = Symbol('foo');  let sm3 = Symbol('foo');  sm2 === sm3 // it will return false if the value looks like same also |

### Arrow function

|  |
| --- |
| function add(a,b){      return a+b;  }  // arrow function syntax below and also if their is single return we can put in one line  const add = (a,b) => a+b;  const add = (a,b) => {      return a + b;  } |

## JavaScript ES7

### Includes()

It can be done on string also.

|  |
| --- |
| const pets = ['cat','dog','bat'];  pets.includes('dog');//true  pets.includes('bird');//false |

### Exponential Operator(^)

Here ^ we use \*\*

|  |
| --- |
| const square = (x) => x\*\*2; |

## JavaScript ES8

### String Padding (padStart and padEnd)

|  |
| --- |
| 'chiranjit'.padStart(20); // will output space including the string of 20 character                          // space will add starting.  'chiranjit'.padEnd(20); // will output space including the string of 20 character                              // space will add end. |

### Objects(keys, values, entries)

|  |
| --- |
| let obj = {      username0:'abc',      username1:'def',      username2:'ghi'  }  Object.keys(obj).forEach((key,index)=>{      console.log(key,obj[key]);  })  // alternative using values  Object.values(obj).forEach(value =>{      console.log(value);  })  //alternative using entries getting in array  Object.entries(obj).forEach(value =>{      console.log(value);  }) |

## *JavaScript ES9*

### Flat

Flat() is a method that we can use on an array. It is used to remove the nested array and put in single array.

|  |
| --- |
| const array = [1,[2,3],4,[5,7],29];  array.flat(); // [1, 2, 3, 4, 5, 7, 29] here it will remove only one layer of nested array by default 1 layer |

Interesting in flat() we can say how many array to flank the array

|  |
| --- |
| const array1 = [1,[2,3],4,[[5,7],29]];  array1.flat(2); // [1, 2, 3, 4, 5, 7, 29] |

And also it removes the empty index from the array

|  |
| --- |
| const array2 = [1,2,,,,,,3,4,5];  array2.flat(2); // [1, 2, 3, 4, 5] |

### flatMap

flatMap() is the method where we can apply flat as well as Map together to an array.

### TrimStart() and trimEnd()

Both the method is used to remove the empty space for starting and ending of the string respectively.

### formEntries

It transforms the list of the key value pairs into an object.

|  |
| --- |
| const userProfiles = [['chiranjit',28],['Arijit',15],['Alpana',49],['Sanjit',53]];  Object.fromEntries(userProfiles); //{chiranjit: 28, Arijit: 15, Alpana: 49, Sanjit: 53} |

### Try , catch block

### Object Spread Operator

|  |
| --- |
| const animals = {  tiger: 23,  lion: 11,  dog: 10  }  const {tiger,...rest} = animals;  //tiger = 23  // rest = {lion: 11, dog: 10} |

## *Advanced Functions*

### Closure

A function runs, the function executed. It’s never going to execute again. But it’s going to remember that there are references to those variables. So the child scope always has access to the parent scope. But parent scope doesn’t have access to child scope.

e.g:-

|  |
| --- |
| const first = () => {  //parent scope      const greed = 'Hi'      const second = () => {  //child scope          alert(greed);      }      return second;  }  const newFuc = first(); |

### Currying

The process of converting a function that takes multiple arguments into a function that takes one at a time.

|  |
| --- |
| const multiply = (a,b) => a\*b;  //converting to currying  const curriedMultiply = (a) => (b) => a\*b;  curriedMultiply(3)(4);  const multiplyBy5 = curriedMultiply(5);  multiplyBy5(4); // 20 |

### Compose

The act of putting two functions together to form a third function where the output of one function is the input of the other.

|  |
| --- |
| const compose = (f,g) => (a) => f(g(a));  const sum = (num) => num + 1;  compose(sum,sum)(5); // output 7 |

### Avoiding Side Effects, functional purity.

|  |
| --- |
| var a = 10;  function b() {      a = 20;// that's the side effect as function has its own scope/universe              // if it’s affecting the outside world, good practice to avoid this side affect  } |

**What are the two elements of a pure function?**

1. Deterministic --> always produces the same results given the same inputs

2. No Side Effects --> It does not depend on any state, or data, change during a program’s execution. It must only depend on its input elements.

## JavaScript Advanced Arrays

|  |
| --- |
| const array = [1,2,4];  const double = [];  const newArray = array.forEach((num)=>{      double.push(num \* 2);  }) |

### Map

Looping of each element in an array and return a new array which we can’t do with forEach loop.

Here we need to always return value

|  |
| --- |
| const array = [1,2,4];  const mapArray = array.map((num) => {  return num \* 2;  });  //short syntax removing the {}  const mapArray = array.map(num => num \* 2); |

### Filter

Filtering the array based on the condition of return.

|  |
| --- |
| const array = [1,2,4,6,10];  const filterArray = array.filter(num =>{      return num > 2;  })  // single syntax  const filterArray = array.filter(num => num > 2); |

### Reduce

We can filter as well as mapping with reduce.

|  |
| --- |
| //accumulator nothing but act as a index of the array  // we can set the default value  const array = [1,2,4,6,10];  const reduceArray = array.reduce((accumulator,num)=>{      return accumulator \* num;  },0); |

## Advanced Objects

### Reference Type

Objects are reference type

Let object1 = {value :10};

Let object2 = object1;

### Context

Where we are within the object.

|  |
| --- |
| const object4 = {      a:function(){          console.log(this); // Here this is the window object      }      } |

### Instantiation

Make a copy of the object and reuse the code.

|  |
| --- |
| class Player {      constructor(name, type){          this.name = name;          this.type = type;      }  introduce(){          console.log(`Hi I am ${this.name}, I am a ${this.type}`);      }  }  class Wizard extends Player{      constructor(name,type){          super(name,type);      }      play(){          console.log(`I am a ${this.type}`);      }  }  const wizard1 = new Wizard('chiranjit','Programmer'); |

## How JavaScript Works

**Memory Heap**: - Whenever we are declaring variable, array, objects memory is allocated to memory heap.

**Memory Leaks** happens when we have unused memory just laying around in Memory Heap it fills up the memory. That’s why global variable is very bad because we are not cleanup which leads to memory leak.

**Call Stack: -** where we are executing the code.

**JavaScript is a single threaded language that can be non-blocking because it has only one call stack.**

**Other Language can have multiple call stack, these are called multi thread.**

**Issue with multi thread environment is deadlock.**

**Synchronous Program means it execute line by line.**

### JavaScript Runtime Environment

Now the asynchronous program in running using Callback function, this callback function get runs in the background.

|  |
| --- |
|  |

## JavaScript Modules

In ES6 very nice way we can separate the module using import and export.

|  |
| --- |
| //In 1.js  export const add = (a,b) => a+b;  //In 2.js  import {add} from './add'; |

With WebPack we can use ES6 in all browser.

