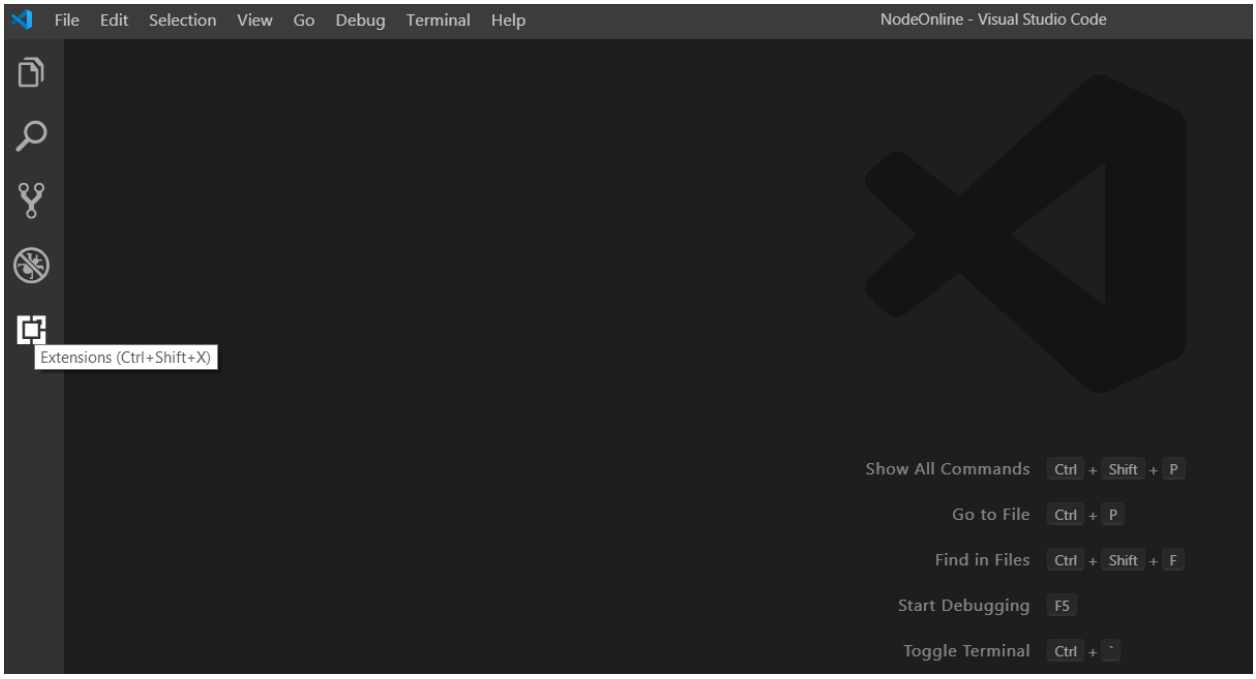
DEEP DIVE!

JAVASCRIPT, NODE.JS

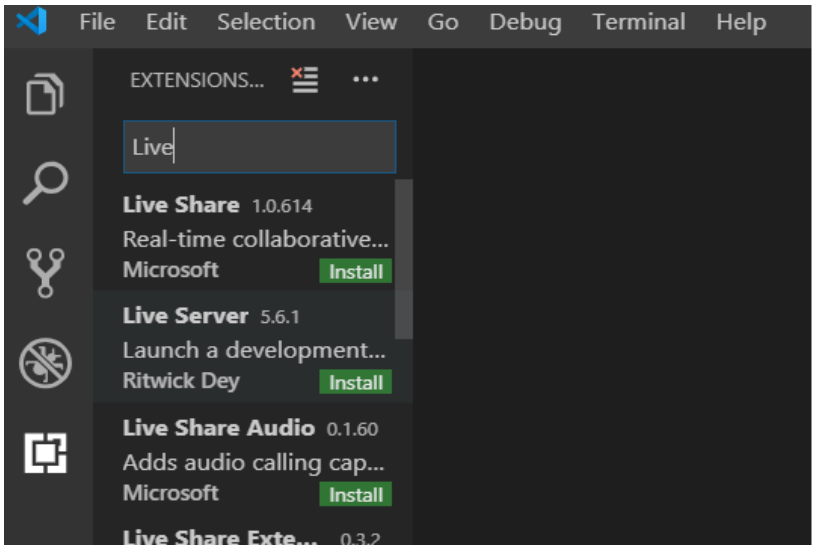
Samurai version

Win Htut (*Green Hackers Team*)

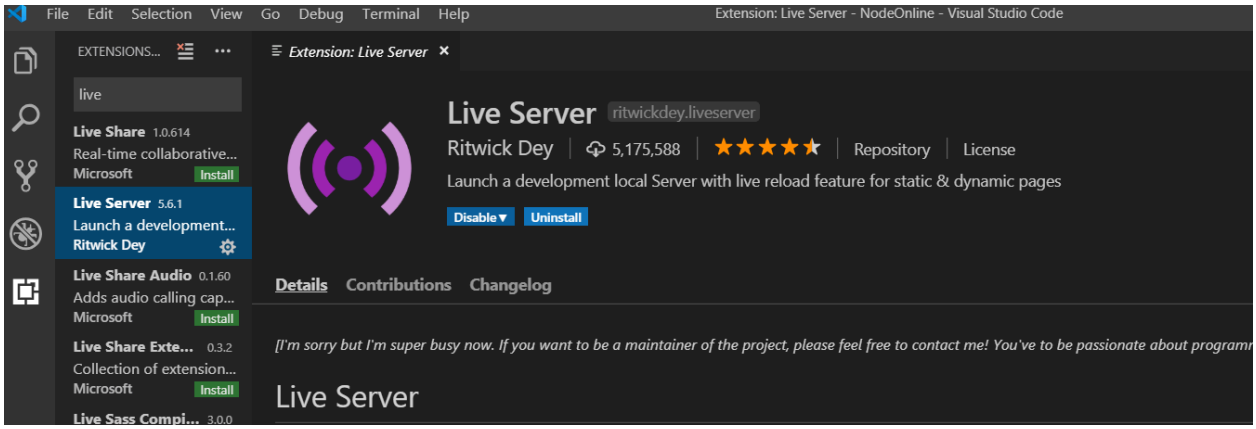
JavaScript ကို စတင္ေရးသားဖို႕အတြက္ Code Editor လိုအပ္ပါမယ္ ။ Visual Studio Code , Atom ,Sublime Text မိမိ နွစ္သက္ရာတို႕အျပင္ အျခားေသာ မိမိတို႕ နွစ္သက္ရာ editor တို႕ကို သံုးနိုင္ပါတယ္ ။ https://code.visualstudio.com/ ၌ Visual Studio Code ကိုေတာ့ ေဖာ္ျပပါ link မွာ download ဆြဲျပီး သံုးနိုင္ပါတယ္ ။ visual studio code ကို download ဆြဲျပီး install လုပ္ထားပါ ။ ထို႕ေနာက္ Live Server , Material Icon Theme , and Monokai ++ extension တို႕ကိုလည္း install လုပ္ေပးပါမယ္ ။ ထိုသို႕ လုပ္ေပးရန္ extension icon ေအာက္ပါ ပံု အတိုင္း နွိပ္ပါ ။



ျပီးလွ်င္ Live Server ကို ေရးပါ ။ live server ေပၚလာလွ်င္ တစ္ခ်က္နွိပ္ပါ သို႕မဟုတ္ တစ္ခါတည္း install လုပ္နိုင္ပါသည္ ။

