## SSH

### Introduction

SSH( Secure Shell) is a protocol to communicate machine with one another over the internet. It allows users to share files as well as control and modify remote computers over the internet and also everything is encrypted.

|  |
| --- |
|  |

### Use of SSH

1. SSH connection in GitHub.
2. Remote connection in another device.

Command : ssh root@<machinename || ipaddress>

### Techniques used in SSH

There are three technique used in SSH. These technique used when we works with Https, block chain etc.

#### Symmetrical Encryption.

It uses one secret key for both encryption and decryption. Disadvantage is if anyone has the key can decrypt the message.

The disadvantage can be solved by Key Exchange Algorithm (Secure way exchange the key with our bad person intercepting it.

#### Asymmetrical Encryption.

It uses two separate keys for encryption and decryption. And it will have public and private key for both the machine which will be communicate. Public key can be shared to outside but the private key should not be share to anybody.

Message that encrypted by machine public key can only be decrypted by the same machine private key.

**Difiie Hellman key exchange algorithm**

**It uses the information of public and private key information of two machines two generate without exchanging the keys. Each machine on its computer can generate asymmetrical key**

#### Hashing

It is another form of cryptography used in secure shell connection. They are never meant to decrypt anything, it simply generate a unique value of a fixed length for each input that it gets (but for the generate key we can’t get the input string its one way)

Using hash function each message that it transmitted must contain a mac and this mac is a hashed generated from the symmetric key.

## Performance

### 3 Keys to increase the performance

1. Front End Side
2. Transfer of file over the wire (network latency)
3. Back End Side

|  |
| --- |
|  |

### Network Performance

#### Minimize Files:-

##### Minimize Text (CSS, Html, JavaScript) – Very easy to do using Web-Pack while build the project files.

##### Minimize Images

Image File Formats:-

JPG: - Usually used for photos, images and things with many colors. And also we cannot change the background of the image with this format.

GIF: - Usually used for small animation. And it usually limit the color counts we can use in it (2- 256) and reducing the color leads to file saving.

PNG: - Usually used in logo and limits the color counts we can use and it tends to lot smaller in size than JPG. We can add transparency to that (means changing the background color).

SVG:- It’s an Extensible Markup Language (XML)-based vector image format for two-dimensional graphics with support for interactivity and animation. And also we can customize it using CSS.

**All remember to pick the right format of images and compress them as much as we can without minimize the quality.**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Category | Palette | Use for |
| JPG | Lossy | Millions of colors | Still Images Photography |
| GIF | Lossless | Maximum 256 colors | Simple animations Graphics with flat colors Graphics without gradients |
| PNG-8 | Lossless | Maximum 256 colors | Similar to GIF Better transparency but no animation Great for icons |
| PNG-24 | Lossless | Unlimited colors | Similar to PNG-8 Handles still images and transparency |
| SVG | Vector/lossless | Unlimited colors | Graphics/logos for web Retina/high-dpi screens |

|  |
| --- |
|  |

**Display Different sized images for different backgrounds:** Using **@media** we can do that

<https://gist.github.com/bartholomej/8415655>

|  |
| --- |
|  |

**Use CDN like imigx:** It says give us all the images no matter how big they are or un-optimized, just upload on to their website and they will give a URL, which we can plugin in our website. And they also do it via CDN (content delivery network)

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

**Remove image metadata: -** Meta data means which device it clicked, when it clicked etc. Use the below url to remove it.

<https://www.verexif.com/en/>

#### Minimize Deliveries

Reducing the download frequency, reducing the no of component a page requires proportionally reduces the no of http request has to make.

Think like that instead of using bootstrap we can use flexbox or CSS grid as for bootstrap we need to download the massive file all the time.

Same apply for JavaScript libraries. So if we want to use libraries always pick the library which is light weight.

**Http protocol has limit to the size of the file and also no of requests it handle**.

So always remember to minimize the file and also limit the trips that the http request makes. So perhaps we can combine the CSS files or JavaScript files.

### Front End

**Some extra information how front end work**: - when a browser sends a URL to a server, the server sends a response as HTML file. As soon as HTML arrives in browser it start creating the DOM, and when the browser parses and read the HTML it incrementally generates the tree model of the html tag we need to build of the website.

While reading the HTML page it encounters a style link to grab the CSS file and then started working on the DOM. Once it grabs all the CSS files it will create the CSSOM (CSS object model).

Then it grabs all the JavaScript file, and then this JS files read by the browser and execute any changes that it might want on to the DOM and the CSSOM.

Once all the things done, the browser combines the DOM and the COM into a render tree. So it knows exactly what to render on the page and now the browser uses the render tree to figure out the layout.

#### Critical Render Path

|  |
| --- |
|  |

How to optimize the steps?

##### HTML

Always render the CSS as soon as possible, because JS needs to runs after the CSS and HTML parsing is finished. **That’s means always mentioned the JS files at the button instead of declaring to head. But some cases we need to put the JS files at the top (for example google analytics which will be script tag, when we want to execute first).**

|  |
| --- |
|  |

##### CSS

CSS is render blocking as for creating render tree we need to wait for CSS. So for that CSS should be light weight, we can do it following below steps:-

|  |
| --- |
|  |

* + 1. **Only load whenever it is needed.**

**Sometime we can do using internal CSS and inline CSS. But it has limitation like we are limited one html page or one element**

* + 1. **Above the fold loading** :- eg **If we want to load css after the page load**

|  |
| --- |
| <script type="text/javascript">  const loadStyleSheet = src => {  if(document.createStylesheet) {  document.createStylesheet(src);  } else {  const stylesheet = document.createElement('link');  stylesheet.href = src;  stylesheet.type = 'text/css';  stylesheet.rel ='stylesheet';  document.getElementsByTagName('head')[0].appendChild(stylesheet);  }  window.onload = () => {  loadStyleSheet('./style3.css')  }  }  </script> |

* + 1. **Media attributes**

|  |
| --- |
| <link rel="stylesheet" href="./script2.js" media="only screen and (min-width:500px)"> |

* + 1. **Less specificity**

As for the first one will take more byte as we send more information with respect to second one

|  |
| --- |
|  |

##### JavaScript

|  |
| --- |
|  |

1. **Load Scripts asynchronously**

By using async we can tell the browser, go ahead and download the JavaScript file with another thread. **But remember we should add them that don’t affect the DOM or CSSOM. Like google analytics script or tracking script we can use.**

1. **Defer loading of Script**

Defer is same as async while will not block loading of the html page, but however. And it will execute after the html is parsed and will execute in order of appearance.

**If a core functionality require JavaScript then async is best and if not require then use defer.**

|  |
| --- |
|  |

1. **Minimize DOM manipulation**

For the below example first it will print “this is script 1 ” as there is no DOM manipulation just printing but the two will change after the page is render as it has DOM manipulation

|  |
| --- |
|  |

1. **Avoid long running JavaScript**

These we will talk more in part 2 we will discuss about optimize the code.

Remember delivery of JavaScript will lead to fast render tree.

##### Module Bundlers

As our application bigger and bigger as especially for single page application, we have more JavaScript files. So have a need for bundler (to bundle all the JS file to one or more depending on the requirement).

As of now in the react app, atomically handle the bundling using something called webpack. There are many bundlers in the market.

* + - * **WebPack**: - If we have a large project, we mostly use webpack. This is mostly setup in the beginning of the project by Team lead or senior developer.
      * **Parcel**: - Use parcel when it is small project. It super easy to use with zero configurations.
      * **Rollup.js** :- Use it when we rolling up our own npm packages

|  |
| --- |
|  |

###### WebPack

<https://webpack.js.org/>

It bundle all our files into static assets. Webpack has four main ideas:-

1. Entry: - is usually the JS file and entry means where webpack enters into the project. Usually this is index.js
2. Output: - Where should webpack output all the files. Usually have a build folder.
3. Loader: - Tools that compiles the code.
4. Plugins:- Play a vital role in output the code and we can do different things using different sort of plugin based on how we want to output the code.

For development of webpack install the below 3 packages:-

npm install --save-dev **webpack** **webpack-dev-server** **webpack-cli**

To check the webpack and worked on use **npm run eject**

#### Code Optimizing

* + - Keep your JavaScript parse and compile time low.
    - Time to first meaningful paint
    - Time to interactive.

|  |
| --- |
|  |

Browser does JIT (just in time compilation) as the JavaScript file loaded on to the browser.

Angular team has come up with AOT (Ahead of Time compilation)

One of the biggest performance culprits is triggering animation during scroll the event

##### Code Splitting

Code Splitting or progressive bootstrapping, where we send minimally functional page compose of HTML, CSS and JavaScript need of the current route (homepage) as more resource arrive, the app can lazy load or unlock more feature. Suppose click on another page it load different JS file. By splitting this up in each compartment, we give browser less work to do.