## Pass by Value vs Pass by Reference

Object, Array is reference type.

|  |
| --- |
| let a = [1,2,3,4,5,6];  let b = a;  //How to copy an object of array without reference type,  // so that if you modify something in array a is not reflected  // in array b;  let b = [].concat(b); |

|  |
| --- |
| let object1 = {value:10};  let object2 = object1;  //Instead of reference to same object1 in object2  //we can clone so that if anything change in object1  // is not reflected in object2  let object2 = Object.assign({},object1);  // Same thing using spread operator.  let object2 = {...object1}; |

### Shallow Clone

|  |
| --- |
| // Shallow clone where we clone the first layer  let object1 = {      value:10,      type: {          deep:'try an copy me'      }  }  //it will not work as clone for type instead it will take as reference type  let object2 = Object.assign({},object1);  // Same thing using spread operator.  let object2 = {...object1}; |

### Deep Clone

How can we do deep clone for the above problem using JSON. But this will leads to performance implication

|  |
| --- |
| // Shallow clone where we clone the first layer  let object1 = {      value:10,      type: {          deep:'try an copy me'      }  }  //Using JSON we can do the deep cloning  let superClone = JSON.parse(JSON.stringify(object1)) |

## Type Coercion

<https://dorey.github.io/JavaScript-Equality-Table/>

<https://www.ecma-international.org/ecma-262/5.1/#sec-11.9.3>

Always try to use ===, if type coercion in JavaScript is really tricky.

|  |
| --- |
| 1 == "1"; // true as it will convert one of them to respective type |

## DOM Manipulation

DOM is something which browser creates to allow us to modify the html and CSS. Web browser allows us to access the DOM through **document** object. **The parent object for document is window object. Window is the object that describe browser.**

**JavaScript can speak with the document object modify the html and css.**

1. getElementsByTagName
2. getElementsByClassName
3. getElementById

## querySelector – It select the first item that it find.

1. querySelectorAll
2. getAttribute – eg **document.querySelector(“li”).getAttribute(“random”);**
3. setAttribute - eg **document.querySelector(“li”).setAttribute(“random”,”1000”);**
   * Changing Styles
4. style.{property} – eg:- **document.querySelector(“h1”).style.background = “yellow”;**

It will break the separation of principle concept to directly change the html. Instead we will use the below.

1. className – e.g:- **document.querySelector(“h1”).className =”give the classname of css file”**
2. classList
   * classList.add – eg:- **document.querySelector(“li”).classList.add(“give the classname”);**
   * classList.remove – eg:- **document.querySelector(“li”).classList.remove(“give the classname”);**
   * classList.toggle – this is really useful if we want to toggle something like on-off.

eg:- **document.querySelector(“li”).classList.toggle(“give the classname”);**

* + Bonus

1. innerHTML :- change the html element to some other element.

e.g:- **document.querySelector(“h1”).innerHTML = “<h2>Hello</h2>”**

1. parentElement : for getting the parent element of a element

e.g:- **document.querySelector(“h1”).parentElement;**

1. children: for getting all the children of a parent.

e.g:- **document.querySelector(“body”).children;**

**It is important to CACHE selectors in variables.**

## DOM Events

Events are things like clicking, mouse entering or hovering, user trying something in a search bar etc. We can listen to these events and react to them using JavaScript.

Check all the events <https://developer.mozilla.org/en-US/docs/Web/Events>

### Mouse Event

1. Click:-

Eg:- var button = document.getElementByTagName(“button”)[0];

// to access the first element of button we include[0]

**button.addEventListener(“click”, function(){**

**Console.log(“click”);**

**} );**

1. Mouseenter:

e.g:- **button.addEventListener(“mouseenter”, function(){**

**Console.log(“click”);**

**} );**

1. Mouseleave

e.g:- **button.addEventListener(“mouseleave”, function(){**

**Console.log(“click”);**

**} );**

1. Adding at end of the list when button is clicked (**createElement, createTextNode,** **appendChild)**

var button = document.getElementById(“click”);

var input = document.getElementById(“userInput”);

var ul = document.querySelector(“ul”);

button.addEventListenent(“click”,function(){

if(input.value.length >0) {

var li = document.**createElement**(“li”);

li.append(document.**createTextNode**(input.value);

ul.**appendChild**(“li”);

input.value = “”;

});

}

1. Keypress

Get the keycodes :<https://www.cambiaresearch.com/articles/15/javascript-char-codes-key-codes>

How to get the event in a function.

Input.addEventListener(“keypress”,function(**event**){

Console.log(event);

if(input.value.length >0 && event.keycode === 13) {

var li = document.**createElement**(“li”);

li.append(document.**createTextNode**(input.value);

ul.**appendChild**(“li”);

input.value = “”;

});

})

## JQuery

JQuery slows may JavaScript in late 2006. But disadvantages it is imperative means (we need tell the program exactly what to do one by one, one action is dependent on another action which is dependent on another action and so on). That means when you website big and complicated it create a huge mess.

Now JavaScript can created many declarative framework or library came like Reactjs, Angular, Vue.js etc.

## Developer Fundamentals

All minimize the DOM manipulation and events.( So we can use react, angular, vue etc.)

## Terminal Command

* + For Window:-
    - dir – Everything in current directory.
    - cd – What is the current directory
    - cd .. – Move to one directory up.
    - cls – To clear the terminal.
    - cd / - To move to root directory.
    - mkdir folderName – To create a new folder.
    - start . – To open the folder.
    - rename index.html about.html
    - del about.html
    - deltree foldername
    - How to change the cmd

<https://www.howtogeek.com/howto/16815/how-to-personalize-the-windows-command-prompt/>

* + For Linux/Mac
    - ls –
    - pwd –
    - clear –
    - open

## GitHub

* + How to clone a repository in local drive?

Command:- git clone <repositorypath> <https://github.com/Chiranjit9955/ChiranjitSaha.github.io.git>

**Below all command will not work if you are not in the repository folder**

* + How to check the status of the folder repository?

**First go to the inside the folder of the repository**

Command:- **git status**

* + How to add new file in the repository?

Command: - git add <filename>

Command:- git add . (for adding all the files in one command)

After add the new files we need to commit?

Command: - git commit –m”<commit message>”

Finally the push command

Command: - git push

* + How to pull the new changes to local folder

Command:- git pull

* + How to compare the local file with the branch file?

Command:- git diff

* + Branching

Master branch is the branch of code present in Production. Don’t directly work on master instead create a new branch

|  |
| --- |
|  |

* + How to check all the branches we have?

Command: - git branch

* + How to create a new branch?

Command: - git branch <newbranchname>

* + How to move to the other branch?

Command: - git checkout <branchname>

* + How to push to branch

Command:- git push origin <branchname>

* + How to merge one branch to other branch?

**Firstly should be in the branch from where we want to pull. Then run the below command.**

Command:- git merge <branchname>

## Open Source

Open source software is software with source code that anyone can inspect, modify, and enhance.

<https://github.com/zero-to-mastery>

## Npm (Node package manager)

Npm Created for developer which able to share the JavaScript they have written.

Developer write code and upload to Npm Registery and from Npm Register other developer can download the file for their use.

|  |
| --- |
|  |

Three things is very important :-]

1. HTML + CSS + JavaScript.
2. Github repository + GIT
3. Package.json file :-File manage the dependency which the packages written by other developer with NPM

**Npm command**:

npm init – Whenever we are creating a new project

npm install – To install all the package required for the project, which is verify from package.json file

* + Npm allows install in two ways:-

1. Globally (npm i –g <package name>) – It can be used anywhere in the computer.
2. Locally (npm i <package name>)– It can be used in that project only.

Lodash(it has many extension function of JavaScript) and liveserver(if you want to host the application locally just like webpack) is good just have a look.

* + To check version of any package

<https://semver.npmjs.com/>

## Node JS

Node js actually uses V8 engine to run JavaScript outside of the browser. Node.js is most popular for building servers, it’s really fast. It also really good for chat application.

In Node we don’t have window object as we see in browser console. Instead we have **global** and also we don’t have document or fetch object in global, i.e. other than browser specific things we can do anything with node.

We also have another object **process. Like process.exit();**

### How to Create, run a Script file in Node and some of the command

* + touch <<scriptname>>.js – To create the script.
  + node <<scriptname>>.js – To run the script , end the process and comes to the terminal.
  + Console.log(\_ \_ dirname) – to get the directory

### Modules in Node

Import and export will not work in Node (but it will work Node version 12.2.0)

Instead use the common js syntax.

|  |
| --- |
| import largeNumber from './Script1.js';  //instead  const Script1 = require('./Script1.js');  const a = 10;  const b = Script1.largeNumber;  console.log(a+ b);  // for export  const largeNumber = 200;  export default largeNumber;  //instead  module.exports ={  largeNumber: largeNumber  } |

### Types of Modules

1. Custom module which we are creating in the above example.
2. File system module which is used for file manipulation.

|  |
| --- |
| const c = require('fs'); |

1. Http Module – Is used to build a server.

|  |
| --- |
| const c = require('http'); |

1. Nodemon

It act a webpack in react like automatically it will detect changes in the project.

npm install nodemon

### Building a Server

We can also change the content-type to JSON

|  |
| --- |
| const http = require('http');  const server = http.createServer((request, response) =>{  response.setHeader('Content-Type','text/html');  response.end('<h1>Hello</h1>');  //or  //if the content type is JSON  const user = {  name:'Chiranjit',  hobby: 'Programmer'  }  response.setHeader('Content-Type','application/json');  response.end(JSON.stringify(user));  });  server.listen(3007); |

Instead of creating server using http, JavaScript has many other inbuilt libraries to use. Most popular is Express.js

### Introduction to Express.js

<https://expressjs.com/>

npm install express

There no need of content-Type it simply convert object to json.