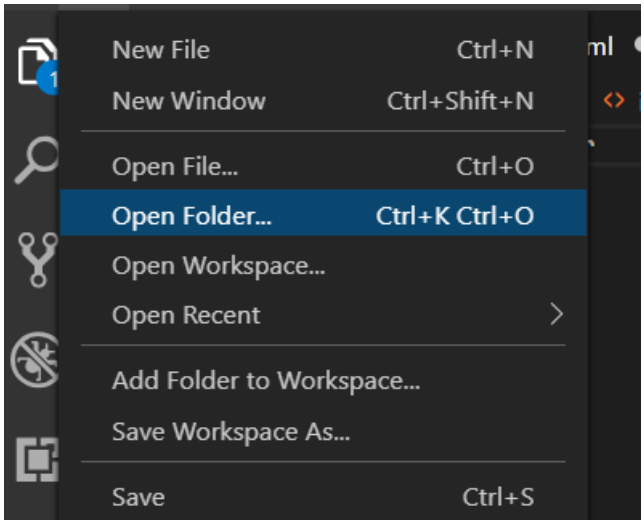


Live Server ကို ေအာက္ပါ ပံုု အတိုင္ုး install လုပ္ပါ ။

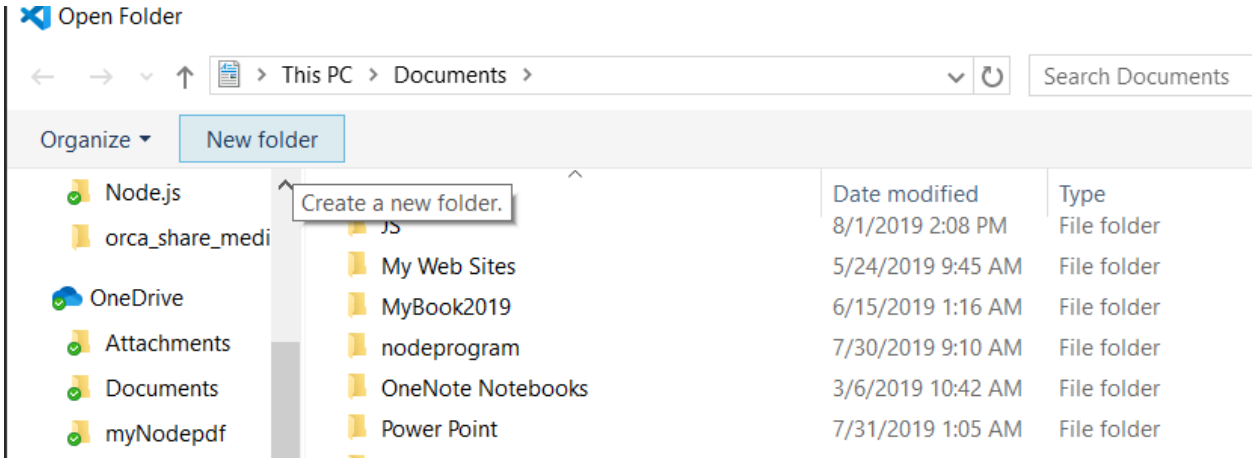


Live server ကို install လုပ္ျခင္ုးမွာ browser တြင္ မိမိ တိုု႔ေရးလိုက္ေသာ javascript program ကို အလြယ္တံုးနွင့္ အလွ်င္ျမန္ဆံုး run နုိင္ရန္ ျဖစ္သည္ ။ ထို႔ေနာက္ Material Icon Theme and Monokai ++ တို႔ကိုလည္ုး extension မွာ ရွာျပီး install လုပ္ထားေပးပါ ။

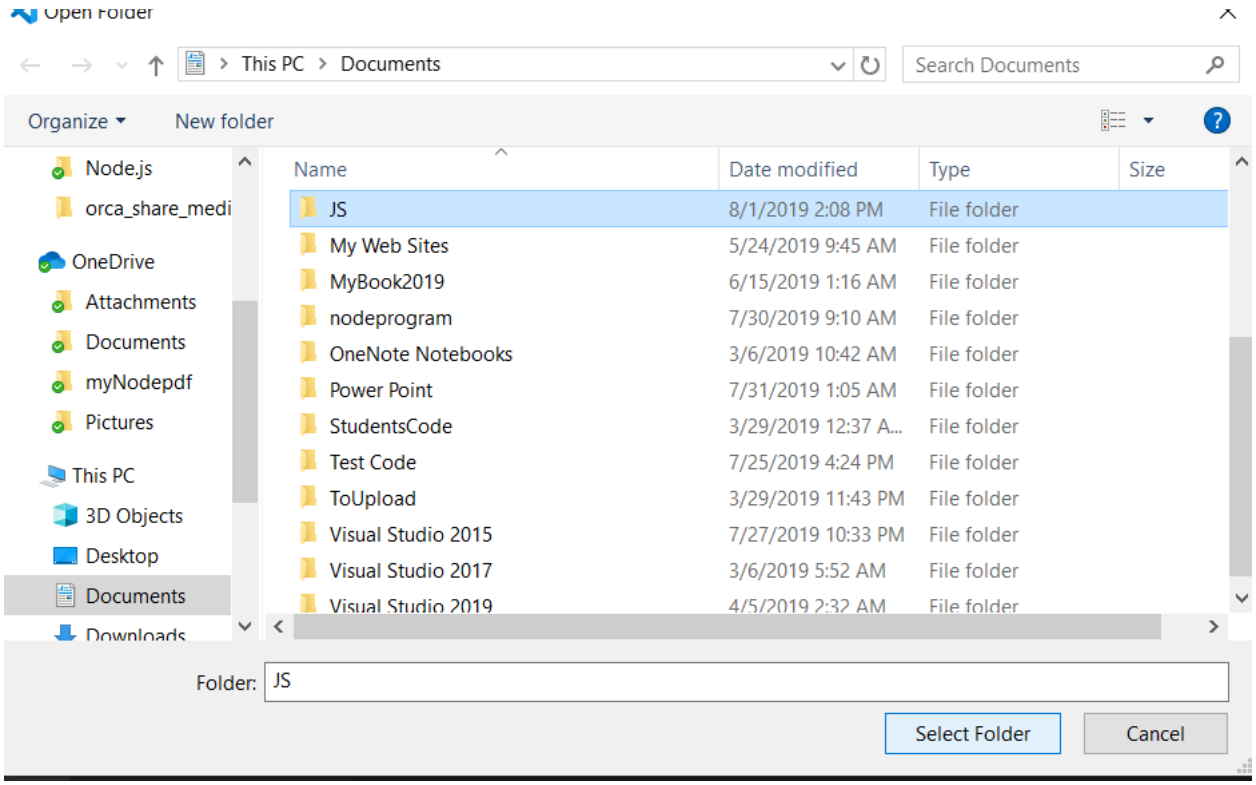
ထို႔ေနာက္ File ထဲကိုသြားပါ ျပီးလွ်င္ Open Folder ကို နွိပ္ပါ ။ ေအာက္ပါ ပံုုထဲ က အတိုင္ုးျဖစ္သည္ ။



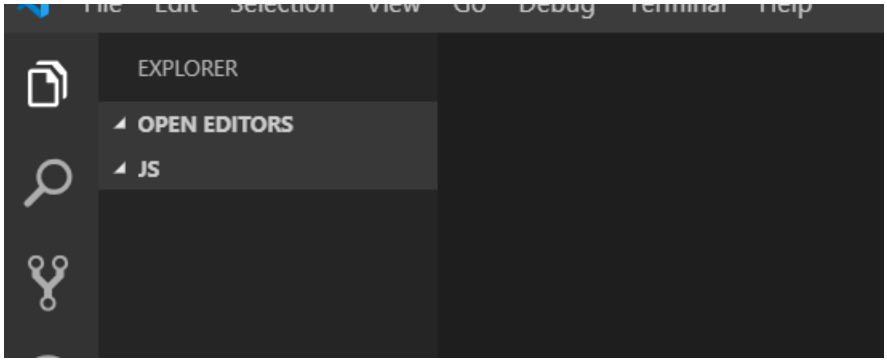
ေပၚလာေသာ window form တြင္ New Folder ကို နွိပ္ပါ ။ ထို folder name ကို JS ဟု နံမည္ ေပးပါ ။ မိမိ ျကိုက္နစ္သာ ရာ ေပးနိုင္ပါသည္ ။



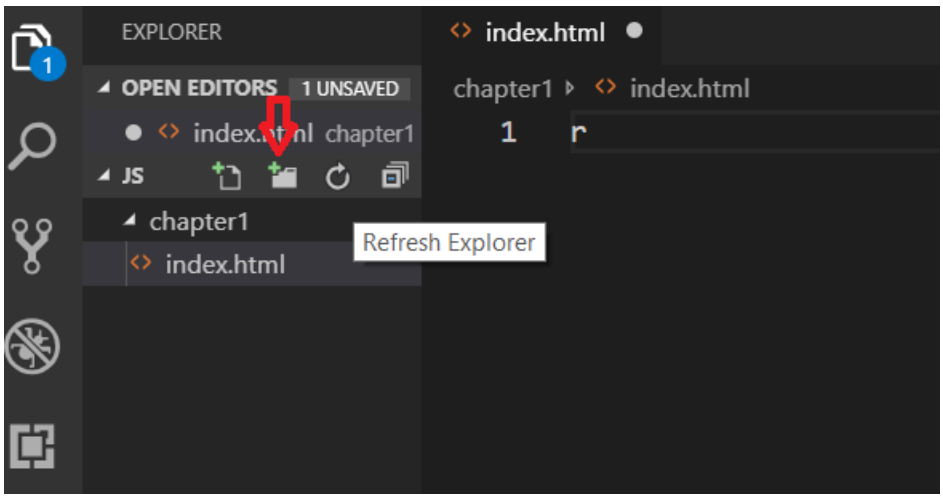
ျပီးလွ်င္ JS file ကို select (တစ္ခ်က္ထိ) လုပ္ပါ ထို႕ေနာက္ ေအာက္ဆံုးမွ Select Folder ကို ႏွိပ္ပါ။