Lazy load means just load after the page becomes interactive. So that user can fill like the app is fast.

|  |
| --- |
|  |

By using import we automatically tell to code split

###### Manual

Let see the below example of code splitting in react app, we just removing from the top and importing only when they them

|  |
| --- |
| import React, { Component } from 'react';  import './App.css';  import Page1 from './Components/Page1';  //import Page2 from './Components/Page2';  //import Page3 from './Components/Page3';  class App extends Component {  constructor() {  super();  this.state ={  route: 'page1',  component : ''  }  }  onRouteChange = (route) => {  //on code splitting  //this.setState({route: route});  //with code splitting  if (route === 'page1') {  this.setState({route: route});  } else if (route === 'page2') {  //it will work if we use webpack  import('./Components/Page2').then((Page2) => {  this.setState({route:route, component:Page2.default})  })  } else if (route === 'page3') {  import('./Components/Page3').then((Page3) => {  this.setState({route:route, component:Page3.default})  })  }  }  render() {  /\*if (this.state.route === 'page1') {  return <Page1 onRouteChange={this.onRouteChange}/>  }  else if (this.state.route === 'page2') {  return <Page2 onRouteChange={this.onRouteChange}/>  }  else if (this.state.route === 'page3') {  return <Page3 onRouteChange={this.onRouteChange}/>  }\*/  if(this.state.route === 'page1') {  return <Page1 onRouteChange={this.onRouteChange} />  }  else {  return <this.state.component onRouteChange={this.onRouteChange} />  }        }  }  export default App; |

###### Using Asynchronous Component

Second way using asynchronous component

1. Asynchronous component and also we use higher order component.

|  |
| --- |
| import React, { Component } from 'react';  export default function asyncComponet(importComponent) {  class AsyncComponent extends Component {  constructor(props) {  super(props);  this.state = {  component: null  }  }    async componentDidMount(){  const {default: component} = await importComponent();  this.setState({  component: component  })  }  render(){  const Component = this.state.component;  return Component ? <Component {...this.props} /> : null;  }  }  return AsyncComponent;  } |

1. In app.js using the Asynchronous function

|  |
| --- |
| import React, { Component } from 'react';  import './App.css';  import Page1 from './Components/Page1';  //import Page2 from './Components/Page2';  //import Page3 from './Components/Page3';  import AsyncComponent from './Components/AsyncComponent';  class App extends Component {  constructor() {  super();  this.state ={  route: 'page1',  component : null  }  }  onRouteChange = (route) => {  //on code splitting  this.setState({route: route});  //with code splitting  /\* if (route === 'page1') {  this.setState({route: route});  } else if (route === 'page2') {  //it will work if we use webpack  import('./Components/Page2').then((Page2) => {  this.setState({route:route, component:Page2.default})  })  } else if (route === 'page3') {  import('./Components/Page3').then((Page3) => {  this.setState({route:route, component:Page3.default})  })  }\*/  }  render() {  if (this.state.route === 'page1') {  return <Page1 onRouteChange={this.onRouteChange}/>  }  else if (this.state.route === 'page2') {  const AsyncPage2 = AsyncComponent(() => import('./Components/Page2'));  return <AsyncPage2 onRouteChange={this.onRouteChange}/>  }  else if (this.state.route === 'page3') {  const AsyncPage3 = AsyncComponent(() => import('./Components/Page3'));  return <AsyncPage3 onRouteChange={this.onRouteChange}/>  }  /\*if(this.state.route === 'page1') {  return <Page1 onRouteChange={this.onRouteChange} />  }  else {  return <this.state.component onRouteChange={this.onRouteChange} />  }\*/  }  }  export default App; |

As our App gets bigger and bigger, we should think of route based and component based code splitting.

**At end idea is how we can shift the least amount of JavaScript to the user.**

###### React.lazy

Using react.lazy is says only load the component when we use it in the render function

|  |
| --- |
| import React, { Component, Suspense } from 'react';  import './App.css';  import Page1 from './Components/Page1';  const Page2Lazy = React.lazy(() => import('./Components/Page2'));  const Page3Lazy = React.lazy(() => import('./Components/Page3'));  class App extends Component {  constructor() {  super();  this.state ={  route: 'page1',  component : null  }  }  onRouteChange = (route) => {  //on code splitting  this.setState({route: route});  }  render() {  if (this.state.route === 'page1') {  return <Page1 onRouteChange={this.onRouteChange}/>  }  else if (this.state.route === 'page2') {  return (  <Suspense fallback={<div>Loading...</div>}>  <Page2Lazy onRouteChange={this.onRouteChange}/>  </Suspense>  );  }  else if (this.state.route === 'page3') {  return (  <Suspense fallback={<div>Loading...</div>}>  <Page3Lazy onRouteChange={this.onRouteChange}/>  </Suspense>  );  }  }  }  export default App; |

How to check performance of a react website?

Ans:- url?**react\_perf**

###### React code optimization

* + How to stop any component to update other than render?

Using **shouldComponentUpdate()** life cycle hooks we can stop it by returning false.

|  |
| --- |
| import React, { Component } from 'react';  class Header extends Component {  shouldComponentUpdate(nextProps, nextState){  return false;  }  render() {  return <h1 className="f2">RoboFriends</h1>  }  }  export default Header; |

#### Progressive Web Apps

A web Apps is a website that allows users to interact with the webpage in many ways, whether it is games, twitter, facebook it is inside of a browser window (such as chrome, safari, firebox etc.).

And native apps on the other are an app that is on a mobile phone like an iphone or samsumg.

Progressive web Apps helps to behave web apps as native app.

Create react app already setup the progressive web apps. Using light house we can check the website is supported or not.

<https://developers.google.com/web/tools/lighthouse>

Check List for Progressive Web Apps:-

<https://developers.google.com/web/progressive-web-apps/checklist>

Progressive needs to below things:-

##### HTTPS

Https is basically encrypted the data that we send to the server.

How to make a website https:-

To do that we need a certificate: - <https://letsencrypt.org/>

##### App Manifest

All use the Meta tag for viewport, which optimize the app for multiple devices.

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="utf-8" />  <link rel="icon" href="%PUBLIC\_URL%/favicon.ico" />  <meta name="viewport" content="width=device-width, initial-scale=1" />  <meta name="theme-color" content="#000000" />  <meta  name="description"  content="Web site created using create-react-app"  />  <link rel="apple-touch-icon" href="%PUBLIC\_URL%/logo192.png" /> |

In create-react-app we already have the manifest.json file present public folder.

|  |
| --- |
| {  "short\_name": "React App",  "name": "Create React App Sample",  "icons": [  {  "src": "favicon.ico",  "sizes": "64x64 32x32 24x24 16x16",  "type": "image/x-icon"  },  {  "src": "logo192.png",  "type": "image/png",  "sizes": "192x192"  },  {  "src": "logo512.png",  "type": "image/png",  "sizes": "512x512"  }  ],  "start\_url": ".",  "display": "standalone",  "theme\_color": "#000000",  "background\_color": "#ffffff"  } |

For checking the manifest in the browser just got to Application tab in chrome developer tool.

##### Service Worker

Is a script that the browser runs in the background, separate from the webpage web apps.

It is generally use for features that don’t need a webpage or user interaction. Think of service worker another worker that now we have in the browser a background worker. It acts as a programmable proxy. It is the reason that our progressive web apps work offline.

<https://jakearchibald.github.io/isserviceworkerready/>

**Service Worker is already implement in the create-react-app. Service Worker is implemented very easily into a react based project. You can see**[**this Git Diff**](https://github.com/jeffposnick/create-react-pwa/compare/starting-point...pwa)**to see what you would need to do to add service worker into an existing create react app project without the default service worker.**

<https://github.com/jeffposnick/create-react-pwa/compare/starting-point...pwa>

**Without a service worker** in an app our browser will send request to the network, we find the server and server respond with files a response.

**With a service worker** in an app our browser will communicate with the servicer worker and check to see if the app really need to communicate with the network (web server) because may the service worker has already have those files. And then service worker will try to access to **Cache Api** (nothing but the cache storage in the browse) and cache api contains the files such as js, css, images any static files. If the files not present then the browser will communicate with the server instead of service worker.

|  |
| --- |
|  |

### Back End

#### CDNs

Its helps by accelerating almost any website by caching its files in server around the world.

**Benefits: -**

* + - Its improve page load speed.
    - Handle higher traffic loads.
    - Blocks spammer.
    - It reduces the bandwidth consumption.
    - Sometime it also does load balancing between multiple servers.
    - Also protects websites from DDOS attack (Distributed Denial of service attack).

**Why we use CDNs, means what the problem we use CDNs?**

Problem is something called latency, it’s the annoying delay that occurs from the time that I hit enter to request a webpage to actually getting the web page to load onto my browser. And that delay is affected by a number of factors. In all cases though a big part of it is the delay by the physical distance between you and the web sites hosting server

We have different company give the facility:-

CloudFlare: - <https://www.cloudflare.com/>

Amazon: - <https://aws.amazon.com/cloudfront/>

Microsoft: - <https://azure.microsoft.com/en-in/services/cdn/>

|  |
| --- |
|  |

#### GZIP

GZIP other than optimizing images, GZipping is probably the best way to optimize performance.

By default most recent browser it uses GZIP.

In express.js

Step 1: - Import the compression npm package

npm install compression

Step 2: - Use it as a middle ware

|  |
| --- |
|  |

We have another one called **brotli** which is 20% better compression then GZIP

<https://github.com/google/brotli>

#### Database Scaling

Below are the main ways to scale a database

|  |
| --- |
|  |

* **Identify Inefficient Queries**

Making sure to only request what we absolutely need on our server and using things like indexes to improve querying of data.

**Indexing** is a way of sorting a number of records on multiple fields. So if I create an index on a field that is age of the users table it creates a new copy of the data structure which holds the field value and a pointer to the record that it relates to. Think of it as creating another file on the computer instead of having just the user table file. Now we also have a user’s index file. But next time we query any user age questions we can just use the index file to immediately find that user allowing us to uses binary searches, again to really increase performance. Downside of the indexing is that these indexes require additional space on disk.

|  |
| --- |
|  |

* **Increasing Memory**

We can do that by improving the hardware that the database is working on. By increasing the amount of information in memory we aren’t accessing the disk as often as we increase the speed so that we can store more information in memory for a database.

* **Vertical Scaling**

This is usually done with redis or memchached, vertical scaling can also include increasing memory. But in this case we are taking about adding another service so that the system effectively uses the resources better. So adding a system like redis is kind of like adding memory but not doing it on the same server.

Basically frequent data which the user access we can cached in redis, so that when the user request something it will check first in redis and then if found it checking will move to database. Means we have memory access instead of disk access.

* **Sharding**

Basically breaking up the data in separate disk.

* **More databases**

It helps to break the requests in multiple databases instead of giving the entire request to one database. Basically distribute the load. PostgreSQL by default it do that or we can use load balancing.