**We can also do post, put, delete also.**

|  |
| --- |
| const express = require('express');  const app = express();  app.get('/',(req,res)=>{  res.send("hello");  })  //or Json pass the object  const user = {  name: 'Chiranjit',  hobby: 'Programmer'  }  app.get('/profile',(req,res)=>{  res.send("getting profile");  })  res.send(user);  app.listen(3008); |

### Express Middleware

If you want to some operation before any request to process we can use middleware.

Middleware receive ahead of time before it gets to the route the request, modifies it and passes the next function to keep it going.

|  |
| --- |
| const express = require('express');  const app = express();  app.use((req,res,next)=>{  console.log("Middle ware operation");  next();  })  app.get('/',(req,res)=>{  res.send("getting profile");  })  app.listen(3008); |

### Body-parser

If we want to use the request object (**like what data send in request body**) in express we need to use the middleware, the middleware we want is called body parser.

npm install body-parser

|  |
| --- |
| const express = require('express');  const bodyParser = require('body-parser');  const app = express();  //for request body is from form data  app.use(bodyParser.urlencoded({extended:false}));  // for request body is JSON object  app.use(bodyParser.json());  app.post('/profile',(req,res)=>{  console.log(req.body);  user ={  name: "chiranjit",  hobby: "Programmer"  }  res.send(user);  })  app.listen(3008); |

### Restful API

Is defines the set of functions which developer can perform request and receive responses via http protocol such as (get, put, post, delete).

With restful API we able to create a API is restful something that follows the that everybody can agree on, so that we have compatibility between different system.

Restful APIs are stateless means calls can be made independently of one another and each call contains all data necessary to complete it successfully. Server doesn’t need to keep memorizing things, each request that comes in has enough information that the server can respond it.

* **If we want to get data from query parameter in get request.**

<http://localhost:3008/?name=chiranji&age=28>

|  |
| --- |
| const express = require('express');  const app = express();  app.get('/',(req,res)=>{  console.log(req.query);  res.send("getting root");  })  app.listen(3008); |

* **If we want get data from request body**

We talk in Body Parser section and req.body

* **If we want get the request headers data**

|  |
| --- |
| app.get('/',(req,res)=>{  console.log(req.headers);  res.send("getting root");  }) |

* **It we want to get the parameter value**

|  |
| --- |
| const express = require('express');  const app = express();  app.get('/:id',(req,res)=>{  console.log(req.params);  res.send("getting root");  })  app.listen(3008); |

* **If we want to send status with response**

|  |
| --- |
| const express = require('express');  const app = express();  app.get('/:id',(req,res)=>{  console.log(req.params);  res.status(404).send("not found");  })  app.listen(3008); |

* **How to send static files dat like html , text etc to the response**

**Inside the public folder index.html is present.**

|  |
| --- |
| const express = require('express');  const app = express();  app.use(express.static(\_\_dirname + '/public'));  app.listen(3008); |

### Node File System Module

* + **How to read a file**

|  |
| --- |
| const fs = require('fs');  fs.readFile('./File.txt',(err,data)=>{  if(err){  console.log("Errorr");  }  console.log(data.toString("utf8"));  }) |

* + **How to append a text in a file**

|  |
| --- |
| fs.appendFile('./File.txt','This is so cool', err =>{  if(err){  console.log(err);  }  }); |

* + **How to write in a file**

|  |
| --- |
| fs.writeFile('./File1.txt','Sad to see you go', err=>{  if(err){  console.log(err);  }  }) |

* + **How to delete a file**

|  |
| --- |
| fs.unlink('./File1.txt',err =>{  console.log(err);  }) |

### Bcrypt Node.js

<https://www.npmjs.com/package/bcrypt-nodejs>

It allows us to create a very secure login.

Always remember when we are submitting sensitive data (like password etc.), we should not send this information in query string by get request instead we need to add a post request so that information we will present in the request body. And also we need to send over https and also we store the information in hash.

|  |
| --- |
| bcrypt.hash("bacon", null, null, function(err, hash) {  // Store hash in your password DB.  });  // Load hash from your password DB.  bcrypt.compare("bacon", hash, function(err, res) {  // res == true  });  bcrypt.compare("veggies", hash, function(err, res) {  // res = false  }); |

**Highlight:**

1. **Always send any sensitive information from the front end to the backend using https in a post body.**
2. **Always store the password in database using hash, and every time the user signs in check that whatever user input it.**

**Enable CORS**

npm install cors

<https://www.npmjs.com/package/cors>

|  |
| --- |
| var express = require('express')  var cors = require('cors')  var app = express()    app.use(cors())    app.get('/products/:id', function (req, res, next) {  res.json({msg: 'This is CORS-enabled for all origins!'})  })    app.listen(80, function () {  console.log('CORS-enabled web server listening on port 80')  }) |

## React.js

Before React, jQuery was their which leads to inconsistency are totally sequentially. React can be used in mobile devices; virtual reality apps, desktop apps anywhere have a view react can use it.

React divide everything in small component, which at the end combines to create the website.

And Components can be reusable.

|  |
| --- |
|  |

### Data flow in react is top-bottom only

|  |
| --- |
|  |

### Virtual DOM

As we know always need to minimize the amount of DOM manipulation, because it reduces the performance of the web and also increases bug.

With React it creates the virtual DOM. Virtual DOM is just a JavaScript object and this object just describes the current state of the website.

We just give the object to react and the react bot will automatically make changes to the DOM in the most optimum way possible.

### Eco-System

React has great ecosystem. It is probably one of the biggest ecosystems in JavaScript

|  |
| --- |
|  |

### create-react-app

npm install –g create-react-app

This is the package which creates a startup project that contains webpack, babel, lint and debugging.

### Creating the new project

Create-react-app <project name>

New ways is

npx create-react-app my-app

<https://create-react-app.dev/docs/getting-started/>

### Creating a Component

React has the idea of component that better to have functionality in style per component, so that each component has own separate universe.

Step 1:- Create a js file

1. It needs to import React and component.
2. Create class which extends Component.
3. Export the class.

Adding CSS file to the component.

1. Create a new css file and import it in the component

Hello.js

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  class Hello extends Component {  render() {  return (  <div>  <p>Welcome to React</p>  <h1>Hello World</h1>  </div>  );  }  }  export default Hello; |

Step 2:- How to use in other file.

All remember in React application we will have React package as well as ReactDOM to connect react to the DOM

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import Hello from './Hello';  import \* as serviceWorker from './serviceWorker';  ReactDOM.render(<Hello />, document.getElementById('root')); |

### Tachyons

npm install tachyons

Tachyons allows us to have similar to bootstrap, predefined class names that we just add to components to make things look really nice.

<https://tachyons.io/>

After install just import in the component “import tachyons” basically in index.js

Why className because this html code is called as JSX. It allows writing Html syntax in JavaScript.

And always return one element otherwise it will give error.

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  class Hello extends Component {  render() {  return (  <div className="f1 tc">  <p>Welcome to React</p>  <h1>Hello World</h1>  </div>  );  }  } |

### Props

It allows send data from parent to child using string interpolation.