ျပီးလွ်င္ ေအာက္ပါ ပံုထဲက အတုိင္း JS ဆိုသည့္ folder ေလး တစ္ခုကို ျမင္ရပါမည္ ။

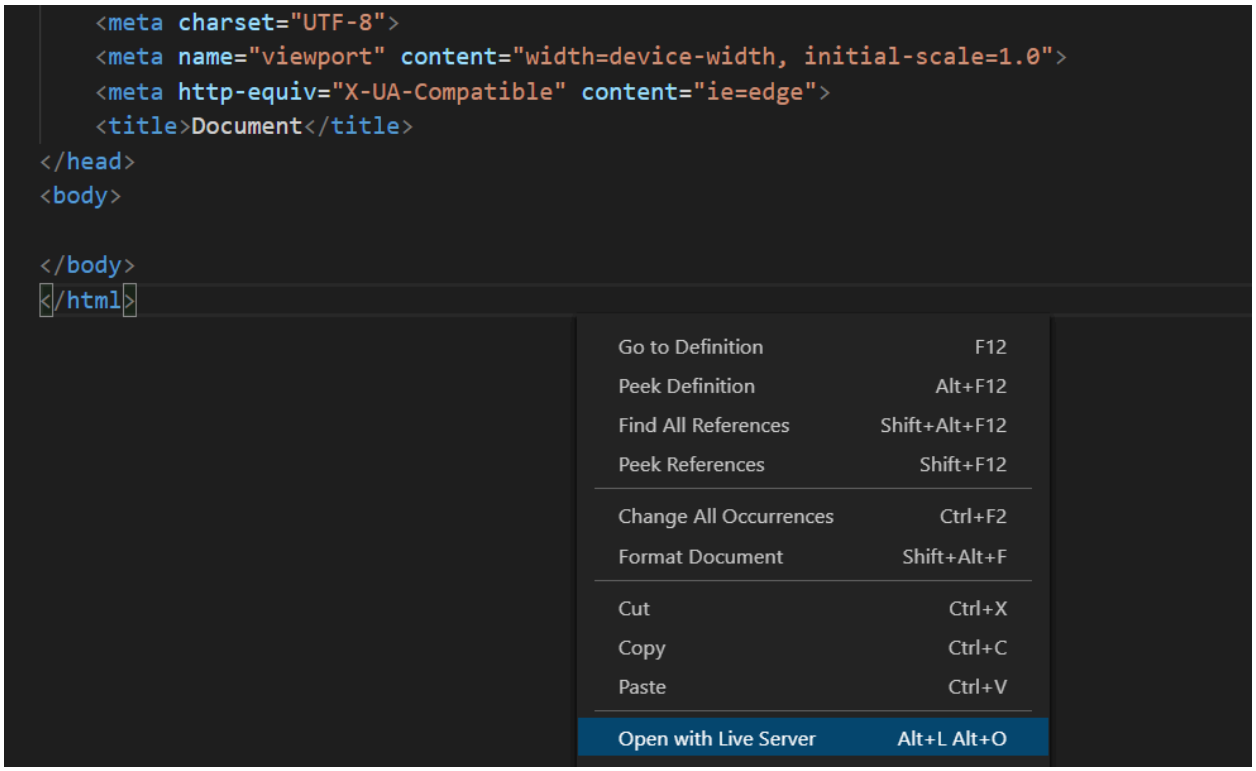


ထို folder ထဲတြင် ပထမဆံုး folder တစ္ခု ထပ္ တည္ေဆာက္ပါ ။ နံမည္ကို chapter1 ဟု ေပးပါ ။ မိမိ ျကိုက္ရာစသ္ကာ နံမည္ကို ေပးနုိင္ပါသည္ ။ ထို chapter1 folder ထဲတြင် index.html ဆိုသည့္ html file တစ္ခု ကို တည္ေဆာက္ပါ ။ File directory မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။



ထို႕ေနာက္တြင္ေအာက္ပါ ပံု အတုိင္း doc ဟုေရးပါ ျပီးလွ်င္ enter နွိပ္လိုက္ပါ ။ HTML စာသားမ်ား ေပၚလာသည္ကို ျမင္ရပါမည္ ။

Right click နှိပ်ပါ၊ပီးလ်ွင် Open with live Server ကို နှိပ်လိုက်ပါ ။

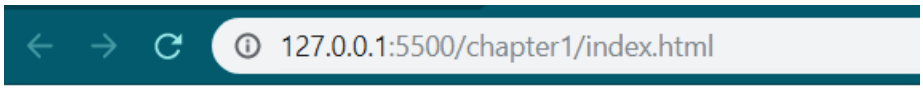


၊ပီးလ်ွင် browser တစ်ခု ပြင့်လာပါမည် ။ access ေတာင်းလာပါက access လုပ်ေပးလိုက်ပါ ။ ေပၚလာေသာ browser သည် page အလြတ်၊ကီးသာ ၊ဖစ်သည် ။ မည်သည့် စာသားမွ မပါဝင်ေသးပါ။ မိမိ ေဖာ်၊ပလိုေသာ စာသားမ်ား ပါဝင်လာရန် code အနည်းငယ် ၊ဖည့်ေပးရမည် ။

Body tag ထဲတြင် h1 element တစ်ခု ထပ်၊ဖည့်၊ပီး စမ်း၊ကည့်နိုင်ပါသည် ။ HTML မ်ားကို ေတာ့ ယခု အပိုင်းမွာ အေသးစိတ် ရှင်းလင်းမည် မဟုတ်ပါ ။ အနည်းငယ်သာ HTML အေ၊ကာင်းကို ေရးသားမည် ၊ဖစ်၊ပီး javascript ကိုသာ ဦးတည် ေရးသား သြားပါမည် ။

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Document</title>
</head>
<body>
    <h1>Hello JS world</h1>
</body>
</html>
```

Code ထပ်ျဖည့်ျပီး program ကို save လိုက်သည် နှင့် တစ်ျပိုင်နက် browser တြင် စာသား်မ်ား ေျပာင်းလဲ သြားသည်ကို ျမင်ရပါမည် ။ မိမိတို့ install လုပ်ထား်ေသာ live server extension ေျကာင့် ျဖစ်သည် ။



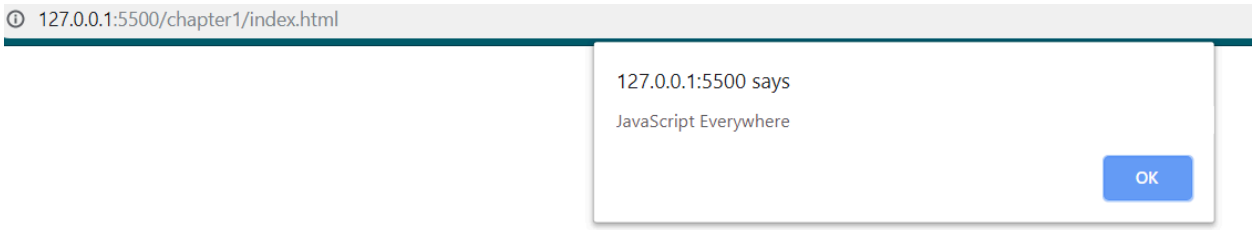
Hello JS world

Adding a script to the HTML file

HTML file ထဲမွာ script တစ်ခု ထည့်ေရးပါမည် ။ script ထည့်ေရးရမည့်ေနရာက head element ထဲမွာလည်း ရသလို body element ထဲမွာလည်း ထည့်ေရးနိုင်ပါတယ် ။

```
<script>
  alert('JavaScript Everywhere');
</script>
```

အထက်ပါ code မ်ားအား ထည့်ေရးျပီး program ကို save လိုက်ပါ ။ ထို့ေနာက် browser တြင် alert တစ်ခု တက်လာသည်ကို ျမင်ရပါမည်။ ထို alert တြင် မိမိတို့ ထည့်ေပးလိုက်ေသာ စာသား်မ်ားကို ျမင်ရပါမည် ။ alert function သည် browser တြင် alert တစ်ခု ျပေပးရန် အတြက် အသံုးျပဳပါသည်။ ထို function အဆံုးတြင် semicolon ကို သံုးထားပါသည် ။ semicolon ကိုသံုးျခင်းသည် statement တစ်ခု ဆံုးျပီျဖစ်ေျကာင်း ေဖာ်ျပလိုေသာ အခါ တြင် အသံုးျပဳပါသည် ။ C / C++ ,Java programming မ်ားတြင်ေတာ့ semicolon ကို မျဖစ် မေန အျဖစ်အသံုးျပဳရပါသည် ။ သို့ေသာ် javascript တြင်ေတာ့ မလိုအပ်ပါဘူး ။ statement မ်ား function မ်ားအဆံုးတြင် semicolon အသံုးျပဳေပးျခင်းကေတာ့ program မ်ားလာတဲ့အခါ trace လုပ်ရ လြယ်ကူေစျပီး ေကာင်းမြန်သည့် အေလ့အထ တစ်ခု ျဖစ်ပါသည်။



Adding JavaScript File to HTML page

JavaScript file တစ်ခု ကို chapter1 folder ေအာက္တြင္ တည္ေဆာက္ပါမည္ ။ ထိုသို့ တည္ေဆာက္ေပးဂွန္ chapter1 folder ကို right click ျဖင့္ new file ကို ေရြးျပီး တည္ေဆာက္နုိင္ပါသည္ ။ ျပီးေနာက္ထို့file ကို app.js ဟု နံမည္ ေပးလုိက္ပါမည္ ။ နံမည္ ေပးျပီး ေနာက္ထို့ file တြင္ alert function ကိုသုံးျပီး မိမိတို့ ေရးသားလိုသော စာသား အနည္းငယ်ကို အေက်ပါ အတိုင်ေရးသားပါ။