* **Database Type**

|  |
| --- |
|  |

#### Caching

Caching is the process of storing some data in cached, cached is a temporary storage component area where the data is stored so that in future data can be served faster.

Caching is everywhere

|  |
| --- |
|  |

**Suppose we change something in the css/javascript files changed after deployed to Prod, then how the cache is updated?**

Solution is **Cache busting** in react app just npm run build it will create new css/javascript files with different random number, so that when it load it refresh from the server instead of cache.

**We already see caching the client side in react app by using bundling or service worker.**

##### How to cache in the server side or the express app?

We can take advantage of http cache which comes with all browsers.

|  |
| --- |
|  |

|  |
| --- |
|  |

#### Load Balancing

Load Balancing is the way to balance multiple requests at the same time and distributed them to different services.

Instead of creating own load balancing, we have company to serve load balancer:-

* Digital Ocean: - <https://www.digitalocean.com/products/load-balancer/>
* Amazon : - <https://aws.amazon.com/elasticloadbalancing/>

##### NGINX

<https://www.linode.com/docs/web-servers/nginx/how-to-configure-nginx/>

As long we have nginx that act as a reverse proxy and pass down the request to the particular server (1, 2, 3) it can receive the file and server them really quick to the browser

Also we cache those request, so that next time the user request the same html page instead of going to the server it can serve from the cache.

|  |
| --- |
|  |

Also check the below architecture where we use the CDN

|  |
| --- |
|  |

##### Load Test

Tools we have for loadtesting:

* + - loadTest : -<https://www.npmjs.com/package/loadtest>
    - Artillery: - <https://artillery.io/>
    - Siege: - <https://github.com/JoeDog/siege>
    - Wrk2: - <https://github.com/giltene/wrk2>

We will use the loadtest npm package.

Step 1: - npm install -g loadtest

Command :-

loadtest -t <timeinsecond> -c <noofconcurrency> --rps <noofrequestperseconf> <url>

e.g:- loadtest -t 5 -c 100 --rps 100 http://localhost:80

## Testing

Testing is a method of software development where individual unit or source code, assets or programs are tested to see whether they work properly.

TDD: - Test driven development that is the idea of writing test even before write the application code. So that we start writing code based on the code that you provide.

### Types of Testing:-

#### Unit Test: -

#### Test individual function or classes. Unit Test should cover all small pure function in an application. A Small pure function means that takes an input and return an output and also should have side effects like affecting other function; this is big benefit of functional programming with respect to OOPS.

Unit Test don’t test the contract, contract means connection between server and database, function and another function

#### Integration Test: -

Testing different pieces of code worked together. E.g.:- Testing database connection with the express that requesting the database.

Testing the cross communication between two unit of code.

#### Automation Test (End-to-End or UI Test): -

This usually involves testing real life scenarios on the browser. We can human to do that or programmatically it runs the test cases for us.

It’s always runs inside the browser or browser like environment.

E:g :- Nightmare, **testcafe**, cypress, webdriverIO

### Testing Libraries:-

1. First we need a testing library. This is the scaffolding giving us the ability to do some function call and some new methods to write some test. These are just NPM installed library.
   * + - 1. Jasmine
         2. JEST
         3. Mocha
2. Second we need assertion library. Assertion library is a tool that allows testing that the variable contain the accepted value.
   * + - 1. Jasmine
         2. JEST
         3. Chai <https://www.chaijs.com/>
3. Third we need a test runner. It allows us to run the test. A test runner can run test in different environment
   * + - 1. Jasmine
         2. JEST
         3. Mocha
         4. Karma: - This allows running test in the browser.

Usually the JS files works with DOM to make the web apps, but when we running test and this karma.js does allows us to run the test on the browser. But running the test in the browser takes more times using the browser API.

**How to run a test outside the browser:-**

* Other option is **Puppeteer** (by Google) which we called as head less browser; it’s a node library which provides high level API to control the head list version. We also configure to do things such as generate screenshot, pdf of pages, automate form submission, UI testing, monitor keyboard input.
* **JSDOM** it’s a make a fake version of DOM, we can DOM like api to work without actually needing the browser.

1. Fourth we need mock, spies and stub.
   1. Spies :- provide information about function
   2. Stub:- replaces selected function with a function to ensure the accepted behavior can happen
   3. Mock:- Faking a function and behavior to test different parts of a process
      * + 1. JASMINE
          2. JEST
          3. Sinon.js
2. Fifth we have code coverage, to check how much code is covered for testing.
   * + - 1. Istanbul
         2. JEST

|  |
| --- |
|  |

### JEST

Step 1:- **npm install --save-dev jest**

Installing in Dev only as testing we need to do in development environment only

Step 2:- we usually type **npm test** to run the test file. But we can automatically run test using the below syntax in the package.json, after running one time **npm test**

|  |
| --- |
| {  "name": "TEST",  "version": "1.0.0",  "description": "",  "main": "index.js",  "scripts": {  "test": "jest --watchAll \*.js"  },  "keywords": [],  "author": "",  "license": "ISC",  "dependencies": {},  "devDependencies": {  "jest": "^25.1.0"  }  } |

#### Pure Function

Step 3 :- Creating the test case for the below function

**Script.js**

|  |
| --- |
| const googleDatabase = [  'cats.com',  'souprecipes.com',  'flowers.com',  'animals.com',  'dogpictures.com',  'changelife.com'  ];  const googleSearch = (searchInput, db) => {  const matches = db.filter(website => {  return website.includes(searchInput)  })  return matches.length > 3 ? matches.slice(0,3) : matches  }  //console.log(googleSearch('c',googleDatabase));  module.exports = googleSearch; |

**Script.test.js**

it() is the predefined function in jest and dbMock is the mocking of the database instead of real database.

|  |
| --- |
| const googleSearch = require('./Script');  dbMock = [  'dog.com',  'cheesepuff.com',  'disney.com',  'dogpictures.com'  ]  it('it is searching google',() => {  expect(googleSearch('test', dbMock))  .toEqual([]);    expect(googleSearch('dog', dbMock))  .toEqual(['dog.com','dogpictures.com']);  }) |

How to group test case of same function using describe()

|  |
| --- |
| const googleSearch = require('./Script');  dbMock = [  'dog.com',  'cheesepuff.com',  'disney.com',  'dogpictures.com'  ]  it('this is a test',() => {  expect('hello').toBe('hello');    })  describe('googleSearch',() =>{  it('it is searching google',() => {  expect(googleSearch('test', dbMock))  .toEqual([]);    expect(googleSearch('dog', dbMock))  .toEqual(['dog.com','dogpictures.com']);  })    it('work with undefined and null input',() =>{  expect(googleSearch(undefined,dbMock))  .toEqual([]);    expect(googleSearch(null,dbMock))  .toEqual([]);  })    it('does not return more than 3 matches',() => {  expect(googleSearch('.com',dbMock).length)  .toEqual(3);  })    }) |

#### Asynchronous Function

As we know fetch() is not present in node as it is window object, for implementing node we need to install **npm install node-fetch**

|  |
| --- |
| const fetch = require('node-fetch');  const getPeoplePromise = fetch => {  return fetch('https://swapi.co/api/people')  .then(response => response.json())  .then(data => {  return {  count: data.count,  results : data.results  }  })  }  //check the difference between promise and async code  const getPeople = async (fetch) => {  const getRequest = await fetch('https://swapi.co/api/people');  const data = await getRequest.json();  return {  count: data.count,  results : data.results  }  }  module.exports = {  getPeople,  getPeoplePromise  } |

**In Script.test.js**

|  |
| --- |
| const fetch = require('node-fetch');  const swapi = require('./Script2');  it('calls swapi to get people', () => {  //for asynchronous testing always use the below assertions and also return the promise  //Here 2 we passed because we expect() two times  expect.assertions(2)  return swapi.getPeople(fetch).then(data => {  expect(data.count).toEqual(87);  expect(data.results.length).toBeGreaterThan(5);  })  }) |

#### Mock

With a mock we can fake a function, mock function also known as spies in JEST because they let us spy on the behavior of the function.

|  |
| --- |
| const fetch = require('node-fetch');  const swapi = require('./Script2');  it('get peope returns count and result',() =>{    const mockFetch = jest.fn()  .mockReturnValue(Promise.resolve({  json: () => Promise.resolve({  count:87,  results: [0,1,2,3,4,5]  })  }))  expect.assertions(4);  return swapi.getPeople(mockFetch).then(data =>{  expect(mockFetch.mock.calls.length).toBe(1);  expect(mockFetch).toBeCalledWith('https://swapi.co/api/people')  expect(data.count).toEqual(87);  expect(data.results.length).toBeGreaterThan(5);  })  }) |

### Enzyme

This allows us to render the component in a test environment.