Allows remember all JavaScript code should put inside {} in JSX code

**In Index.js (parent)**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import Hello from './Hello';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  ReactDOM.render(<Hello greeting={"Hello" + "How are you"}/>, document.getElementById('root')); |

Now this greeting can be used in child component in Hello

**In Hello.js (Child)**

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  class Hello extends Component {  render() {  return (  <div className="f1 tc">  <p>Welcome to React</p>  <h1>{ this.props.greeting}</h1>  </div>  );  }  } |

How to convert the component class to function?

|  |
| --- |
| import React, {Component} from 'react';  import './Hello.css';  const Hello =(props) =>{  return (  <div className="f1 tc">  <p>Welcome to React</p>  <h1>{props.greeting}</h1>  </div>  );  }  export default Hello; |

### Export without default

Export without default can have multiple exports.

|  |
| --- |
| export const robots = [  {  id: 1,  name: 'Leanne Graham',  username: 'Bret',  email: 'Sincere@april.biz'  },  {  id: 2,  name: 'Ervin Howell',  username: 'Antonette',  email: 'Shanna@melissa.tv'  }  ] |

While importing it other file it should be in {} so that we can pass multiple export

import {robots} from './robots';

### How to use template string in the component

**In child component**

|  |
| --- |
| import React from 'react';  const Card = (props) => {  return (  <div className="tc bg-light-green dib br3 pad3 ma2 grow bw2 shadow-5">  <img alt="robots" src={`https://robohash.org/${props.id}?200x200` }/>  <div>  <h2>{props.name}</h2>  <p>{props.email}</p>  </div>  </div>    );  }  export default Card; |

**In Parent component How we are passing to child component**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './index.css';  import Card from './Card';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import {robots} from './robots';  ReactDOM.render(  <div>  <Card id={robots[0].id} name={robots[0].name} email={robots[0].email}/>  <Card id={robots[1].id} name={robots[1].name} email={robots[1].email}/>  <Card id={robots[2].id} name={robots[2].name} email={robots[2].email}/>  <Card id={robots[3].id} name={robots[3].name} email={robots[3].email}/>  </div>  , document.getElementById('root')); |

### How to use Destructuring we can do something nice

|  |
| --- |
| import React from 'react';  const Card = (props) => {  const {name, email,id} = props;  return (  <div className="tc bg-light-green dib br3 pad3 ma2 grow bw2 shadow-5">  <img alt="robots" src={`https://robohash.org/${id}?200x200` }/>  <div>  <h2>{name}</h2>  <p>{email}</p>  </div>  </div>    );  }  export default Card; |

Another way is to

|  |
| --- |
| import React from 'react';  const Card = ({name, email,id}) => {  return (  <div className="tc bg-light-green dib br3 pad3 ma2 grow bw2 shadow-5">  <img alt="robots" src={`https://robohash.org/${id}?200x200` }/>  <div>  <h2>{name}</h2>  <p>{email}</p>  </div>  </div>    );  }  export default Card; |

Imp:- As of their was lot of copy and paste in parent component , as we know in One Way Data Flow we always had a parent such as app component that has children.

Using this knowledge we can have one big component that have different children, so that we make component small and reusable.

### How to loop list using map

|  |
| --- |
| import React from 'react';  import Card from './Card';  const Cardlist = ({robots}) => {  const CardComponent = robots.map((user,i) => {  return <Card id={robots[i].id} name={robots[i].name} email={robots[i].email}/>  })  return(  <div>  {CardComponent}  </div>  );  }  export default Cardlist; |

And also remember when we are doing loop we should give some unique key

|  |
| --- |
| import React from 'react';  import Card from './Card';  const Cardlist = ({robots}) => {  const CardComponent = robots.map((user,i) => {  return (  <Card  key={i}  id={robots[i].id}  name={robots[i].name}  email={robots[i].email}  />  );  })  return(  <div>  {CardComponent}  </div>  );  }  export default Cardlist; |

### State

How to communicate child component with another child of different node?

At first child will communicate with parent then parent will communicate with the child with another node.

**State** is simply a description of APP or Simply an Object. An object that describes our application. It can be changed and affect our app.

**Props** are simply thing that come out of state. It’s never changes and it just input we get and never modify them.

**For State declaration we need declare inside constructor and also here we declare inside a class as constructor is required. These state properties can change and affect the App. And usually state is present in the parent component so that it can passes state to different components.**

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import {robots} from './robots';  class App extends Component {  constructor(){  super();  this.state = {  robots: robots,  searchfield: ''  }  }  render(){  return (  <div className="tc">  <h1>Robo Friends</h1>  <SearchBox />  <Cardlist robots = {robots}/>  </div>  )  }  }  export default App; |

### How child communicate to parent

**Using Event we are communicate with parent and using props parent send data to child**

Here every time we type something the onChange will call the searchChange function

|  |
| --- |
| import React from 'react';  const SearchBox = (searchChange) => {  return (  <div className="pa2">  <input  className="pa3 ba b-green bg-lightest-blue"  type="search"  placeholder="Search Robots"  onChange = {searchChange}  />  </div>  );  }  export default SearchBox; |

SearchChange is declare in parent component

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import {robots} from './robots';    class App extends Component {  constructor(){  super();  this.state = {  robots: robots,  searchfield: ''  }  }  onSearchChange(event){  console.log(event.target.value);  }  render(){  return (  <div className="tc">  <h1>Robo Friends</h1>  <SearchBox searchChange = {this.onSearchChange} />  <Cardlist robots = {this.state.robots}/>  </div>  )  }  }  export default App; |

### How to update in React (setState)

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import {robots} from './robots';    class App extends Component {  constructor(){  super();  this.state = {  robots: robots,  searchfield: ''  }  }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value})  const filteredRobots = this.state.robots.filter(robot => {  return robot.name.toLowerCase().includes(this.state.searchfield.toLowerCase);  });  console.log(filteredRobots);  }  render(){  return (  <div className="tc">  <h1>Robo Friends</h1>  <SearchBox searchChange = {this.onSearchChange} />  <Cardlist robots = {this.state.robots}/>  </div>  )  }  }  export default App; |

**Whenever we create a function inside always need to create with arrow function**

|  |
| --- |
| onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  } |

### Life Cycle Hooks

Methods come with react, and they life cycle hooks because it automatically trigger on app when loaded in browser. We can just put the methods into a class component they will automatically get triggered.

1. Mounting :-

When refresh the website the app component gets mounted into the document.getElementById(‘root’)

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

If we check the index.html file the webpage is nothing but a div with id of root.

<div id="root"></div>

When we a mount a component we are replacing the above div adding our app component. Mounting is the start of the app.

Mounting has four function defined, and also order is same as defined below:-

* + [**constructor()**](https://reactjs.org/docs/react-component.html#constructor)
  + [static getDerivedStateFromProps()](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
  + [**render()**](https://reactjs.org/docs/react-component.html#render)
  + [**componentDidMount()**](https://reactjs.org/docs/react-component.html#componentdidmount)

1. Updating

Whenever a component changes. It has below function defined:

* + [**static getDerivedStateFromProps()**](https://reactjs.org/docs/react-component.html#static-getderivedstatefromprops)
  + [**shouldComponentUpdate()**](https://reactjs.org/docs/react-component.html#shouldcomponentupdate)
  + [render()](https://reactjs.org/docs/react-component.html#render)
  + [**getSnapshotBeforeUpdate()**](https://reactjs.org/docs/react-component.html#getsnapshotbeforeupdate)
  + [componentDidUpdate()](https://reactjs.org/docs/react-component.html#componentdidupdate)

1. UnMounting

Whenever a component is removed from a DOM or page.

* + [componentWillUnmount()](https://reactjs.org/docs/react-component.html#componentwillunmount)

### Fetching from JSON placeholder (fetch)

Which we will discuss more briefly in next section. And also fetch is part of window object it comes with browser, it is tool to make request to server.