```
alert("hello coder .");
```

ထိုသို့ေရးသားျပီးေနာက်တွင် app.js ဆိုသည့် js file အား html code များထည့်တွင် add ေပးရန် လိုအပ်ပါသည်။ html file ထဲမှ body tag ထဲတွင် အေက်ပါ စာသားေလးကို ထည့်ေပးရပါမည်။

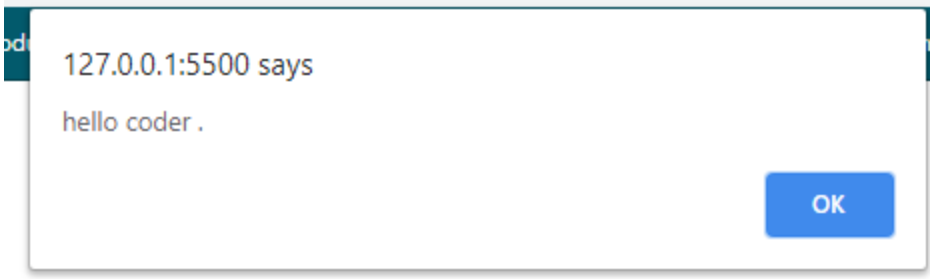
```
<script src="app.js">
```

ထို့ေကြာင့် index.html ဆိုသည့် file ထဲတွင် အေက်ပါ code အားလုံး ရှိသင့်ပါသည်။

```
<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>JS Crash Course</title>

</head>

<body>

    <header>

        <h1>Js Crash Course With Update Features</h1>

    </header>
```

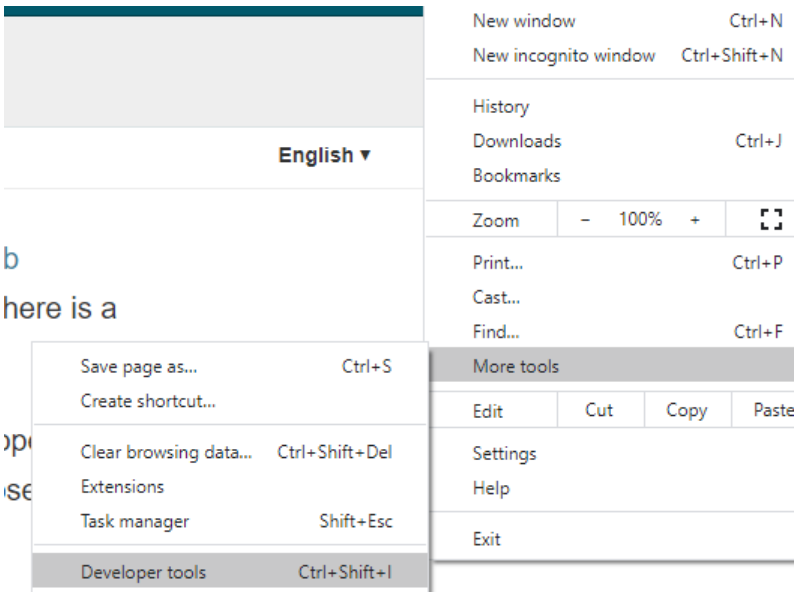
```
<script src="app.js">


</script>

</body>

</html>
```

ထို့နောက် program ကို save လိုက်ပါ ပြီးလျှင် မိမိတို့ web browser ၌ အောက်ပါအတိုင်း alert လေး တက်လာသည်ကို မြင်ရပါမည်။

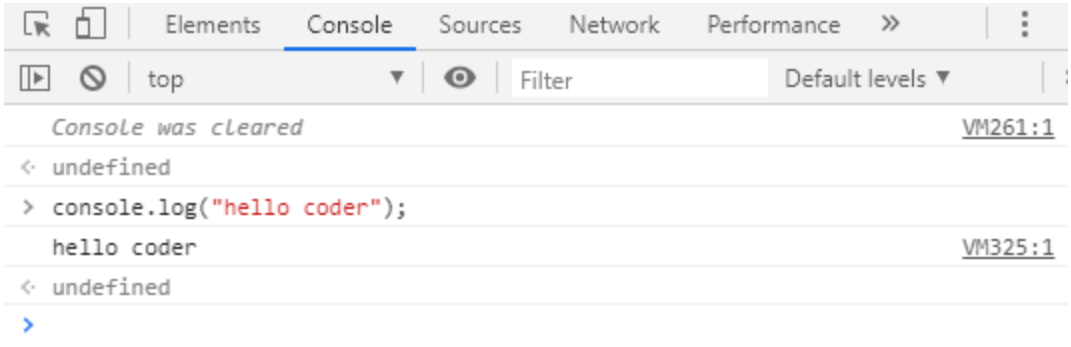


Console Object

JavaScript ရဲ့ console object သည် browser တွင်ပါဝင်သော debugging console ကို အသုံးပြု နိုင်စေရန် ဖြစ်သည်။ console object ထဲတွင် methods များစွာ ပါဝင်လှသည်။ ဥပမာ log method သည် console တွင် စာသားများကို ဖော်ပြရာတွင်သုံးပါသည်။ console ကိုသုံးနိုင်ရန် browser ရဲ့ ညာဘက် ထောင့်တွင် ရှိသော setting ထဲမှ more tools -> developer tools ထဲသို့ သွားရမည်။



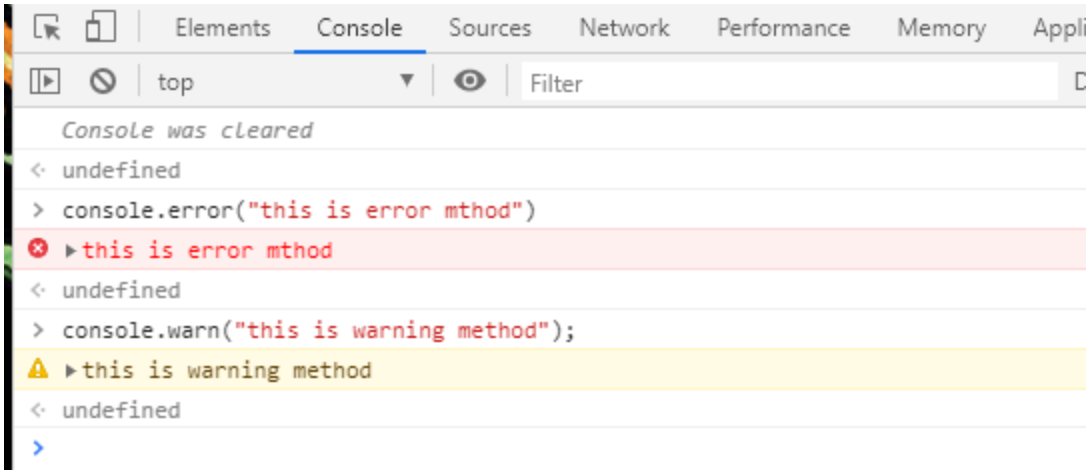
ထို့နောက် ပေါ်လာသော console ထဲတွင် အောက်ပါ အတိုင်း console.log(“hello coder”); ဟု ရေးကြည့်ပါ hello coder ဆိုသည့် စာသားလေး ပေါ်လာသည်ကို မြင်ရပါမည်။



error method

console.error(“This is an error”)

အထက်ပါ အတိုင်း error ကို ပြလိုသော အခါတွင်လည်း error method ကိုသုံးနိုင်သည်။



warn method သည်လည်း ထိုနည်း အတိုင်းပင်ဖြစ်သည် warning လုပ်လိုသော အခါတွင် အသုံးပြုပါသည်။ console object နှင့်ပတ်သက်သည့် method များကို ယခု link တွင် လေ့လာနိုင်သည်။

Data Types

JavaScript မှာ variable တွေကို ကြေငြာရာတွင် keyword သုံးခုကို အသုံးပြုပါတယ်။ var , let ,const တို့ဖြစ်ပါတယ်။ ES6 နောက်ပိုင်းတွင် var အစား let ကိုသာ အသုံးများလာကြပါတယ် အဘယ်ကြောင့် ဆိုသော် var ရဲ့ အားနည်းချက်များကို let က ပြန်လည် ကောင်းမွန်အောင် လုပ်ထားတဲ့ အတွက် ဖြစ်ပါတယ်။ အောက်ပါ program နှစ်ပုဒ်တွင် let and var ကွဲပြားပုံကို သိနိုင်ပါသည်။

var keyword

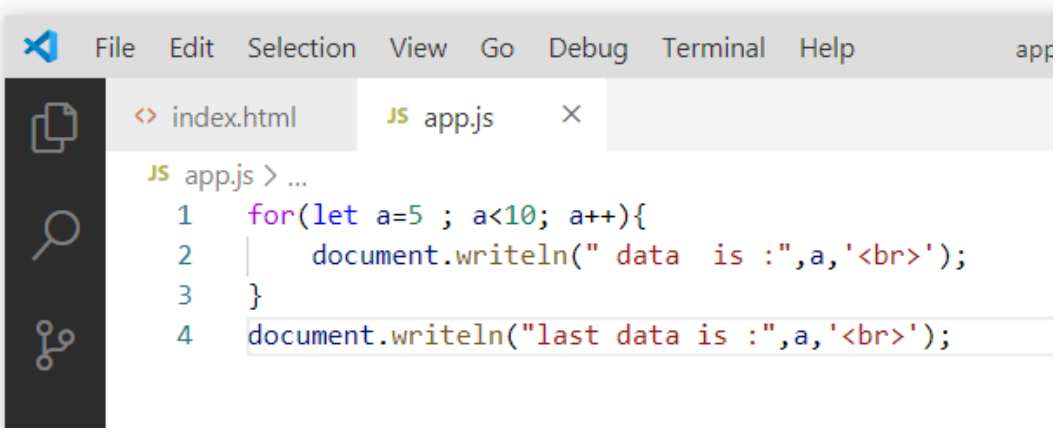
```
for(var a=5 ; a<10; a++){