<https://airbnb.io/enzyme/>

Step 1:-

Install the below package

npm i --save-dev **enzyme** **enzyme-adapter-react-16**

And it automatically setup in react project by react-scripts

Step 2:-

In setupTests.js which is already present in create-react-app.

|  |
| --- |
| import {configure} from 'enzyme';  import Adapter from 'enzyme-adapter-react-16';  configure({adapter: new Adapter() }) |

Step 3:-

Set up the component with enzyme

**Card.js**

|  |
| --- |
| import React from 'react';  const Card = ({name, email, id}) => {  return (  <div className='bg-light-green dib br3 pa3 ma2 grow bw2 shadow-5'>  <img alt="robot" src={`https://robohash.org/${id}?size=200x200`} />  <div>  <h2>{name}</h2>  <p>{email}</p>  </div>  </div>  );  }  export default Card; |

**In Card.test.js**

|  |
| --- |
| import {shallow, mount, render} from 'enzyme';  // most of the time we will use shallow  // mount :- we use for full DOM rendering, it idea if we have a component that interact with API of DOM  // this means it has to run an environment that atleast looks like the browser environment  //render :- Is used to render react component, unlike to react-DOM it renders to a static html  import Card from './Card';  import React from 'react';  it('expect to render card component',() => {  expect(shallow(<Card />).length).toEqual(1);  }) |

### Snapshot Testing

It basically makes a copy of the component and if anything changed in the component after taking the snapshot test case will fail.

|  |
| --- |
| import {shallow} from 'enzyme';  import Card from './Card';  import React from 'react';  it('expect to render card component',() => {  expect(shallow(<Card/>).debug()).toMatchSnapshot();  }) |

### Code Coverage for a react project

**npm test -- --coverage --watchAll=false**

### Testing the State component

CounterButton.js

|  |
| --- |
| * import React, {Component} from 'react'; * class CounterButton extends Component { * constructor(props) { * super(props); * this.state = {count: 1}; * } * shouldComponentUpdate(nextProps, nextState) { * if (this.state.count !== nextState.count) { * return true; * } * return false; * } * render() { * console.log('counter'); * return ( * <button * id="counter" * color={this.props.color} * onClick={() => this.setState(state => ({count: state.count + 1}))}> * Count: {this.state.count} * </button> * ); * } * } * export default CounterButton |

CounterButton.test.js

|  |
| --- |
| import {shallow} from 'enzyme';  import React from 'react';  import CounterButton from './CounterButton';  it('expect to render Counter component', () => {  const mockColor = 'red'  expect(shallow(<CounterButton color={mockColor}/>).debug()).toMatchSnapshot();  })  it('correctly increments the counter',() => {  const mockColor = 'red'  const wrapper = shallow(<CounterButton color={mockColor}/>);  wrapper.find('[id="counter"]').simulate('click');  expect(wrapper.state()).toEqual({count: 2});  expect(wrapper.props().color).toEqual('red');  }) |

## Typescript

Typescript allows the JavaScript to behave like statically type language. Typescript reduces error in production but it does not we should include in any project.

### Dynamic Type vs Static Type and (Strongly Type vs Weakly Type )

|  |
| --- |
|  |

<https://android.jlelse.eu/magic-lies-here-statically-typed-vs-dynamically-typed-languages-d151c7f95e2b>

A language is **statically** typed if the type of a variable is known at compile time. For some languages this means that you as the programmer must specify what type each variable is (e.g.: Java, C, C++); other languages offer some form of *type inference*, the capability of the type system to deduce the type of a variable (e.g.: OCaml, Haskell, Scala, Kotlin)

The main advantage here is that all kinds of checking can be done by the compiler, and therefore a lot of trivial bugs are caught at a very early stage.

Examples: C, C++, Java, Rust, Go, Scala

A language is **dynamically** typed if the type is associated with run-time values, and not named variables/fields/etc. This means that you as a programmer can write a little quicker because you do not have to specify types every time (unless using a statically-typed language with *type inference*).

Examples: Perl, Ruby, Python, PHP, JavaScript

A **strongly-typed** language is one in which variables are bound to specific data types, and will result in type errors if types do not match up as expected in the expression — regardless of when type checking occurs.

*Example: - Python is strong-typed, and so is Java, c#.*

temp = “Hello World!”  
temp = temp + 10; // program terminates with below stated error

A **weakly-typed**language on the other hand is a language in which variables are not bound to a specific data type; they still have a type, but type safety constraints are lower compared to strongly-typed languages.

Example:- PHP is weakly-typed, and so is C, JavaScript

Var a = “Hello World!”;  
a = a + 10; // no error caused  
console.log(a); // “Hello World!10”;

### Static Type in JavaScript

There are many tools other than TypeScript to make JavaScript static type

|  |
| --- |
|  |

Angular is completely using TypeScript.

### Typescript

Typescript always needs a node.js, as the compiler of the typescript need an environment of node.js as need to run JavaScript.

**npm install -g typescript**

To run a Typescript we need to type**: tsc**

To create a config file: **tsc –init**

#### Types

|  |
| --- |
| //boolean  let isCool: boolean = false;  //number  let age: number = 56;  //string  let eyeColor: string = 'brown';  let favoriteQuote: string = `I'm not old, I'm only ${age}`;  //Array  let pets: string[] = ['cat', 'mouse', 'dragon'];  let pets2: Array<string> = ['pig', 'lion', 'dragon'];  //Tuple  let basket: [string, number];  basket = ['basketball', 10];  //Enum  enum Size {Small = 1, Medium, Large}  let sizeName: string = Size[2];  alert(sizeName); // Displays 'Medium' as its value is 2 above  //Any  let whatever: any = 'aaaaghhhhhh noooooo!';  //void  let sing = (): void => console.log('Lalalala')  //null and undefined  let meh: undefined = undefined;  let noo: null = null;  //never- if the function never returns  let error = (): never => {  throw Error('blah!');  }  // Type Assertions:  let ohhithere: any = "OH HI THERE";  let strLength: number = (ohhithere as string).length;  //Interface  interface RobotArmy {  count: number,  type: string,  magic?: string  }  let fightRobotArmy = (robots: RobotArmy) =>{  console.log('FIGHT!');  }  let fightRobotArmy2 = (robots: {count: number, type: string, magic?: string}) =>{  console.log('FIGHT!');  }  //Function  let fightRobotArmyF = (robots: RobotArmy): void =>{  console.log('FIGHT!');  }  let fightRobotArmy2F = (robots: {count: number, type: string, magic?: string}): void =>{  console.log('FIGHT!');  }  // \*\*\* Classes  class Animal {  private sing: string;  constructor(sound: string) {  this.sing = sound;  }  greet() {  return "Hello, " + this.sing;  }  }  let lion = new Animal("Lion");  // lion.sing  //In TypeScript, there are several places where type inference  //is used to provide type information when there is no explicit  //type annotation. For example, in this code  let x = 3;  // automatimally detexts x is a number.  //Union Type  let confused: string | number = 'hello' |

## SPA vs Server Side Rendering

**SPA**: - it will have one html file and other thing will be done via JavaScript without rendering the whole page.

**Server Side Rendering**: - navigate to each page need a trip to server.

**Two main issues with SPA:-**

1. More JavaScript is send to the client on the initial request, takes longer to send and execute JavaScript.
2. Search Engine Optimization (SEO) performance, it lot harder to do good SEO on a SPA than a server side render application.

### CSR vs SSR

**CSR:-**

Benefits are if we click on a page to load more content, the browser does not have to make another request to the server.

Problems are sometimes we can have blank pages for initial rendering to happen.

|  |
| --- |
|  |

|  |
| --- |
|  |

**SSR:-**

It renders lot faster on initial request compare to CSR.

It is better for static sites, initial page load is faster and SEO is btter

|  |
| --- |
|  |

|  |
| --- |
|  |

**Some Graph of CSR, SSR and Progressive web apps**

|  |
| --- |
|  |

|  |
| --- |
|  |

We can Progressive Rendering by code splitting as we see in performance section

|  |
| --- |
|  |

**Prerender.io**

It allows JavaScript website to be crawled for search engine.

<https://prerender.io/>

### Server Side Rendering in React

We need both in server and the browser. Remember that as we are showing for react, but the principle and idea will be same for Angular, vue.js etc.

|  |
| --- |
|  |

We have tools for Server Side Rendering in React

<https://www.gatsbyjs.org/>

<https://nextjs.org/>

|  |
| --- |
|  |

### Next.js

Next.js includes routing, webpack, we can different types of CSS feature, supports TypeScript and many more to create server side rendering very fast.

* + - * Creating the package.json

npm init –y

* + - * As we are building a react application, which is server side rendering so we need react and react-dom also

npm install **next** **react** **react-dom**

* + - * Change the package.json to start next

|  |
| --- |
| * { * "name": "Nextjs", * "version": "1.0.0", * "description": "", * "main": "index.js", * "scripts": { * "start": "next" * }, * "keywords": [], * "author": "", * "license": "ISC", * "dependencies": { * "next": "^9.2.1", * "react": "^16.12.0", * "react-dom": "^16.12.0" * } * } |

Npm start will give error to create pages directory

* + - * Create the pages directory/folder
      * Create the page (index.js) in the pages folder and then create the component and export it

|  |
| --- |
| * const Index = () => ( * <div> * <h1>SSR Magician</h1> * </div> * ); * export default Index; |

* + - * Navigate to other page using Link, using Link doing the client side routing if we use <a href> it will do server side routing.

|  |
| --- |
| import Link from 'next/link';  const Index = () => (  <div>  <h1>SSR Magician</h1>  <Link href="/about">About</Link>  </div>  );  export default Index; |

* + - * Dynamic Apps with Next.js

For doing fetch operation we need the below package in next

npm install isomorphic-unfetch

|  |
| --- |
| import Link from 'next/link';  import fetch from 'isomorphic-unfetch';  const Robots = (props) => {  return (  <div>  <h1>Robots</h1>  <Link href='/'>  <button>Home</button>  </Link>  <div>  {  props.robots.map(robot => (  <li key={robot.id}>  <Link href= {`robots/${robot.id}`}>  <a>{robot.name}</a>  </Link>  </li>  ))  }  </div>  </div>  )  }  Robots.getInitialProps = async function(){  const res = await fetch('https://jsonplaceholder.typicode.com/users');    const data = await res.json();  return {  robots : data  }  }  export default Robots; |

* + - * Next.js updated with create-next-app

<https://nextjs.org/blog/create-next-app>

* + - * Updating the next.js in the project

npm i next@latest react@latest react-dom@latest

## Security

Check the below principles if we applied in our application for security.

|  |
| --- |
|  |

### Injections

Injections are Injecting code into another piece of code, most common injection are SQL injection. Anytime user input something from the website we can do injection.