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from './SearchBox';  import Cardlist from './Cardlist';  import './App.css';    class App extends Component {  constructor(){  super();  this.state = {  robots: [],  searchfield: ''  }  }  componentDidMount(){  fetch('https://jsonplaceholder.typicode.com/users')  .then(response =>{  return response.json();  })  .then(users =>{  this.setState({robots: users});  })    }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  }  render(){    const filteredRobots = this.state.robots.filter(robot => {  return robot.name.toLowerCase().includes(this.state.searchfield.toLowerCase());  });  return (  <div className="tc">  <h1 className="f1">Robo Friends</h1>  {console.log(filteredRobots)}  <SearchBox searchChange = {this.onSearchChange} />  <Cardlist robots = {filteredRobots}/>  </div>  )  }  }  export default App; |

### Children

React can use to create component for functionality also like scrolling

As of now we see only self closing custom component. But we didn’t see component that wraps another component, then how we are going talk with the wrap component. There come the children, every props has children define.

|  |
| --- |
| class App extends Component {  constructor(){  super();  this.state = {  robots: [],  searchfield: ''  }  }  componentDidMount(){  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(users => this.setState({robots: users}));    }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  }  render(){    const filteredRobots = this.state.robots.filter(robot => {  return robot.name.toLowerCase().includes(this.state.searchfield.toLowerCase());  });  if(this.state.robots.length === 0){  return <h1>Loading</h1>  }  else {  return (  <div className="tc">  <h1 className="f1">Robo Friends</h1>  {console.log(filteredRobots)}  <SearchBox searchChange = {this.onSearchChange} />  <Scroll>  <Cardlist robots = {filteredRobots}/>  </Scroll>  </div>  );  }  }  }  export default App; |

In scroll component

|  |
| --- |
| import React from 'react';  const Scroll = (props) => {  return (  <div style={{overflowY:'scroll', border:'1px solid black',height:'500px'}}>  {props.children}  </div>  )  };  export default Scroll; |

### Inline style in the custom component inside JSX

Use {{}}

|  |
| --- |
| import React from 'react';  const Scroll = (props) => {  return (  <div style={{overflowY:'scroll', border:'1px solid black',height:'500px'}}>  {props.children}  </div>  )  };  export default Scroll; |

### Pure component vs smart component

Component without state is termed as pure component.

Smart component is the component where state, life cycle hooks present that has the class syntax.

### Folder (Components and Containers)

All pure components will be present in components folder

All smart components will be present in containers folder.

### Keeping the react projects up to date

Always remember to update the package.json file with the respective version.

npm audit fix :- it will try to audit and fix the some version issue

npm audit:- it will list out all the vulnerabilities

npm audit fix --force – it will update all the package with latest version

### Error Boundary

Error Boundary we need react version 16 or higher. Using **componentDidCatch()** predefined life cycle hooks method.

Error Boundary when we put our code in production, when we don’t have the details error in development mode.

**Create the component for Error to catch**

componentDidCatch :- Act as a try catch block in JavaScript

|  |
| --- |
| import React, { Component } from 'react';  class ErrorBoundary extends Component{  constructor(props){  super(props);  this.state = {  hasError: false  }  }  componentDidCatch(error,info){  this.setState({hasError:true});  }    render(){  if(this.state.hasError){  return <h1>Error Occured.Please contact support team</h1>  }  return this.props.children  }  }  export default ErrorBoundary; |

**How to use the ErrorBoundary component in the other component to catch error**

**In app.js**

Here below if any error in cardlist component it catches error and display error message

|  |
| --- |
| import React, {Component} from 'react';  import SearchBox from '../Components/SearchBox';  import Cardlist from '../Components/Cardlist';  import Scroll from '../Components/Scroll';  import './App.css';  import ErrorBoundary from '../Components/ErrrorBoundary';  class App extends Component {  constructor(){  super();  this.state = {  robots: [],  searchfield: ''  }  }  componentDidMount(){  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(users => this.setState({robots: users}));    }  onSearchChange =(event) => {  this.setState({searchfield: event.target.value});  }  render(){  const {robots,searchfield} = this.state;  const filteredRobots = robots.filter(robot => {  return robot.name.toLowerCase().includes(searchfield.toLowerCase());  });  return !robots.length ?  <h1>Loading</h1> :  (  <div className="tc">  <h1 className="f1">Robo Friends</h1>  {console.log(filteredRobots)}  <SearchBox searchChange = {this.onSearchChange} />  <Scroll>  <ErrorBoundary>  <Cardlist robots = {filteredRobots}/>  </ErrorBoundary>  </Scroll>  </div>  );  }  }  export default App; |

### How to fetch data from backend

Here ComponentDidMount() is the life cycle hooks in React

|  |
| --- |
| componentDidMount() {  fetch('http://localhost:3008')  .then(response => response.json())  .then(data => console.log(data));  } |

### Submitting a post request to backend with Request headers and request body

|  |
| --- |
| onSubmitSignIn = () => {  fetch('http://localhost:3008/signin',{  method:'post',  headers: {'Content-Type':'application/json'},  body: JSON.stringify({  email: this.state.signInEmail,  password: this.state.signInPassword  })  })  .then(response => response.json())  .then(data =>{  if(data === 'success'){  this.props.onRouteChange('home')  }  });  } |

### Deploying our React App

Via GitHub pages

<https://create-react-app.dev/docs/deployment/#github-pages-https-pagesgithubcom>

### State Management

State description how the app should look like. Child component update the parent component using event.

Think State as memory, an app needs to remember things in order to work otherwise we will have simple html webpages.

|  |
| --- |
|  |

When the app is grows bigger and bigger it very tough to manage the state. So Redux solve this problem for us.

In redux we keep the state in a store, the store is simply the state but in one massive object. The one simple object that describe how our app should look and all we do it just pass down the state to whichever component needs it as props

|  |
| --- |
|  |

### Why Redux?

React is very good in view side but not necessary in managing state.

But using Redux solves the problem. And Redux is very good in the below points:-

1. Good for managing large state.
2. Useful for sharing data between containers.
3. Predictable state management using the 3 principles :-
   * + Single Source of truth.
     + State is read only.
     + Change using pure functions

|  |
| --- |
|  |

**With Redux all the actions goes to one reducer, then the reducer goes to store and then the store determines how the app should look like.**

**Redux uses the architectural pattern of Flux pattern, which is unidirectional data flow.**

**Store is nothing but the state**

|  |
| --- |
|  |

Before that we have something called MVC. The problem is MVC we have controller that update the different piece of model and model can change the view and that view can trigger the another change and so on and so forth.

|  |
| --- |
|  |

Redux at the end of the data is same as this.State. With Redux we can replace all this.state from react and happens all in redux library.

|  |
| --- |
|  |

But one caveat, we would technically have redux with the state but also still keep a little bit of react state in a component. Redux does not completely replace this.state

|  |
| --- |
|  |

### Installing Redux

npm install redux

We need another package which we tell react to use redux, because redux can be used in other libraries.

npm install react-redux

Here only connection will be done with containers or the smart components and with redux store and also component will component with the store and viceversa.

|  |
| --- |
|  |

### Redux Actions and Reducers

|  |
| --- |
|  |

**Creating an action**

**actions.js**

Below action is going to take text which is what the user input and it’s going to return an object type and payload. Payload is the common name that is used in redux, payload is we are sending whatever data is needed to go on to the Reducer.

|  |
| --- |
| import {CHANGE\_SEARCH\_FIELD} from './constants.js';  export const setSearchField = (text) => ({  type: CHANGE\_SEARCH\_FIELD,  payload: text  }) |

**Create a constants.js file for all the type declaration instead of hardcode.**

|  |
| --- |
| export const CHANGE\_SEARCH\_FIELD = 'CHANGE\_SEARCH\_FIELD'; |

**Creating the Reducer**

Reducer is a big function just read the action and spits out the state.