    document.writeln(" data  is :",a,'<br>');

}

document.writeln("last data is :",a,'<br>');
```

Js Crash Course With Update Features

data is :5
data is :6
data is :7
data is :8
data is :9
last data is :10

ဖော်ပြပါ program ကို run ကြည့်သော အခါတွင် အောက်ပါ ပုံထဲက အတိုင်း output အချို့ကို မြင်ရပါမည်။ for statement အပြင်ဘက်မှ နေပြီး a ဆိုသည့် variable ကို print ထုတ်ကြည့်သည့် အခါ 10 ကို မြင်ရပါမည်။ var နေရာတွင် let ကို သုံးပြီး အောက်ပါ အတိုင်း ပြန် run ကြည့်ပါက 10 မပေါ်တော့ တာကို မြင်ရပါမည်။

data is :5
data is :6
data is :7
data is :8
data is :9



Difference between var and let

Projects များ ရေးသားရာတွင် var and let ကို globally အနေနဲ့ အတူတကွ ရေးသားကြသော်လည်း let သည် global window object မဟုတ်ပါဘူး အောက်ပါ example ကို ကြည့်ပါ။

```
var varVariable= "this is var variable";
```

```
let letVariable="this is let variable";


console.log(window.varVariable,"\n");

console.log(window.letVariable);
```

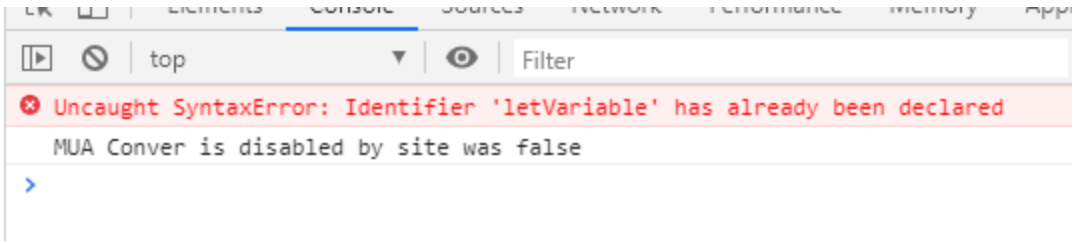
```
this is var variable
undefined
MUA Conver is disabled by site was false
>
```

var keyword ကိုသုံးပြီး ကြေငြာထားသော စားသားသည် ပေါ်လာသော်လည်း let ဖြင့် ကြေငြာထားသော စာသားကိုတော့ မသိပါဘူး။ ထို့ကြောင့် let variables ကို for loop, while loop သို့မဟုတ် statement တွေရဲ့ scope အတွင်မှာသာ သုံးကြပါတယ်။ အထက်တွင် looping နှစ်ခု ပတ်ထားသော program တွင်လည်း var နဲ့ ကြေငြာထားသော variable ကို သူ့ scope အပြင်ဘက်က မြင်နိုင်သောလည်း let နဲ့ ကြေငြာထားရင်တော့ သူ့ scope အပြင်ဘက် မမြင်နိုင်ပါဘူး။

Redeclaration

let variables ကို နောက်တစ်ကြိမ် declare လုပ်လို့ မရပါဘူး သို့သော် var ကိုတော့ နောက် တစ်ကြိမ် ပြန်ကြေငြာနိုင်ပါတယ်။ အောက်ပါ code များ run ပြီး browser console တွင် ကြည့်နိုင်ပါသည်။

```
var varVariable= "this is var variable";

let letVariable="this is let variable";


var varVariable= " again this is var variable";

let letVariable=" again this is let variable";
```



JavaScript const: Declaring Constants in ES6

ES6 မှာ const ဆိုတဲ့ keyword ကိုသုံးပြီးတော့ constant variable ကို ကြေငြာတာ ပါဝင်လာပါတယ်။ const ကို value တစ်ခုအား read-only အနေဖြင့် အသုံးပြုလိုသော အခါမှာ သုံးပါတယ်။ naming convention အရ const များကို ကြေငြာရာတွင် name ကို စာလုံး အကြီးများဖြင့် သာ အသုံးပြုပါတယ်။ let keyword ကိုသုံးထားတဲ့ variable တွေဟာ mutual ဖြစ်ပါတယ် ဆိုလိုတာက သူ့ value တွေကို ပြန် ချိန်းနိုင်ပါတယ်။သို့သော် const သည် ပြန်ချိန်းလို့ မရပါဘူး။ const keyword ကိုသုံးလျှင် နောက်ထပ် သိထားရမည့် အချက် သည် const ကို ကြေငြာလိုက်သည်နှင့် တစ်ပြိုင်နက် value ကို assign လုပ် ပေးရပါမည်။ ထို့သို့ ချက်ချင်း assign မလုပ်ပဲ နောက်တစ်ကြောင်းမှ assign လုပ်လျှင် SyntaxError ဖြစ်ပါမည်။

```
const CON = 0.99;

CON = 0.1 ; //error

const CON; //error
```

```
▼Uncaught TypeError: Assignment to constant variable.
    at app.js:2
  (anonymous) @ app.js:2

MUA Conver is disabled by site was false
```

JavaScript const and Objects

const ကို object အနေဖြင့် အသုံးပြုမည်ဆိုလျှင်တော့ value ရဲ့ properties တွေကိုတော့ object ကိုသုံးပြီး ချိန်းလို့ရပါတယ်။ အောက်ပါ program ကို run ကြည့်ပါက ကောင်းမွန်စွာ အလုပ် လုပ်သည်ကို မြင်ရပါမည်။

```
const CAT={age: 1};

CAT.age=2;

document.writeln(CAT.age);
```

အောက်ပါ အတိုင်းရေးလျှင်တော့ error တက်ပါမည်။

```
const CAT={age: 1};

CAT.age=2;

document.writeln(CAT.age);
```

```
CAT={age:2}; //error
```

အကယ်၍ const object တစ်ခုမှာ value ရဲ့ Properties တွေကို မပြောင်းလဲလိုဘူးဆိုရင်တော့ Object.freeze() ကိုသုံးလို့ရပါတယ်။

```
const CAT=Object.freeze({age: 1});