How to Prevent it:-

1. Sanitize input :- Idea of wild list philosophy, means for data validation or anything the user input only allow the user to input data of expected type. Like text, email, phone number make sure using JavaScript only those things are used. Some of the thing we do is removing HTML tags.
2. Parametrize Queries:- It’s a way to prevent sql injection using Knex.js or other ORMS

### 3rd Party Libraries

We may many third party libraries in our project, but sometimes some library is forbidden the security policy.

|  |
| --- |
|  |

**Nsp** :- after installed type **nsp check** in the project, it will give all the vulnerabilities.

**Synk**:- audit the node modules directory of our project. **Synk test** command and also it should connect to the GitHub account by synk auth

If you have NPM version 6 or higher, npm has finally added security as a built-in feature when downloading packages (yey!). You will get notified of security vulnerabilities in the packages you install now.

### Logging

Logging basically means getting the information for the system. Logging means able to detect the issue quickly and prevent any bad things that might be happen with the app.

The two tools we can use in node.js.

|  |
| --- |
|  |

1. Morgan :- Its can http request logger middleware for node.js

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

1. Winston:-

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

### HTTPS

HTTPS is SSL/TLS certificates because HTTP without the S is a clear text protocol, it means the information we send from client to the server it’s in plain text, if anybody intercept the message they can read it.

SSL/TLS create a tunnel know as HTTPS this encrypt the information between client and server while we making http request.

Two ways we can get HTTPS in the website:-

1. LetsEncrypt :- <https://letsencrypt.org/>
2. CloudFlare: - If we host our website by cloudflare, we are going to get HTTPS.

It also added benefits they helped on **DDOS (Distributed Denial of Service Attack)** means when tons of computer is try to send request to a server at the same time

<https://www.cloudflare.com/en-in/>

### XSS + CSRF

**XSS (Cross Side Scripting):-**

XSS occurs whenever an application includes an untrusted data in a new webpage without proper validation. XSS allows executing script in Victim browser.

Cross-site scripting (XSS) is a type of computer security [vulnerability](https://en.wikipedia.org/wiki/Vulnerability_(computer_science)) typically found in [web applications](https://en.wikipedia.org/wiki/Web_application). XSS enables attackers to [inject](https://en.wikipedia.org/wiki/Code_injection) [client-side scripts](https://en.wikipedia.org/wiki/Client-side_script) into [web pages](https://en.wikipedia.org/wiki/Web_page) viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass [access controls](https://en.wikipedia.org/wiki/Access_control) such as the [same-origin policy](https://en.wikipedia.org/wiki/Same-origin_policy).

**CSRF (Cross Site Request Forgery) :-**

Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they’re currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request. With a little help of social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker’s choosing. If the victim is a normal user, a successful CSRF attack can force the user to perform state changing requests like transferring funds, changing their email address, and so forth. If the victim is an administrative account, CSRF can compromise the entire web application.

Prevents by:-

|  |
| --- |
|  |

The tools for prevent CSRF:-

1. Csurf:- <https://www.npmjs.com/package/csurf>

### Code Secrets

In code secrets we need to mainly focus on two main things, so that the secret information should not be revealed like APIKey, DBPassword etc.:-

|  |
| --- |
|  |

1. Environment Variable:-

In React App we can file .env where we can declare all the environment variable want to use.

* + - * In .env file we declare a variable

**REACT\_APP\_SAY\_HI = HII**

* + - * And in the application we can use it

**Console.log(PROCESS.env.REAT\_APP\_SAY\_HI)**

For project other than react we can use the package called dotenv, for react it is already in build.

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

1. Commit History

GitHub will all the commit history of the app, so never ever put the secret files(password, API\_KEY etc.) in GitHub

### Secure Headers

Headers are really important that it tells browser what to do with some of the content that we delivered and also to the server what to do as well.

For securing the headers in express we have helmet package

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

|  |
| --- |
|  |

### Access Control

Access Control is having restriction on what authenticate user are allowed or not. Always give least amount of privilege possible.

|  |
| --- |
|  |

CORS: - Cross Origin Resource Sharing. By enabling CORS we are allowing the server to access from outside.

And also we can allow the particular website to only access the server. Please check the below url.

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

CORS is one of the examples for access control same apply to Administrative privilege, Database Access.

Main idea is Principal of Least Privilege never gives more privilege then we need to.

### Data Management

Some always have backup of the Data (Website, Data, Static files) some more safe never have one point of failure.

Limit the sensitive Data exposure, like email, password, phone number, access token etc. by encrypting it.

What can we do in Data Management?

|  |
| --- |
|  |

**Password Hashing**: - Can be done by **bcrypt, scrypt , Arogan2** as they are really good because it try and tested. We don’t store plain password in the database instead we will hash and then save in database.

**Encrypt Data**: - pgcrypto for PostgreSQL

### Don’t Trust Anyone

Don’t Trust anyone from outside

|  |
| --- |
|  |

For example: - Sometime any false client can send multiple false requests to the server. For that we can use a rate limiter present in node.js

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

### Authentication

Authentication means making sure that the person on the other end is who they say they are. This is done by password to authenticate that the user that they are.

|  |
| --- |
|  |

## Docker

|  |
| --- |
|  |

Previously we have virtual machine; means a sandbox environment contains a full-fledged computer (virtual hardware, OS, Kernel Software)

Dockers came and change the idea of Virtualization (they wrap up the software in a complete file system that contains everything in around). With Docker container we are able to have fast and easy access to run the small single application and we can multiple container.

**Docker is a container platform unlike virtual machines container bundle only the libraries and setting required for the application. They are efficient light weight and self-contained and this make sure that the application run the same in anywhere and everywhere.**

### Container

We first will have **host** (i.e. machine or computer) where will host the container and top of the host will have the **container** something which create with Docker. Inside the container will have **image** is used to bundle the application into a standalone package that can live inside a container.

Inside the image we will have all the dependency packages like node server and then image is read by Docker because it just a file and this file generates the container that runs node.

|  |
| --- |
|  |

The key part is that the environment inside the container is completely isolated from the host machine.

**Docker bundle the application into an image a standalone executable package and Docker execute this image in the container. And the environment inside the container is completely isolated from the host machine.**

We will develop the below on, everything will be present in one single image file.

|  |
| --- |
|  |

**How the production app looks like**

|  |
| --- |
|  |

### Docker Hub

With Docker Hub it provide a store like website where we search and download images to use that is already prewritten by the people.

<https://hub.docker.com/>

### DockerFile

Step 1:- Create image file with name Dockerfile in the project (at the root) that Docker can use

Step 2:- Inside the file import the image from Docker hub

|  |
| --- |
| FROM node:chakracore |

Step 3:- At the end of the file run the CMD command

|  |
| --- |
| CMD ["/bin/bash"] |

Step 4:- Run the command in the terminal at the period (.) to build all, but we can specify what we want to run

docker build -t <ContainerName> .

Step 5:- Run the container with the below command

docker run -it <ContainerName>

* + **How to run an app inside the container?**

|  |
| --- |
| FROM node:11.3.0  // Give any directory where we want the file to run in docker container  WORKDIR /usr/src/FaceRecognitionBrain-api  //Copy everything from root of the project to the container root  COPY ./ ./  // Give the command to install all the package in the container  RUN npm install  CMD ["/bin/bash"]docker |

How to host the container running app in the machine?

Command : - docker run -it -p <port no of the docker running>:<port no of local> superawesome

docker run -it -p 3008:3008 superawesome

### Docker Commands

* + - * How to build an Image and create a container?

**Run the command in the terminal at the period (.) to build all**

Command: - docker build -t <any name you want to give to a container> .

* + - * How to run a container?

Command: - docker run -it <ContainerName>

* + - * How to run a container at background?

Command: - docker run –it –d <containername>

* + - * How to check which all containers is running at background?

Command: - docker ps

* + - * How to go inside a container running at background?

**ContainerID we can get from the command docker ps**

Command: - docker exec -it < ContainerID> bash

* + - * How to stop running a container?

Command: - docker stop <ContainerID>

* + - * How to host the container running app in the machine?

Command: - docker run -it -p <port no of the docker running>:<port no of local> superawesome

Better use the command : - docker-compose up --build

* + - * How to build the compose file

Command :- <docker-compose| dockercompose filename> build

* + - * How to run the compose file

Command: - docker-compose run

* + - * How to stop a service running and up a service

Command: - docker-compose down

Command: - docker-compose up

### Docker Compose

Docker Compose is used to orchestrate our application services during development. Basically to combine both PostgreSQl, API server etc.

Step 1 : - Create the docker compose file in the root directory, <filename>.yml

|  |
| --- |
| version: "3.7"  services:  smart-brain-api:  container\_name: backend  build: ./  command: npm start  working\_dir: /usr/src/smart-brain-api  ports:  - "3008:3008" |

The above instruction is saying to compose to create a service called **smart-brain-api** using the docker file present in the root directory, we map the container to the **port 3008** and also run **npm start** in the container

Step 2:- Build the compose file

Command :- <docker-compose| dockercompose filename> build

Step 3:- Run the compose file

Command: - docker-compose run <servicename provider in the compose file/ smart-brain-api >

**How to backend service with postgres service?**

|  |
| --- |
| version: "3.2"  services:  smart-brain-api:  container\_name: backend  build: ./  command: npm start  working\_dir: /usr/src/FaceRecognitionBrain-api  #Create the environment variable which we will be using in the app for database connection  environment:  POSTGRES\_URI: postgres://sally:secret@postgres:5432/smart-brain-docker  # link the backend with postgres service in future we will add redis  links:  - postgres  ports:  - "3008:3008"    postgres:  # create the host, username , password for postgres which we used at top  environment:  POSTGRES\_USER: sally  POSTGRES\_PASSWORD: secret  POSTGRES\_DB: smart-brain-docker  POSTGRES\_HOST: postgres  # build is used for the postgres folder where we create the docker file for  # postgres for create the image  build: ./postgres  ports:  - "5432:5432" |

**In the app use the environment from the docker compose file as describe above**

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

**For creating the new table via docker check the video no 278 (Docker compose 6) , as they are many configuration to set and for insert record in database predefined check the video no 280(Docker compose 7).**