**reducer.js**

Below it is simply saying we receive a action called CHANGE\_SEARCH\_FIELD, if that the case then return the new state with action.payload

|  |
| --- |
| import {CHANGE\_SEARCH\_FIELD} from './constants.js';  const initialState = {  searchField: ''  }  export const searchRobots = (state=initialState, action={}) => {  switch(action.type){  case: CHANGE\_SEARCH\_FIELD:  return Object.assign({}, state, {searchField: action.payload });  default:  return state;  }  } |

**Instead of using Object.assign we can use the spread operator (…)**

|  |
| --- |
| export const searchRobots = (state=initialState, action={}) => {  switch(action.type){  case: CHANGE\_SEARCH\_FIELD:  return { ... state, {searchField: action.payload }}  }  } |

### Redux Store and Provider

Let connect redux to react application.

For that import provider and connect from ‘react-redux’.

**In this section we will discuss about provider and in next section about connect.**

import {provider, connect} from 'react-redux';

**Let create the store**

Store is the source of all i.e. it’s a big object that describes the state of our app, so that react can render it and make changes and display to the user.

**Import the createStore from redux and create the store which will get the reducer**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {provider, connect} from 'react-redux';  import {createStore} from 'redux';  import './index.css';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import App from './Containers/App';  import { searchRobots } from './reducers';  const store = createStore(searchRobots)  ReactDOM.render(<App />, document.getElementById('root')); |

**How to pass the store to the component tree?**

We don’t want to pass the store all the way down to the component tree over and over again. React-Redux has given a component (provider). Now the provider component is going to take care the passing down the store to all the components in the component tree from the app.

|  |
| --- |
| ReactDOM.render(  <Provider store="store">  <App/>  </Provider>, document.getElementById('root')); |

### Redux Connect

In containers component only we will do the redux connect. And connect is higher order function means function that returns another function.

**Step 1:- Import the action in the smart component app.js**

|  |
| --- |
| import { setSearchField } from '../actions' |

**Step 2:- Import the connect method from react-redux.**

import {connect} from 'react-redux'

**Step 3:- Use the connect method in very bottom where we do export default. And connect is higher order function means function that returns another function.**

**And connect two parameters** (mapStateToProps, mapDispatchToProps)

export default connect(mapStateToProps, mapDispatchToProps)(App);

Here basically we connected the app component and said to subscribe to any state changes in the redux store and the App component knows that there is a redux store somewhere and any time there is changes to it, it might be interested to it.

Now we need to tell it what we are interested in i.e. what state should I listen and what action/dispatch should it listen

**Step 4:- Define the parameter mapStateToProps**

**This function is going to receives a state and it’s going to returns an object.**

**The object is basically searchField from reducer with the state of the searchRobots, which is going to be use as a prop by the app component. Because in index.js we can have created the store with searchRobots reducer.**

|  |
| --- |
| const mapStateToProps = state => {  return {  searchfield: state.searchRobots.searchfield  }  } |

**Step 5:- Define the parameter mapDispatchToProps**

**It’s is going to receive a dispatch, dispatch (basically what triggers the action). To dispatch into the reducer, so this dispatch can now be used to send action**

|  |
| --- |
| const mapDispatchToProps = (dispatch) => {  return {  onSearchChange: (event) => dispatch(setSearchField(event.target.value))  }  } |

**mapStateToProps :- Is used to what piece of state app need to listen to and send down as props.**

**mapDispatchToProps:- What props app need to listen to that are actions, that needs to get dispatch.**

|  |
| --- |
| import React from 'react'  import {connect} from 'react-redux'  import CardList from '../Components/CardList';  import SearchBox from '../Components/SearchBox';  import Scroll from '../Components/Scroll';  import ErrorBoundary from '../Components/ErrorBoundary';  import { setSearchField } from '../actions'  const mapStateToProps = state => {  return {  searchField: state.searchField  }  }  const mapDispatchToProps = (dispatch) => {  return {  onSearchChange: (event) => dispatch(setSearchField(event.target.value))  }  }  class App extends React.Component {  constructor() {  super();  this.state = {  robots: []  }  }  componentDidMount() {  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(users => this.setState({robots:users}));  }  render() {  const {robots} = this.state;  const {searchField,onSearchChange} = this.props;  const filteredRobots = robots.filter(robot => {  return robot.name.toLowerCase().includes(searchField.toLowerCase())  })  return !robots.length ?  <h1>Loading..</h1> :  (  <div className="tc">  <h1 className="f2">RoboFriends</h1>  <SearchBox onSearchChange = {onSearchChange} />  <Scroll>  <ErrorBoundary>  <CardList robots={filteredRobots}/>  </ErrorBoundary>  </Scroll>  </div>  );  }  }  //subscribe to any state changes in the redux store  export default connect(mapStateToProps, mapDispatchToProps)(App); |

### Redux Middleware

It simply listens for action; depending on the action Middleware can modify the action or trigger another action.

|  |
| --- |
|  |

### Redux logger

It is middleware which we can use for logging purpose in console.

npm install redux-logger

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {Provider} from 'react-redux';  import {createStore, applyMiddleware } from 'redux';  import {createLogger} from 'redux-logger';  import './index.css';  import App from './Containers/App';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import { searchRobots } from './reducers';  const logger = createLogger();  const store = createStore(searchRobots, applyMiddleware(logger));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>, document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

### Redux Async Actions (redux thunk)

Using middle ware we can do this. In Redux we have the middle ware redux thunk to handle asynchronous actions like Ajax calls.

It’s a middle ware that provides a get state and dispatch functions that are passed on you are able to handle something called side effects like Ajax calls. There are other options also but redux thunk is best and simplest to start off with

npm install --save redux-thunk

**step 1:- Import the middleware and apply the middleware**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {Provider} from 'react-redux';  import {createStore, applyMiddleware } from 'redux';  import {createLogger} from 'redux-logger';  import thunkMiddleware from 'redux-thunk';  import './index.css';  import App from './Containers/App';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import { searchRobots } from './reducers';  const logger = createLogger();  const store = createStore(searchRobots, applyMiddleware(thunkMiddleware,logger));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>, document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

**Step 2:- Implement constants, actions, reducers, provider and the connect for the async action.**

**Step a: - Create the constants**

|  |
| --- |
| export const REQUEST\_ROBOTS\_PENDING = 'REQUEST\_ROBOTS\_PENDING';  export const REQUEST\_ROBOTS\_SUCCESS = 'REQUEST\_ROBOTS\_SUCCESS';  export const REQUEST\_ROBOTS\_FAILED = 'REQUEST\_ROBOTS\_FAILED'; |

**Step b: - Create the actions using the above constants and implement the functionality of asynchronous operation**.

|  |
| --- |
| import {  CHANGE\_SEARCHFIELD,  REQUEST\_ROBOTS\_SUCCESS,  REQUEST\_ROBOTS\_PENDING,  REQUEST\_ROBOTS\_FAILED  } from './constants'  export const setSearchField = (text) => ({  type: CHANGE\_SEARCHFIELD,  payload: text  })  export const requestRobots = (dispatch) => {  dispatch({type: REQUEST\_ROBOTS\_PENDING});  fetch('https://jsonplaceholder.typicode.com/users')  .then(response => response.json())  .then(data => dispatch({type: REQUEST\_ROBOTS\_SUCCESS, payload: data}))  .catch(error => dispatch({type: REQUEST\_ROBOTS\_FAILED, payload: error}))  } |

**Step c: - Create the reducers using the above constants**

|  |
| --- |
| import {  CHANGE\_SEARCHFIELD,  REQUEST\_ROBOTS\_SUCCESS,  REQUEST\_ROBOTS\_PENDING,  REQUEST\_ROBOTS\_FAILED  } from './constants'  const initialStateSearch = {  searchField: ''  }  export const searchRobots = (state = initialStateSearch, action={}) => {  switch(action.type) {  case CHANGE\_SEARCHFIELD:  return Object.assign({}, state ,{searchField:action.payload});  default:  return state;  }  }  const initialStateRobots = {  isPending: false,  robots: [],  error: ''  }  export const requestRobots = (state = initialStateRobots, action={}) => {  switch(action.type) {  case REQUEST\_ROBOTS\_PENDING:  return Object.assign({}, state, {isPending: true});  case REQUEST\_ROBOTS\_SUCCESS:  return Object.assign({}, state, {robots: action.payload, isPending: false});  case REQUEST\_ROBOTS\_FAILED:  return Object.assign({}, state, { error: action.payload, isPending:false});  default:  return state;  }  } |