CAT.age=2;

document.writeln(CAT.age);
```

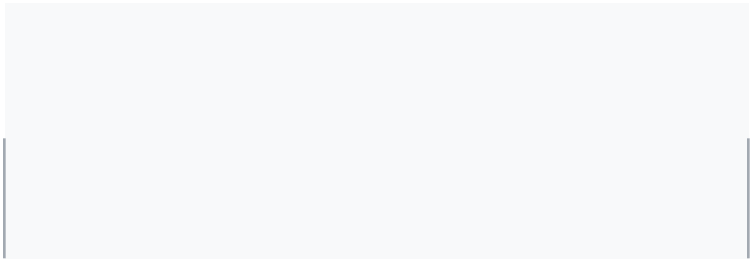
အထက်ပါ program ကို run ကြည့်ပါက output အနေဖြင့် 1 ကိုသာရရှိပါမည်။

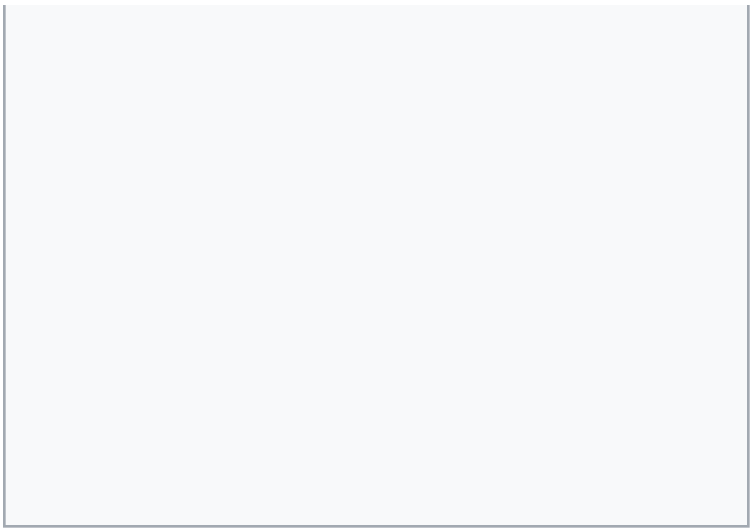
Adding new properties to Const Object

const နဲ့ သုံးထားတဲ့ object တစ်ခုရဲ့ properties တွေထဲမှာလည်း မိမိတို့ထပ်ထည့်လိုတဲ့ properties and value တွေကို ထပ်ထည့်နိုင်ပါသေးတယ်။ အောက်ပါ Program တွင် info ထဲ၌ state တစ်ခု ထပ်ထည့်ထားပါသည် ထို state ရဲ့ value အား myanmar ဟု ရေးပေးထားပါသည်။

```
const CAT=Object.freeze({age: 1,
   name:"winhtut",
   info:{ street:"nawayat",
        city:"pyinoolwin"},
   ph:123456
});

CAT.info.state='Myanmar';//no error

document.writeln(CAT.info.state);
```

အထက်ပါ program အား run ကြည့်ပါက error မတက်ပဲ Myanmar ဆိုသည့် စာသားထွက်လာသည်ကို မြင်ရပါမည်။





https://en.wikipedia.org/wiki/Node.js

https://nodejs.org/en/about/

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

console.log([data][, ...args])

stdout (standard output) အေနျဖင့္ ေဖာ္ျပလိုေသာ အခါမ်ိဳးမွာ အသံုးျပဳပါတယ္ newline ပါထားျပီးသားျဖစ္တဲ့ အတြက္ ေအာက္ ကို တစ္ေျကာင္းဆင္းေပးသြားမွာပါ ။ console.log ထဲတြင္ multiple argument မ်ားကိုလည္း ထည့္ျပီး run နုိင္ပါတယ္ ပထမဦးဆံုး argument ကိုေတာ့ ပထမ ဦးဆံုး print လုပ္ ေပးမွာ ျဖစ္ပါတယ္ ။

```
console.log("Hello world from nodejs");
```

node program တစ္ခုဖိုင္ကို run ရန္ node <filename> ဆိုျပီး run ေပးရမည္။

```
const count = 5;
console.log('count: %d', count);
// Prints: count: 5, to stdout
console.log('count:', count);
// Prints: count: 5, to stdout
```

Number ေတြကို output ထုတ္ရာမွာ c programming မွာကဲ့သို့ decimal ေတြကို ကိုယ္စားျပဳေသာ %d ဆိုတာကို ထည့္ျပီး run နိုင္သလို မထည့္လဲလည္း run နိုင္ပါတယ္ ။ argument မ်ားကို ေရးရာတြင္ ‘ ‘ ထဲတြင္ ထည့္ရန္ လုိအပ္ပါသည္ ။ variable name မ်ားကိုေတာ့ ထည့္စရာမလိုအပ္ပါ။ Variable ေတြကို var ဆိုသည့္ data type ျဖင့္လည္း ေျကျငာနိုင္သလို const ျဖင့္လည္း ေျကျငာနိုင္ပါသည္ ။ var သည္ value မ်ားကို ထပ္ပါ ထပ္ပါ assign လုပ္နိုင္ျပီး const ကိုေတာ့ တစ္ခါ သာလွ်င္ assign လုပ္နိုင္ပါသည္ ။ ထို့ေျကာင့္ တစ္ျကိမ္သာ assign လုပ္ရန္လုိအပ္ေသာ အခါတြင္ const ကိုသံုးပါသည္ ။ ဥပမာ package name မ်ားကို ေခၚသံုးရန္ ေျကျငာေပးရေသာ အခါတြင္ const ကိုသာ အသံုးျပဳသင့္ပါသည္ ။ example code မ်ားကို ေအာက္တြင္ ေလ့လာနိုင္ပါသည္ ။

```
const number=10;

console.log(number);
var vNumber=20; ပထမအျကိမ္အသံုးျပဳျခင္း
console.log(vNumber)
vNumber=30; ဒုတိယ အျကိမ္ အသံုး ထပ္ျပဳျခင္း( ထပ္ပါ ထပ္ပါ အသံုးျပဳနုိင္ပါသည္)
console.log(vNumber)
```

အထက္ပါ program ကို run ဖို့ console တြင္ node app.js ဟုေရးပါ ။ ထို့ေနာက္ output ထြက္လာျပီး မွန္ကန္စြာ အလုပ္ လုပ္သည္ကို ျမင္ရပါမည္ ။ ေအာက္ပါ example code မ်ားကို run ျကည့္ရင္ေတာ့ error တက္သည္ကို ျမင္ရမွာပါ အဘယ္ေျကာင့္ဆိုေသာ့ const ျဖင့္ ေျကျငာထားေသာ number ကို ေနာက္ထပ္တစ္ျကိမ္ value assign လုပ္ေသာေျကာင့္ျဖစ္ပါသည္။ example code ကို ေအာက္တြင္ ေဖာ္ျပထားပါသည္ ။

```
const number=10;

console.log(number);
var vNumber=20;
console.log(vNumber)
vNumber=30;
console.log(vNumber)
```

```
number = 20; // error
console.log(number)
```

Importing Node.js Core Module

File System

Node.js မွာ file system နဲ့ပတ္သက္ျပီး ေျမာက္မ်ားစြာ ရွိပါတယ္ဆိုတဲ့အခ်က္မွ

- fs.writeFileSync(file, data[, options]) ကိုေဖာ္ျပလိုပါတယ္။

```
const fs=require('fs')
fs.writeFileSync('notes.txt','This file was created by Node.js')
```

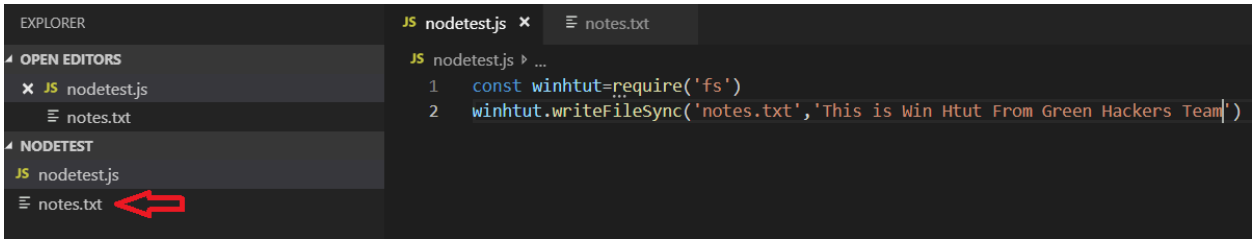
အထက္ပါ program သည္ file တစ္ခု ကို ဖြင့္ျပီး ထို file ထဲတြင္ second argument မွ စာသားမ်ားကို ေရးသားပ့ဲ ျဖစ္ပါသည္ ။

```
const fs=require('fs')
```

require('fs') သည္ fs ဆိုသည့္ module ကို ယခု program ထဲတြင္ ေခၚသံုးရန္ အတြက္ ျဖစ္ပါသည္ ။ ထို fs ဆိုသည့္ module ကို ေခၚသံုးျခင္းမွ သာလ်ွင္ writeFileSync စသည့္ method or function ကိုေခၚသံုးနိုင္မွာ ျဖစ္ပါတယ္ ။ const fs သည္ const ျဖင့္ fs ဆိုသည့္ variable တစ္ခုကို တည္ေဆာက္ လိုက္ျခင္း ျဖစ္ပါသည္ ။ ထို နည္းအားျဖင့္ fs (file system module) ကို program ထဲတြင္ fs ဆိုသည့္ variable ကို သံုးျပီးေခၚဆို နုိင္မွာ ျဖစ္ပါတယ္ ။

writeFileSync သည္ file တစ္ခုကို ဖြင့္ျပီး ထို file ထဲတြင္ မိမိ ေရးသားလိုေသာ စာသားမ်ားကို ေရးနိုင္ရန္ အသံုးျပဳေသာ function ျဖစ္ပါသည္ ။ ေအာက္ပါ အတိုင္းလည္း ေရးသားနုိင္ပါသည္ ။