**How to setup redis in docker?**

Use the redis\_uri to the server where we want to acess

|  |
| --- |
| version: "3.2"  services:  smart-brain-api:  container\_name: backend  build: ./  command: npm start  working\_dir: /usr/src/FaceRecognitionBrain-api  #Create the environment variable which we will be using in the app for database connection  environment:  POSTGRES\_URI: postgres://sally:secret@postgres:5432/smart-brain-docker  REDIS\_URI: redis://redis:6379  # link the backend with postgres service in future we will add redis  links:  - postgres  - redis  ports:  - "3008:3008"    postgres:  # create the host, username , password for postgres which we used at top  environment:  POSTGRES\_USER: sally  POSTGRES\_PASSWORD: secret  POSTGRES\_DB: smart-brain-docker  POSTGRES\_HOST: postgres  # build is used for the postgres folder where we create the docker file for  # postgres for create the image  build: ./postgres  ports:  - "5432:5432"  redis:  image: redis  ports:  - "6379:6379" |

**Run the below command to execute redis in the system from docker:**

**Command : - docker-compose exec redis redis**

### Docker Volumes

**How automatically update the service while changing into the local?**

|  |
| --- |
| version: "3.7"  services:  smart-brain-api:  container\_name: backend  build: ./  command: npm start  working\_dir: /usr/src/FaceRecognitionBrain-api  ports:  - "3008:3008"  volumes:  - ./:/usr/src/FaceRecognitionBrain-api |

## Redis

Redis is **NoSQL in memory** database. It is an open source database which used to build really high performance, scalable web application.

Redis is a **key value store**, and this key value is little bit similar how we handle object in JavaScript.

Redis is used **for short lived data** in our application**, it often used with session, web page hit count etc.**

**Downside with redis is its only takes a snapshot and writes it on disk occasionally, so we might loss data if the server goes down.**

|  |
| --- |
|  |

### Redis Commands

<https://redis.io/commands>

* + - * How to set a key value?

Command: - SET key “value”

SET name “Hello”

* + - * How to get the value for the key?

Command: - GET key

GET name

* + - * How to set multiple key values?

Command: - MSET key1 value1 key2 value2

* + - * How to get if key is present or not?

Command: - EXISTS key

* + - * How to delete a key?

Command: - DELETE key

* + - * How to expire a key after sometime?

Command: - EXPIRE key 10

### Data Types

Redis has 5 main data types:-

1. String
2. Hashes
3. List
4. Sets
5. Sorted Sets

#### Hashes

Command: - HMSET user id 45 name “John”

Here we just created a hash with a key user and this key has both id = 45 and name = john

How to get the value in Hashes?

Command: - HGET user id or getting values HGETALL user

#### Lists

It’s like a linked list in data structure, with List insertion is really fast, but its take some time with respect to array when we are searching using key.

* + - * + Left push in the list

Command: - LPUSH ourlist 10

* + - * + Right push in the list

Command: - RPUSH ourlist 20

* + - * + GET the list value

Command: - LRANGE ourlist <start> <end>

LRANGE ourlist 0 1

* + - * + Right and Left DELETE from the list

Command: - RPOP ourlist

Command: - LPOP ourlist

#### SETS

Sets in Redis are unordered collection of strings, only difference sets with lists is it will not have repeated member or values.

* + - * + How to add elements in a set

Command: - SADD ourSets 1 2 3 4 5..

* + - * + How to get the values from the sets?

Command: - SMEMBERS ourSets

* + - * + How to check value is present or not in a set?

Command: - SISMEMBER ourSets 1 // it will return 1 as we have 1 present else 0

#### Sorted Sets

It is just like set where no repeated collection of value but difference every member in the sorted set will associate with a score and this score will allow being in order.

* + - * + How to add elements in a sorted set?

Command: - ZADD sortedSet 50 “John”

* + - * + How to get the values from the sorted sets?

Command: - ZRANGE sortedSet <startrange> <endrange>

ZRANGE sorted set 0 1

## Sessions and JWT

### Cookies vs Tokens

**Cookie based authentication** is stateful means that authentication records or session must be kept both in the browser as well as in server. Browser keeps the session cookie in the header and the server may store in database to figure out for future authentication. And also if the user logout the session get destroyed both from browser as well as from server.

**Token based authentication** is stateless means the server does not need to keep a record of which user are logged in; instead the server need to do is decode the JWT and make sure it is a valid token. Once the user logout the token is destroyed on the client side but no interaction is required on the server as the server don’t store anything. Example: - **JWT** (Jason Web Token Authentication).

**The biggest advantage of Token web authentication is:-**

**Pros**

* + - The idea of token over cookies, it is **stateless** the backend/server does not required to store records/token in database.
    - Each token is self-contained and contains all the data required to checks its validity as well as convey user information.
    - In JWT we can have any type of data like user information userID, username because when server receives, it can see what type of type the token contains.
    - Cookies really good for one single domain with server-client relationship, but the token based makes really easy to work with different API.
    - Token based authentication can work in both browsers as well as in mobile platform like iOS, android.

**Cons**

* + - The size of the JWT token is lot bigger because it contains user information because we are sending them in every request that definitely increases the number of bytes we have to transfer back and forth.
    - If the JWT token stolen then hacker can see that it’s their inside.

|  |
| --- |
|  |

### Token Based Authentication in Express API and also storing JWT token in redis?

Step 1: - Create the method of to check the sign in is authorize, reject the Promise instead of response from the handleSignIn method.

|  |
| --- |
| const handleSignin = (db,bcrypt,req,res)=>{  const { email, password} = req.body;  if(!email || !password){  return Promise.reject('incorrect form submission');  }  return db.select('email','hash').from('login')  .where(  'email','=', email  )  .then(data => {  const isValid = bcrypt.compareSync(password, data[0].hash);  if(isValid){  return db.select('\*').from('users')  .where('email','=',email)  .then(user => user[0])  .catch(err => Promise.reject('unable to get user'))  } else{  Promise.reject('Wrong Credentials')  }  })  .catch(err => Promise.reject('Wrong Credentials'))  }  const getAuthToken = () => {  console.log('auth ok');  }  const signinAuthentication = (db,bcrypt) => (req, res) => {  /\* to check if the user is authorize then no need to login again \*/  console.log(req.body);  const {authorization} = req.headers;  return authorization ? getAuthToken() :  handleSignin(db,bcrypt,req,res)  .then(data => res.json(data))  .catch(err => res.status(400).json(err));  }  module.exports = { signinAuthentication: signinAuthentication  } |

Step 2:- generating the JWT token and send it to the user.

* + - * Install npm for jsonweb

|  |
| --- |
| var jwt = require('jsonwebtoken');  const handleSignin = (db,bcrypt,req,res)=>{  const { email, password} = req.body;  if(!email || !password){  return Promise.reject('incorrect form submission');  }  return db.select('email','hash').from('login')  .where(  'email','=', email  )  .then(data => {  const isValid = bcrypt.compareSync(password, data[0].hash);  if(isValid){  return db.select('\*').from('users')  .where('email','=',email)  .then(user => user[0])  .catch(err => Promise.reject('unable to get user'))  } else{  Promise.reject('Wrong Credentials')  }  })  .catch(err => Promise.reject('Wrong Credentials'))  /\*if(req.body.email === database.users[0].email &&  req.body.password === database.users[0].password){  res.json(database.users[0]);  } else {  res.status(400).json('error logging in');  }\*/  }  const getAuthToken = () => {  console.log('auth ok');  }  const signToken = (email) => {  const jwtPayload = {email};  return jwt.sign(jwtPayload,'JWT\_SECRET', {expiresIn: '2'});  }  const createSessions =(user) => {  const {email,id} = user;  const token = signToken(email);  return {success: 'true',userId: id , token:token}  }  const signinAuthentication = (db,bcrypt) => (req, res) => {  /\* to check if the user is authorize then no need to login again \*/  const {authorization} = req.headers;  return authorization ? getAuthToken() :  handleSignin(db,bcrypt,req,res)  .then(data => {  return data.id && data.email? createSessions(data) : Promise.reject(data)  })    .then(session => res.json(session))  .catch(err => res.status(400).json(err));  }  module.exports = {  signinAuthentication: signinAuthentication  } |

Step 3: - Adding Redis , storing JWT token in redis and also retrieving the auth token from redis.

We are going to store the JWT token in redis database and after storing we have full control to see what token are available, given out and also revoke them anytime we want (by deleting from the database)

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

npm install redis

|  |
| --- |
| const jwt = require('jsonwebtoken');  const redis = require('redis');  //setup redis and using the environment which we set in docker-compose  const redisClient = redis.createClient(process.env.REDIS\_URI);  const handleSignin = (db,bcrypt,req,res)=>{  const { email, password} = req.body;  if(!email || !password){  return Promise.reject('incorrect form submission');  }  return db.select('email','hash').from('login')  .where(  'email','=', email  )  .then(data => {  const isValid = bcrypt.compareSync(password, data[0].hash);  if(isValid){  return db.select('\*').from('users')  .where('email','=',email)  .then(user => user[0])  .catch(err => Promise.reject('unable to get user'))  } else{  Promise.reject('Wrong Credentials')  }  })  .catch(err => Promise.reject('Wrong Credentials'))  /\*if(req.body.email === database.users[0].email &&  req.body.password === database.users[0].password){  res.json(database.users[0]);  } else {  res.status(400).json('error logging in');  }\*/  }  const getAuthToken = (req,res) => {  const {authorization} = req.headers  return redisClient.get(authorization, (err, reply) => {  if(err || !reply){  return res.status(400).json('Unauthorize');  }  return res.json({id:reply});  })  }  const signToken = (email) => {  const jwtPayload = {email};  return jwt.sign(jwtPayload,'JWT\_SECRET', {expiresIn: '2'});  }  const setToken = (key, value) => {  return Promise.resolve(redisClient.set(key,value));  }  const createSessions =(user) => {  const {email,id} = user;  const token = signToken(email);  return setToken(token, id)  .then(() =>{  return {success: 'true',userId:id,token:token}  })  .catch(console.log)  }  const signinAuthentication = (db,bcrypt) => (req, res) => {  /\* to check if the user is authorize then no need to login again \*/  const {authorization} = req.headers;  return authorization ? getAuthToken(req, res) :  handleSignin(db,bcrypt,req,res)  .then(data => {  return data.id && data.email? createSessions(data) : Promise.reject(data)  })    .then(session => res.json(session))  .catch(err => res.status(400).json(err));  }  module.exports = {  signinAuthentication: signinAuthentication  } |

### Client Session Management

We create our JWT token , stored in Redis database and sending it back to the user.