**Step d: Import the reducer in index.js and also add both the reducer (as we have two reducer now) in the store using combinedReducers (which combines all the reducer into a root).**

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import {Provider} from 'react-redux';  import {createStore, applyMiddleware, combineReducers } from 'redux';  import {createLogger} from 'redux-logger';  import thunkMiddleware from 'redux-thunk';  import './index.css';  import App from './Containers/App';  import \* as serviceWorker from './serviceWorker';  import 'tachyons';  import { searchRobots, requestRobots } from './reducers';  const logger = createLogger();  const rootReducer = combineReducers({searchRobots,requestRobots});  const store = createStore(rootReducer, applyMiddleware(thunkMiddleware,logger));  ReactDOM.render(  <Provider store={store}>  <App />  </Provider>, document.getElementById('root'));  // If you want your app to work offline and load faster, you can change  // unregister() to register() below. Note this comes with some pitfalls.  // Learn more about service workers: https://bit.ly/CRA-PWA  serviceWorker.unregister(); |

**Step e: Implement in container instead of directly use the state, use it from the redux.**

|  |
| --- |
| import React from 'react'  import {connect} from 'react-redux'  import CardList from '../Components/CardList';  import SearchBox from '../Components/SearchBox';  import Scroll from '../Components/Scroll';  import ErrorBoundary from '../Components/ErrorBoundary';  import { setSearchField, requestRobots } from '../actions'  const mapStateToProps = state => {  return {  searchField: state.searchRobots.searchField,  robots: state.requestRobots.robots,  isPending: state.requestRobots.isPending,  error: state.requestRobots.error  }  }  const mapDispatchToProps = (dispatch) => {  return {  onSearchChange: (event) => dispatch(setSearchField(event.target.value)),  onRequestRobots: () => dispatch(requestRobots())  }  }  class App extends React.Component {  componentDidMount() {  this.props.onRequestRobots();  }  render() {  const {searchField,onSearchChange, robots, isPending} = this.props;  const filteredRobots = robots.filter(robot => {  return robot.name.toLowerCase().includes(searchField.toLowerCase())  })  return isPending ?  <h1>Loading..</h1> :  (  <div className="tc">  <h1 className="f2">RoboFriends</h1>  <SearchBox onSearchChange = {onSearchChange} />  <Scroll>  <ErrorBoundary>  <CardList robots={filteredRobots}/>  </ErrorBoundary>  </Scroll>  </div>  );  }  }  //subscribe to any state changes in the redux store  export default connect(mapStateToProps, mapDispatchToProps)(App); |

### Tools using in React

* + React Router: - Changing pages or router of the url
  + Glamorous:- for styling.

### Reactstrap:-

It is library used in react to use bootstrap component. But be careful to use the third party library as it might some other package in our react app like Tachyons.

<https://reactstrap.github.io/>

Step 1: - npm install reactstrap

Step 2: - Install the bootstrap as bootstrap CSS does not comes with reactstrap.

npm install bootstrap

Step 3: - Import the bootstrap css in the index.js

import 'bootstrap/dist/css/bootstrap.min.css';

### React Modal

<https://reactjs.org/docs/portals.html>

The way Modal works is that we want to make sure that having another react DOM render that is outside of the app, by using react portals we can transfer props between app and modal.

Step 1: - Create another html element we can bind to the Modal component in index.html

|  |
| --- |
| <body>  <noscript>You need to enable JavaScript to run this app.</noscript>  <div id="root"></div>  <div id="modal-root"></div>  </body>  </html> |

Step 2: - Create the Modal Component

|  |
| --- |
| import React from 'react';  import ReactDOM from 'react-dom';  import './Modal.css';  const modalRoot = document.getElementById('modal-root');  class Modal extends React.Component{  constructor(props){  super(props);  this.el = document.createElement('div');  }  componentDidMount() {  modalRoot.appendChild(this.el);  }  componentWillUnmount(){  modalRoot.removeChild(this.el);  }  render(){  return ReactDOM.createPortal(this.props.children,this.el);  }  }  export default Modal; |

## HTTP/HTTPS

HTTP is a protocol which allows the fetching of resources such as Html document. It is foundation of any data exchange on the web.

HTTP we can think as a common language that a client and server use to communicate.

Clients and servers can communicate by exchanging individual messages, the messages send by the client usually a web browser are called **request** and messages send by the server usually called **response.**

HTTP Verbs:

it can send data in two ways :- 1) Query String and 2) Body of the request

* GET
* POST –
* PUT
* DELETE

HTTP Messages:-

<https://www.w3schools.com/tags/ref_httpmessages.asp>

**HTTPS(Hyper text transfer protocol secure)**:- The communicate between browser and server are encrypted. It used the transport layer security(TLS) or secured socket layer(SSL). If we ever sending sensitive information like password make sure to have https.

## JSON (JavaScript Object Notation)

JSON is a syntax of storing and exchanging data. It is text, written with JavaScript Object Notation.

We can convert any JavaScript into JSON and it to the server and also we can convert JSON receive from the server into JavaScript Objects.

Another way is XML. Both JSON and XML can be used to receive data from the web server.

How to change JSON to JavaScript and viceversa?

* Var obj = JSON.Parse(‘{“name”:”Chiranjit”, “age”:”27”}; // it will convert to JavaScript object
* Var myJSON = JSON.stringify(obj); // it will convert to JSON\

Originally, the only way to submit some form data to a server was through the <form> tag in HTML. As we have learned, it can do a POST or a GET request. With JSON you can now grab the contents of the <input> in a form and submit those with JSON instead of as a form data. You can now submit to the server whenever you want without it necessarily being a <form>, through AJAX.. What is AJAX you might say?

## Ajax

It allows reading from web server after the page is loaded and update a webpage without reloading the page.

Page update will be quite quicker as we don’t need to page to refresh. Also less data is downloaded on each update less waste bandwidth.

Fetch(‘url’).then(response => {

//do something

})

## Promises

A promise is an object that may produce a single value sometime in the future. Either a resolved value, or a reason that it’s not resolved (rejected).

Before promise we have call back like event listener. Eg:- el.addEventListener(“click”,submitForm);

|  |
| --- |
|  |

Promises are new in ES6 and little powerful than callback function. The above callback pyramid using promises.

Promises are good for asynchronous programming

|  |
| --- |
|  |

Eg of promise

|  |
| --- |
| const promise = new Promise((resolve,reject)=>{  if(true){  resolve('Stuff worked');  }else {  reject('Error, it broke');  }  })  promise  .then(result => result + '!')  .then(result2 => {  console.log(result2);  }) |

Error handling inside promise

|  |
| --- |
| promise  .then(result => result + '!')  .then(result2 => {  throw Error  console.log(result2);  })  .catch(() => console.log('error!')); |

**Promise.all([])**

Is used to run multiple promises at a time.

|  |
| --- |
| const promise1 = new Promise((resolve,reject)=>{  setTimeout(resolve,100, 'Hii');  });  const promise2 = new Promise((resolve,reject)=>{  setTimeout(resolve,1000,'Hello');  })  const promise3 = new Promise((resolve,reject)=>{  setTimeout(resolve, 5000, 'JaiHo');  })  Promise.all([promise1,promise2,promise3])  .then(values =>{  console.log(values);  }); |

**Real time example of promises**

|  |
| --- |
| const urls = [  "https://jsonplaceholder.typicode.com/users",  "https://jsonplaceholder.typicode.com/albums",  "https://jsonplaceholder.typicode.com/posts"  ];  Promise.all(urls.map(url =>{  return fetch(url).then(resp => resp.json())  })).then(results => {  console.log(results);  }).catch((ex) => console.log(ex)); |