```
const winhtut=require('fs')
winhtut.writeFileSync('notes.txt','This is Win Htut From Green Hackers Team')
```



အထက္ပါ program ကို run ျကည့္ခ်င္ notes.txt ဆိုသည့္ text file တစ္ခု ျဖစ္လာသည္ကို ျမင္ရမွာပါ ။ ထို txt file ကို ဖြင့္ျကည့္ခ်င္ second argument ေနရာတြင္ ေရွးသားထားေသာ စာသားမ်ားကို ျမင္ရမွာ ျဖစ္ပါတယ္။

appendFileSync

txt file တစ္ခုထဲသို့ data မ်ား append လုပ္ခ်င္တဲ့ အခါမ်ိဳးမွာ အသံုးျပဳပါတယ္ ။ first argument ေနရာတြင္ append လုပ္ခ်င္ေသာ file name ကိုေရးသားျပီး ဒုတိယ ေနရာတြင္ append လုပ္ခ်င္ေသာ data ကိုေရးေပးရပါသည္ ။

```
const winhtut=require('fs')
//winhtut.writeFileSync('notes.txt','This is Win Htut From Green Hackers Team')

winhtut.appendFileSync('notes.txt',' This is appending to file')
```

ေအာက္ပါ အတိုင္းလည္း မိမိ ေဖာ္ျပလိုေသာ data ကို ေဖာ္ျပနိုင္ပါသည္ ။

```
const winhtut=require('fs')
//winhtut.writeFileSync('notes.txt','This is Win Htut From Green Hackers Team')
var data=5+5;
winhtut.appendFileSync('notes.txt',data)
```

Importing Our Own Files

Project ျကီးလာတာနဲ့ အမွဲ code ေတြကို maintain လုပ္ဖို့ အရမ္းခက္ခဲ လာပါတယ္ ထို့ျပသနာေတြကို ေျဖ ရွင္းဖို့အေကာင္းဆံုး နည္းလမ္းကေတာ့ code file ေတြ အားလံုးကို ခြဲထားျပီး အေရးျကီးတဲ့ main code file တစ္ခုကေန လိုအပ္သလို ထိန္းခ်ဳပ္သံုးစြဲျခင္းက error မ်ားစြာကို ေရွာင္ရွားနိုင္ပါတယ္ ။ ပထမဆံုး file နွစ္ခုျဖင့္ပါ app.js and appp.js ဆိုတဲ့ file နွစ္ခု။

Appp.js ဆိုတဲ့ code file ထဲမွာ ေအာက္ပါ code မ်ားကို ေရးပါ

```
var methods={};

methods.control=function(){
  console.log('Control System');
}
methods.sensor=function(){
  console.log('Sensing System');
}
exports.data=methods;
```

ရွင္းလင္းခ်က္။

```
var methods={}; //function ေတြကို ထိန္းဖို့ အတြက္ methods တစ္ခု
ေျကျငာထားလိုက္တာပါ

methods.control=function(){ // contorl function တစ္ခု စတင္ တည္ေဆာက္ထားတာပါ ။
  console.log('Control System');// output တစ္ခု ျပေပးတာပါ။
} // control function ဆံုးတဲ့ ေနရာ
```

```
methods.sensor=function(){ // sensor ဆိုတဲ့ function
တစ္ခုကိုစတင္ေျကျငာေပးထားပါတယ္။
  console.log('Sensing System'); // output တစ္ခု ျပေပးျခင္း
}
exports.data=methods;
```

အထက္မွာ တည္ေဆာက္ထားတဲ့ function နွစ္ခုကို methods နဲ့ control လုပ္ထားပါတယ္ ထို methods ကိုပဲ export လုပ္ေပးထားတာပါ ။ ထိုသို့ exports လုပ္ေပးထားမွ သာလွ်င္ အျခားေသာ file မ်ားကေန ျပဳးေတာ့ appp.js ထဲက instruction ေတြကို ေခၚယူသံုးစြဲ နိုင္မွာ ျဖစ္ပါတယ္ ။ export ေနာက္မွာ data ဆိုတဲ့ variable တစ္ခု ထည့္ေပးထားတာကို လည္း သတိျပဳပါ ။ appp.js ထဲက function တစ္ခုျခင္းစီကို သီးျခားကိုင္တြယ္ နိုင္ပါတယ္ ။

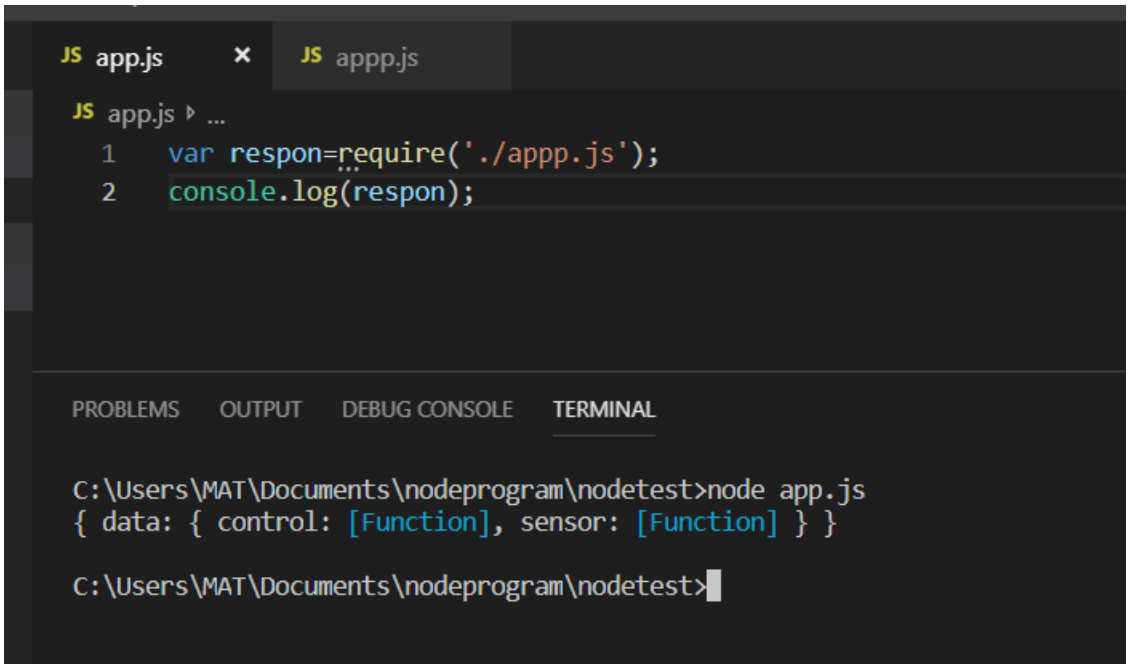
ေနာက္ code file တစ္ခု ျဖစ္တဲ့ app.js ကေနျပီး အထက္ပါ function ေတြကို လွမ္းေခၚသံုးပါမယ္ ။ ထို့ေျကာင့္ app.js ထဲမွာ ေအာက္ပါ code မ်ားကို ေရးသားပါ ။

