## 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.

|  |
| --- |
|  |

## 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/>

## 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/>