## Async Await

It is same as promises only the syntax is change.

|  |
| --- |
|  |

Another example

|  |
| --- |
| //via promise  fetch('https://jsonplaceholder.typicode.com/users')  .then(resp => resp.json())  .then(result =>{  console.log(result);  });  //via async function  async function fetchUser() {  const resp = await fetch('https://jsonplaceholder.typicode.com/users')  const data = await resp.json()  console.log(data);  } |

## ES9 – Async

### Finally

If we want to run a code no matter what after a promise.

|  |
| --- |
| const urls = [  'https://swapi.co/api/people/1',  'https://swapi.co/api/people/2',  'https://swapi.co/api/people/3',  'https://swapi.co/api/people/4'  ]  Promise.all(urls.map(url=>{  return fetch(url).then(resp =>resp.json())  })).then(results=>{  console.log(results[0]);  console.log(results[1]);  console.log(results[2]);  console.log(results[3]);  })  .catch((ex)=> console.log(ex))  .finally(() => console.log('it will regardless of any error or success')); |

### for await

It allows to loop through async await

|  |
| --- |
| //via async function  const urls = [  'https://swapi.co/api/people/1',  'https://swapi.co/api/people/2',  'https://swapi.co/api/people/3',  'https://swapi.co/api/people/4'  ]  // without using for await  const getData = async function(){  try{  const [people1, people2, peopel3,people4] = await Promise.all(urls.map(url=>{  const response = await fetch(url);  return response.json();  }))  console.log(people1);  console.log(people2);  console.log(peopel3);  console.log(people4);  }  catch(ex){  console.log(ex);  }  }  //using for await  const getData2 = async function(){  const arrayOfPromises = urls.map(url => fetch(url));  for await(let request of arrayOfPromises){  const data = await request.json();  console.log(data);  }  } |

## Backend Basic

|  |
| --- |
|  |

## APIs (Application Programming Interface)

Is a messenger that takes a request and tells the system what to do? It applies a way the machine communicates to share information. So api is free and some are chargeable.

<https://api.github.com/users/chiranjit9955>

## Databases

It is the collection of data. DBMS it is the collection of programs which allow us to access databases and work with data and also allow us control access to the database user.

### Relational Databases

Relational consists of two or more tables with columns and rows. The relation between tables and fields is called a schema. In a relationship database the schema must be clearly define before any information can be added.

|  |
| --- |
|  |

### How to communicate relational databases with backend api

Using SQL we able to communicate with databse.

### NoSQL / Non-Relational Databases

Build an application without define a schema. If our data requirement is not clear on outside of our project, maybe we have massive amount of unstructured data.

MongoDB is something document oriented; it stores information as document like instead of table we will have each user as a document.

### How to communicate Non-Relational databases with backend api

For Mongo DB – MongoDB query language.

### Difference between Relational and Non-Relational Databases

|  |
| --- |
|  |

### PostgreSQL

Command : psql -U postgres to connect to the postgreSQL

Command : \q to disconnect from postgresql

Commnad: \c <databasename> to redirect to particular database

Command: \d to get all the objects in the database

### Connect PostgreSQL to Server

Using Knex.js

<http://knexjs.org/>

* Install the below two package in the project
* npm install knex
* npm install pg

|  |
| --- |
| const knex = require("knex");  const db =knex({  client: 'pg',  connection: {  host : '127.0.0.1',  user : 'postgres',  password : '',  database : 'face-recognition-brain'  }  });  db.select('\*').from('users').then(data =>{  console.log(data);  }); |

Please check the <http://knexjs.org/>

Just one example for insert

|  |
| --- |
| app.post('/register',(req,res)=>{  const {email,password, name} = req.body;  db('users')  .returning('\*')  .insert({  email:email,  name: name,  joined: new Date()  })  .then(user =>{  res.json(user[0]);  })  .catch(err => res.status(400).json('unable to register'));  }) |

Select example

|  |
| --- |
| app.get('/profile/:id',(req,res)=>{  const {id} = req.params;  db.select('\*').from('users')  .where({  id:id  })  .then(user =>{  if(user.length){  res.json(user[0]);  } else {  res.status(400).json('not found')  }    })  .catch(err => res.status(400).json('error getting user'))  }) |

Transaction: - where we insert in multiple table at once if any one table fail to insert we rollback everything.

## Security

* + Always remember to do the validation from client as well as in server (e.g. empty input field etc)
  + All api key will not be present in the browser instead we can move to server side.

## Deployment

We can use Heroku for deployment

<https://devcenter.heroku.com/articles/git>

|  |
| --- |
|  |

### After installation deploy via git

1. heroku create – to create an app in heroku
2. heroku open – to open the app in browser
3. git remote –v – to confirm that a remote named heroku has been set for your app
4. git push heroku master – Now instead of pushing to git we need to push to heroku
5. heroku logs --tail - To get the logs in heroku server

### Setup ProstgreSQL in Heroku app

<https://devcenter.heroku.com/articles/heroku-postgresql>

1. heroku addons - command to determine whether your app already has Heroku Postgres provisioned
2. heroku pg:info - To see if your database is running on legacy infrastructure
3. heroku pg:psql - To establish a psql session with your remote database
4. heroku config – To check the database heroku is using

|  |
| --- |
| const db =knex({  client: 'pg',  /\* connection: {  host : '127.0.0.1',  user : 'postgres',  password : 'test',  database : 'face-recognition-brain'  } \*/  connection: {  connectionString: process.env.DATABASE\_URL,  ssl: true  }  }); |

### After creating an app from Heroku website

<https://dashboard.heroku.com/apps/facerecognition-chiranjit/deploy/heroku-git>

1. heroku login - If you haven't already, log in to your Heroku account and follow the prompts to create a new SSH public key.
2. heroku git:remote -a <app-name> - Initialize a git repository in a new or existing directory
3. git add . – If you don’t change anything also need to run the command.
4. git commit –m”commit message” - If you don’t change anything also need to run the command
5. git push heroku master

**We can also create environment variable in Heroku like api key and use then in the code. Instead of hardcoded in the code.**

## Resources

### For all new features of ES :-

<https://github.com/daumann/ECMAScript-new-features-list>

### React tilt

<https://www.npmjs.com/package/react-tilt>

### CSS3 Pattern example

<https://leaverou.github.io/css3patterns/>

### Paricles js to look the background really nice

<https://vincentgarreau.com/particles.js/>

### For react

<https://www.npmjs.com/package/react-particles-js>

### Reactstrap – to add bootstrap component in react app.

<https://reactstrap.github.io/>

### Deploying website via Github pages

<https://create-react-app.dev/docs/deployment/#github-pages-https-pagesgithubcom>

### Image Recognition API

<https://www.clarifai.com/>

### JavaScript in Robotics

<http://johnny-five.io/>

### State of JavaScript for both Frontend and Backend

<https://2018.stateofjs.com/back-end-frameworks/overview/>

### Express.js

<https://expressjs.com/>

### Bcrypt

<https://www.npmjs.com/package/bcrypt-nodejs>

### Universal Database Tool

Free multi-platform database tool for developers, database administrators, analysts and all people who need to work with databases. Supports all popular databases: MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, MS Access, Teradata, Firebird, Apache Hive, Phoenix, Presto, etc.

<https://dbeaver.io/>

### Database Tutorial

<https://www.khanacademy.org/computing/computer-programming/sql#concept-intro>

<https://www.codecademy.com/learn/learn-sql>

### Knes.js ( connecting to database from sever)

<http://knexjs.org/>

### pg-promise ( connecting to database from sever)

<https://github.com/vitaly-t/pg-promise>

### Heroku (for deployment)

<https://devcenter.heroku.com/articles/git>

### Podcast of Software Developer

<https://softwareengineeringdaily.com/category/podcast/>

### Advanced JavaScript course

<https://www.pluralsight.com/courses/javascript-development-environment>