And now we have to store the information on the browser of the user and also we need to send this token actually in the authorization header so that we can get our profile.

We will store the token in session storage in the browser or local storage.

Eg:-

|  |
| --- |
| //Setting in LocalStorage  const selectedSeats = document.querySelectorAll(".row .seat.selected");  const seatsIndex = [...selectedSeats].map(seat => [...seats].indexOf(seat));  localStorage.setItem("selectedSeats", JSON.stringify(seatsIndex));  //Here seatIndex is array, so we need to convert to JSON  //Getting from LocalStorage  //Get Data from LocalStorage and populate UI  const populateUI = () => {  const selectedSeat = JSON.parse(localStorage.getItem("selectSeats"));  }; |

**Difference between local storage and session storage is**

* + Data is local storage present even if the browser is closed. But for session storage is simply just for one single session

|  |
| --- |
| import React from 'react';  import './SignIn.css'  class SignIn extends React.Component{  constructor(props){  super(props);  this.state = {  signInEmail: '',  signInPassword: ''  }  }  onEmailChange = (event) =>{  this.setState({signInEmail: event.target.value});  }  onPasswordChange = (event) => {  this.setState({signInPassword: event.target.value});  }  saveAuthTokenInSession = (token) => {  console.log(token);  window.sessionStorage.setItem('token', token);  }  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.userId && data.success === 'true'){  this.saveAuthTokenInSession(data.token);  this.props.loadUser(data);  this.props.onRouteChange('home')  }  });  } |

**How to send the token from session storage in the authorization header after successfully login(second time accessing the server**)?

- In the app.js by the lifecycle hook **componentDidMount**() before anything happens in the front end, check the token if token exists we need to access the signin endpoint again and give the authorization token to the server, so that the backend check the authorization present then send the authorization id from the redis database, otherwise if token not exists do the normal operation

|  |
| --- |
| class App extends Component{  constructor(){  super();  this.state = initialState;  }  componentDidMount(){  const token = window.sessionStorage.getItem('token');  if(token){  fetch('http://localhost:3008/signin',{  method: 'post',  headers:{  'Content-Type': 'application/json',  'Authorization': token  }  })  .then(res =>res.json())  .then(data => {  if(data && data.id){  console.log('Success we need to get user profile')  }  })  .catch(console.log);  }  } |

### Authorization Using Middleware

We will create auth middleware that always check and protects any endpoint should be very sensitive and should only have authority access them.

In new controllers or create a middleware folder

|  |
| --- |
| const redisClient = require('./signin').redisClient;  const requireAuth = (req, res, next) => {  const {authorization} = req.headers;  if(!authorization){  return res.status(401).json('unthorized');  }  return redisClient.get(authorization,(err, reply) => {  if(err || !reply){  return res.status(401).json('unthorized');  }  console.log('you shall pass');  return next()  })  }  module.exports = {  requireAuth: requireAuth  } |

In Server .js

|  |
| --- |
| app.get('/',(req,res)=>{res.send('its working');})  app.post('/signin',signin.signinAuthentication(db,bcrypt))  app.post('/register',(req,res) => {register.handleRegister(req,res,db,bcrypt)});  app.get('/profile/:id',auth.requireAuth, (req,res)=> {profile.handleProfileGet(req,res,db)})  app.post('/profile/:id', auth.requireAuth, (req,res) =>{profile.handleProfileUpdate(req,res,db)})  app.put('/image', auth.requireAuth, (req,res) => {image.handleImage(req,res,db)});  app.post('/imageurl', auth.requireAuth, (req,res) => {image.handleApiCall(req,res)}); |

**Now in the client side in the every fetch require where authorization is required we need to pass the token from the sessionStorage.**

|  |
| --- |
| onButtonSubmit = () => {  this.setState({imageUrl:this.state.input});  console.log('click');  /\* app.models.predict(  Clarifai.FACE\_DETECT\_MODEL,  this.state.input) \*/  fetch('http://localhost:3008/imageurl',{  method:'post',  headers:{  'Content-Type':'application/json',  'Authorization': window.sessionStorage.getItem('token')  },  body: JSON.stringify({  input: this.state.input  })  })  .then(response => response.json())  .then(response => {  if(response){  console.log(response);  fetch('http://localhost:3008/image',{  method:'put',  headers: {  'Content-Type':'application/json',  'Authorization': window.sessionStorage.getItem('token')},  body: JSON.stringify({  id: this.state.user.id  })  })  .then(response => response.json())  .then(count =>{  this.setState(Object.assign(this.state.user, {entries:count}))  })  .catch(console.log)  }  console.log(response);  this.displayFaceBox(this.calculateFaceDetection(response))  // console.log(response.outputs[0].data.regions[0].region\_info.bounding\_box)  })    .catch(err => console.log(err));  } |

## AWS

|  |
| --- |
|  |

### Amazon Web Services

|  |
| --- |
|  |

#### EC2: -

Think EC2 as a basic server we can run, like we can run backend rest api instead of digital ocean or heroku.

#### S3

S3 is called as Object storage Service, means each object is stored as a file with its metadata and ID. With S3 it enables us to upload or store also download pretty much any file or object that we may want. It has a size limit of 5GB.

For example in the smart brain project like we can store the images that user sends to the clarifai API, so that in future we can train the model better.

#### Lambda

With lambda we can run code virtually for any type of application or backend service. It simply says just give me a function and let me know when to run it. Zero administration or configuration is required.

We just upload our code and lambda takes care of everything require running and scaling the code with high availability.

#### CloudFront

CloudFront helps us to speeds up distribution of our static files, like html , css , js , image files can now be put on cloudFront and amazon deliver this content through a worldwide network of data center exactly like CDN does. This makes our app extremely fast when we request our html, css and JavaScript files because Amazon has so many server every around world, suppose if we are in India whatever server closest to it is going to serve the static files and also it provides automatically https connection.

#### DynamoDB

DynamoDB is a fast NoSQL database and also manage scaling, it uses key-value storage model

#### Amazon Flowchart

It show what people use to make their architecture. We can configure on our requirement of the project.

|  |
| --- |
|  |

### Monolithic vs Micro-Services

|  |
| --- |
|  |

With Docker and AWS now making things extremely easy for manage and connect them , the idea of micro services is more popular means to split the application into small services instead of single monolithic application. And micro service is a small application that has its own architecture consisting of own business logic and concern with doing one things really well.

Benefits of micro services is now we have small chunks that can be tested on their own, we can different developer team on their own and all them can be released individually to prod one at a time.

### Serverless

Let install the Serverless framework, so that we can start playing around amazon lamda and incorporate the features of batch system into the smart brain app.

Serverless will take care of the entire configuration on AWS for us.

Serverless is not only for AWS also for Azure, openwhisk(IBM) , Google.

**npm install -g Serverless**

#### Serverless Command

* + - * To check what all template present: - sls create --help
      * To create a template of nodejs :- sls create -t aws-nodejs
      * To map the template with IAM :- sls config credentials --provider aws --key <key that created via IAMservice> --secret <secret key created via IAM service>
      * To deploy to aws : - sls deploy
      * To invoke a lambda function: - sls invoke --function <functionname from yml file>

**New Template of aws-node.js**

**In handle.js**

|  |
| --- |
| 'use strict';  const emojis = [  '😄','😃','😀','😊','😉','😍','🔶','🔷', '🚀'  ];  module.exports.rank = (event, context, callback) => {  const rank = event.queryStringParameters.rank;  const rankEmoji = emojis[rank >= emojis.length ? emojis.length - 1 : rank];  const response = {  statusCode: 200,  headers: {  "Access-Control-Allow-Origin" : "\*" // Required for CORS support to work  },  body: JSON.stringify({  message: 'Rank generated!',  input: rankEmoji,  }),  };  callback(null, response);  // Use this code if you don't use the http event with the LAMBDA-PROXY integration  // callback(null, { message: 'Go Serverless v1.0! Your function executed successfully!', event });  }; |

**serverless.yml**

|  |
| --- |
| # Welcome to Serverless!  #  # This file is the main config file for your service.  # It's very minimal at this point and uses default values.  # You can always add more config options for more control.  # We've included some commented out config examples here.  # Just uncomment any of them to get that config option.  #  # For full config options, check the docs:  # docs.serverless.com  #  # Happy Coding!  service: rankly-lamda  # app and org for use with dashboard.serverless.com  #app: your-app-name  #org: your-org-name  # You can pin your service to only deploy with a specific Serverless version  # Check out our docs for more details  # frameworkVersion: "=X.X.X"  provider:  name: aws  runtime: nodejs12.x  # you can overwrite defaults here  stage: prod  # region: us-east-1  # you can add statements to the Lambda function's IAM Role here  # iamRoleStatements:  # - Effect: "Allow"  # Action:  # - "s3:ListBucket"  # Resource: { "Fn::Join" : ["", ["arn:aws:s3:::", { "Ref" : "ServerlessDeploymentBucket" } ] ] }  # - Effect: "Allow"  # Action:  # - "s3:PutObject"  # Resource:  # Fn::Join:  # - ""  # - - "arn:aws:s3:::"  # - "Ref" : "ServerlessDeploymentBucket"  # - "/\*"  # you can define service wide environment variables here  # environment:  # variable1: value1  # you can add packaging information here  #package:  # include:  # - include-me.js  # - include-me-dir/\*\*  # exclude:  # - exclude-me.js  # - exclude-me-dir/\*\*  functions:  rank:  handler: handler.rank  events:  - http:  path: rank  method: get  # The following are a few example events you can configure  # NOTE: Please make sure to change your handler code to work with those events  # Check the event documentation for details  # events:  # - http:  # path: users/create  # method: get  # - websocket: $connect  # - s3: ${env:BUCKET}  # - schedule: rate(10 minutes)  # - sns: greeter-topic  # - stream: arn:aws:dynamodb:region:XXXXXX:table/foo/stream/1970-01-01T00:00:00.000  # - alexaSkill: amzn1.ask.skill.xx-xx-xx-xx  # - alexaSmartHome: amzn1.ask.skill.xx-xx-xx-xx  # - iot:  # sql: "SELECT \* FROM 'some\_topic'"  # - cloudwatchEvent:  # event:  # source:  # - "aws.ec2"  # detail-type:  # - "EC2 Instance State-change Notification"  # detail:  # state:  # - pending  # - cloudwatchLog: '/aws/lambda/hello'  # - cognitoUserPool:  # pool: MyUserPool  # trigger: PreSignUp  # - alb:  # listenerArn: arn:aws:elasticloadbalancing:us-east-1:XXXXXX:listener/app/my-load-balancer/50dc6c495c0c9188/  # priority: 1  # conditions:  # host: example.com  # path: /hello  # Define function environment variables here  # environment:  # variable2: value2  # you can add CloudFormation resource templates here  #resources:  # Resources:  # NewResource:  # Type: AWS::S3::Bucket  # Properties:  # BucketName: my-new-bucket  # Outputs:  # NewOutput:  # Description: "Description for the output"  # Value: "Some output value" |

### IAM service

We have to this IAM if you want to use any AWS services and this control permission. Create IAM service and give programmatic access permission.

|  |
| --- |
|  |

After giving all access it will give a accesskey and secret key for AWS profile.

|  |
| --- |
|  |

## CI/CD

### Continuous Integration

Continuous Integration is a development practice where developer integrate code in a shared repository frequently and each integration can then be verified by an automate build and automate test.

Benefit is we can detect error quickly and locate them more easily.

|  |
| --- |
|  |

### Continuous Delivery

Continuous Delivery is a practice of keeping the code base deployable at any point. It is an extension of the continuous integration to make sure that we can release new changes to customer quickly in a sustainable way. We set the activity daily, weekly or monthly with respect to business requirement.

|  |
| --- |
|  |

### Continuous Deployment

It is closely related to continuous delivery and it is referring to keeping the application deployable at any point of time. It goes directly to production there is no manual developer, team lead , manager that clicks a button and say that all the acceptance test have passed and lets deploy to prod.

|  |
| --- |
|  |

### Building Great Software

|  |
| --- |
|  |

* + - **TravisCI or CircleCI** for hosted CI servers.
    - **Jenkins** for your own managed CI servers.

|  |
| --- |
|  |

### CircleCI

Step 1: - Create a folder with name .circleci because any machine files or folder with dot are automatically hidden by default user

Step 2: - Create an yaml file with config.yml where we will configure to what the circleci server will run for us.

|  |
| --- |
| version: 2  jobs:  build:  docker:  - image: circleci/node:12.0  steps:  - checkout  - run: echo "npm installing"  - run: npm install  - run: CI=true npm run build  test:  docker:  - image: circleci/node:12.0  steps:  - checkout  - run: echo "testing stuff"  - run: npm test  hithere:  docker:  - image: circleci/node:12.0  steps:  - checkout  - run: echo "hello"  workflows:  version: 2  build\_test\_and\_int:  jobs:  - build  - hithere  - test:  requires:  - hithere |

Step 3: - Create an account in circleci and via the github repository that we want to CI. After automatically any pull request the circleci will automatically runs

<https://circleci.com/gh/Chiranjit9955>

Now if you want to validate something like formatting before the pull request, we can do via prettier

#### Prettier

<https://prettier.io/>

Prettier helps to format the code.

Here we will use the precommit hook which automatically formats the code before it moves to github

Step 1:- Install the pretty-quicky husky

npm install pretty-quick husky –dev

step 2:- Add the pre-commit command in the package.json, it helps to automatically be detected and run every time we make a commit

|  |
| --- |
| "scripts": {  "predeploy": "npm run build",  "deploy": "gh-pages -d build",  "start": "react-scripts start",  "build": "react-scripts build",  "test": "react-scripts test",  "eject": "react-scripts eject",  "pre-commit": "pretty-quick --staged"  }, |

Step 3: - Install prettier

npm install --save-dev --save-exact prettier

## Resources

### Digital Ocean

Cloud platform for hosting

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

### Asymmetric Encryption

<https://www.youtube.com/watch?v=NmM9HA2MQGI>  
  
<https://www.youtube.com/watch?v=Yjrfm_oRO0w>  
  
<https://www.youtube.com/watch?v=vsXMMT2CqqE&t=>  
  
<https://www.youtube.com/watch?v=NF1pwjL9-DE>

### Set SSH for GitHub

<https://github.com/antonykidis/Setup-ssh-for-github/blob/master/Setup-ssh-on-github.pdf>

### Blog

<https://zerotomastery.io/blog/?tag=WDM>

### Image Types

<https://99designs.com/blog/tips/image-file-types/>

<https://pageweight.imgix.com/> (for analyze any website)

<https://www.sitepoint.com/gif-png-jpg-which-one-to-use/>

### JPG image optimizer

<http://jpeg-optimizer.com/>

### PNG image optimizer

<https://tinypng.com/>

### Media Queries for image optimization

<https://gist.github.com/bartholomej/8415655>

<https://css-tricks.com/snippets/css/media-queries-for-standard-devices/>

### Remove Meta tag of an image

<https://www.verexif.com/en/>

### Performance testing of a website

<https://www.webpagetest.org/>

<https://developers.google.com/speed/pagespeed/insights/>

### Prefetching, preloading, prebrowsing

<https://css-tricks.com/prefetching-preloading-prebrowsing/>

### http2

<https://developers.google.com/web/fundamentals/performance/http2/>

### Road Map of a developer

<https://i.udemycdn.com/redactor/raw/2020-01-18_15-55-19-320398734f15655bb65e6c7b288e0bf2.jpg>

### WebPack

<https://webpack.js.org/>

### WebPack Configuration

<http://web.jakoblind.no/webpack-config/>

### React code splitting

<https://reactjs.org/docs/code-splitting.html>

### What web can do today?

<https://whatwebcando.today/>

### Check List for Progressive Web Apps

<https://developers.google.com/web/progressive-web-apps/checklist>

### How to get certificate for HTTPS?

<https://letsencrypt.org/>

### Hosting a website that automatically gives https?

<https://www.cloudflare.com/en-in/>

### Generate Favicon

<https://realfavicongenerator.net/>

### Check which browser implements Service Worker

<https://jakearchibald.github.io/isserviceworkerready/>

### Implement service worker in create-react-app

<https://github.com/jeffposnick/create-react-pwa/compare/starting-point...pwa>

### Push notification in service worker

<https://auth0.com/blog/introduction-to-progressive-web-apps-push-notifications-part-3/>

### Improve Front End Performance of a website

<https://progressivetooling.com/>

### Free API for testing

<https://swapi.co/>

### JEST

<https://jestjs.io/docs/en/getting-started>

### JEST Cheat Sheet

<https://github.com/sapegin/jest-cheat-sheet>

### Enzyme

<https://airbnb.io/enzyme/>

<https://airbnb.io/enzyme/docs/api/>

### Editor Plugin for Typescript

<https://github.com/Microsoft/TypeScript/wiki/TypeScript-Editor-Support>

### Adding Typescript to Create React app

<https://create-react-app.dev/docs/adding-typescript/>

### GatsBy and next.js for server side rendering in react

<https://www.gatsbyjs.org/>

<https://nextjs.org/>

### Server side vs Client side routing

<https://medium.com/@wilbo/server-side-vs-client-side-routing-71d710e9227f>

### SEO in JavaScript

<https://developers.google.com/search/docs/guides/javascript-seo-basics>

### Prerender.io :- It allows JavaScript website to be crawled for search engine.

<https://prerender.io/>

### SQL Injection play around

<https://www.hacksplaining.com/exercises/sql-injection>

### Morgan: Logger for node.js

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

### Winston: Logger for node.js

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

### LetsEncrypt: - for getting the SSL/TLS certificate

<https://letsencrypt.org/>

### Csurf:- To prevent CSRF

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

### dotenv – for creating .env in the project

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

### Helment – for securing the HTTP Headers

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

### Storing Password Securely

<https://rangle.io/blog/how-to-store-user-passwords-and-overcome-security-threats-in-2017/>

### Hacking

<https://www.hacksplaining.com/lessons>

<https://watchyourhack.com/>

### Docker Hub: - To download the image file for different package.

<https://hub.docker.com/>

### Docker Command

<https://docs.docker.com/engine/reference/builder/#usage>

### Docker Compose command

<https://docs.docker.com/compose/reference/up/>

### Redis Command

<https://redis.io/commands>

### Redis npm

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

### Token vs cookies

<https://dzone.com/articles/cookies-vs-tokens-the-definitive-guide>

<https://scotch.io/bar-talk/why-jwts-suck-as-session-tokens>

### JWT

<https://jwt.io/>

<https://github.com/auth0/node-jsonwebtoken>

### Caching

<https://www.freecodecamp.org/news/the-hidden-components-of-web-caching-970854fe2c49/>

<https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching>

<https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers>

### CI/CD Tools

<https://code-maze.com/top-8-continuous-integration-tools/>

## Summary what we have done

|  |
| --- |
|  |

|  |
| --- |
|  |