```
var respon=require('./appp.js');
console.log(respon);
```

ရွင္းလင္းခ်က္။

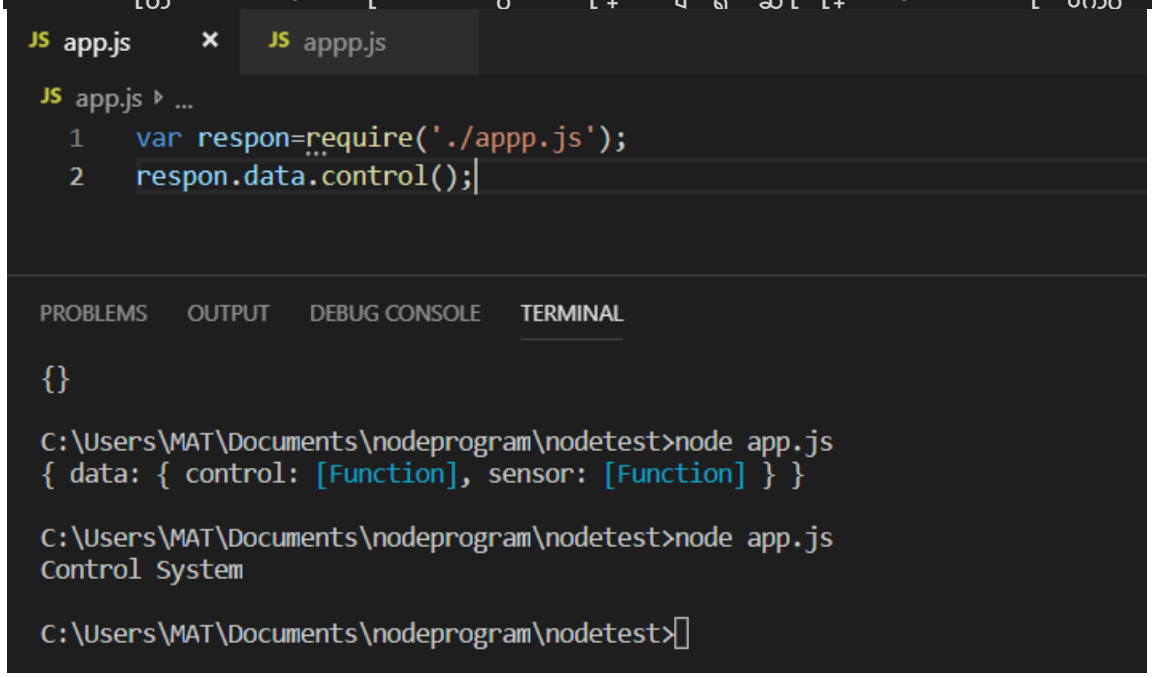
```
var respon=require('./appp.js'); // appp.js ဆိုတဲ့ file ထဲက code ေတြကို app.js ဆိုတဲ့ file ထဲမွာ
သံုးမယ္လို့ ေျကျငာ ေပးလုိက္တာပါ ။ respon ဆိုတဲ့ variable တစ္ခု လည္း ေျကျငာထားပါတယ္ ။ ထို့ေျကာင့္ respon သည္ appp.js ကို ကိုယ္စား ျပဳေနျခင္းျဖစ္ပါတယ္ ။
console.log(respon); // respon ကို output အေနနဲ့ ေဖာ္ျပျကည့္တာပါ ။
```

output ကို ေအာက္ပါ အတိုင္း ျမင္ေတြ့ရမယ္ဆိုရင္ ကြ်န္ေတာ္တို့ လုပ္တဲ့ step အားလံုး မွန္ကန္ပါတယ္ ။



Function တစ္ခုခ်င္းစီကို Access ျပဳလုပ္ျခင္း ။ appp.js ထဲမွ function မ်ားကို app.js ထဲမွ access လွမ္းလုပ္မွာ ျဖစ္ပါတယ္ ။

```
var respon=require('./appp.js');
respon.data.control(); // respon ထဲက data ထဲကမွ control ဆိုတဲ့ function ကို လွမ္းျပီး access လုပ္တာပါ ။ output ကို ေအာက္ပါ အတိုင္း ျမင္ရမယ္ဆို ရင္ step အားလံုး မွန္ကန္ပါတယ္ ။
```



Function ကို parameter မ်ားျဖင့္ ေခၚယူျခင္း ။

Appp.js ထဲမွာ ေအာက္ပါ code မ်ားကို ေရးသားပါမယ္ ။

```
var methods={};
var output=0;

methods.sumNumber=function(a,b){
  output=a+b;
  return output;
}
methods.circleCircumference=function(radius){
  output=2*Math.PI*radius;
  return output;
}
methods.areaOfRectangle=function(a,b){
  output = a*b;
  return output;
}
exports.data=methods;
```

ထို႕ေနာက္ app.js ထဲမွာ ေအာက္ပါ code မ်ားကို ေရးသားပါမယ္ ။

```
var respon=require('./appp.js');
console.log(respon)
```

ထို႕ေနာက္ program ကို run ျကည့္ရုံ အရင္ဆုံး ေအာက္ပါ အတိုင္း ျမင္ရပါမယ္ ။

```
{ data:
  { sumNumber: [Function],
    circleCircumference: [Function],
    areaOfRectangle: [Function] } }
```

ဒါဆိုရင္ function ေတြ ရွိေနသည္ကို ျမင္ရပါမယ္ ။ program first step ေအာင္ျမင္ပါတယ္ အခု ဆက္လက္ျပီး function တစ္ခု ျခင္းစီကို access လုပ္ပါမယ္ parameter ေတြလည္း ထည့္ေပးပါ့မယ္ ။

ဒုတိယ တစ္ဆင့္ အေနျဖင့္ app.js ထဲတြင္ ေအာက္ပါ code မ်ားကို ေရးသားပါမယ္ ။

```
var respon=require('./appp.js');
console.log(respon.data.sumNumber(9,10));
console.log(respon.data.circleCircumference(10));
console.log(respon.data.areaOfRectangle(9,10));
```

output အေနျဖင့္ ေအာက္ပါ တို႕ကို ျမင္ရပါမယ္ ။

```
19
62.83185307179586
90
```

File system နွင့် own module import လုပ်ျခင်းကို ပိုမို နားလည်ုိင်ရန် ေအာက်ပါ sample code မ်ားကို ေလ့လာရန် လိုအပ်ပါသည် ။

Vij.js ဟု code file တစ်ခု ေဆာက်ျပီး ေအာက်ပါ code မ်ားကို ထည့်ပါ ။

```
var respon=require('./appp.js')
const fs=require('fs')

fs.writeFileSync('nData.txt','This is Cricle Data : ')
fs.appendFileSync('nData.txt',respon.fun.aCircle(5));

fs.writeFileSync('nControl.txt','This is Cricle Data : ')
fs.appendFileSync('nControl.txt',respon.fun.control(10,20));
```

ထို့ေနာက် appp.js ဟု code file တစ်ခု ေဆာက်ျပီး ေအာက်ပါ code မ်ားကို ထည့်ပါ။

```
var methods={};
methods.aCircle=function(radius){
   return 3.1415*radius*radius;
}
methods.control=function(analog,digital){
   return analog+digital;
}
exports.fun=methods;
```

ထို့ေနာက် file မ်ားအား save ျပီး node vij.js ကို run ျကည့်ပါ။ ထို့ေနာက် result အေနျဖင့် file အသစ်မ်ား တည်ေဆာက်ျပီး ထို File ထဲတြင်စာသား်မ်ား data မ်ား ပါဝင်ေနပါက ေအာင်ျမင်ပါသည်။

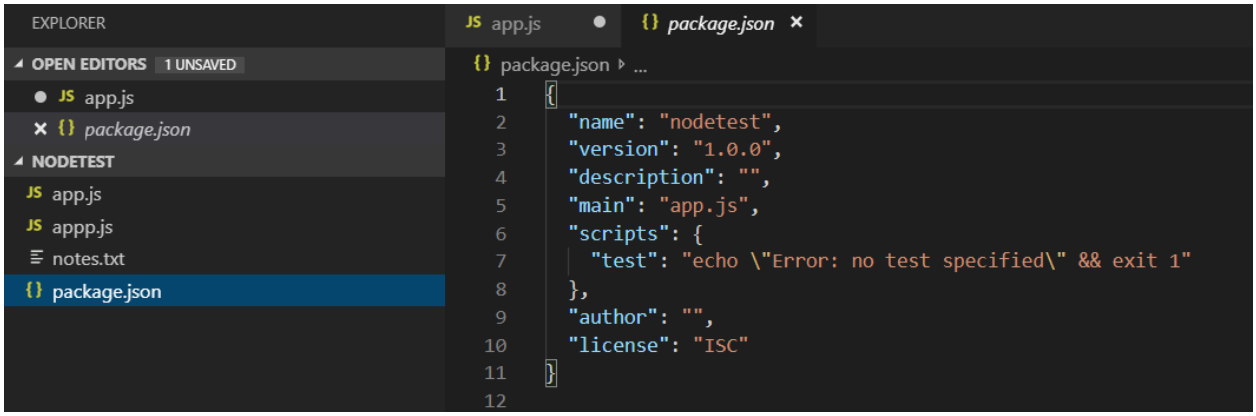
Importing npm Modules

Npm ဆိုတာက (Node Package Manager) ျဖစ်ျပီး JavaScript programming language အတြက် package manager ျဖစ်ပါတယ် ။ v0.6.3 ေနာက်ပိုင်း ကတည်းက node.js ကို install လုပ်လုိက်တနဲ့ npm က ပါလာျပီးသားျဖစ်ပါတယ် ။ console မှာ npm –version လို့ေရးလိုက်မယ်ဆိုရင် npm version ကို ျမင်ရမှာ ျဖစ်ပါတယ် ။ npm ကို စတင်ဖို့အ အတြက် console မှာ npm init ဆိုတဲ့ စာသားကိုက် ေရးေပးရပါမယ် ။ ထို့ေနာက် စတင်တာ ေအာင်ျမင်ဖို့အ အတြက် enter မ်ားသာ ဆက်တိုက် ေခါက်ေပးသြားရပါမယ် ။

```
About to write to C:\Users\MAT\Documents\nodeprogram\nodetest\package.json:

{
  "name": "nodetest",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

```
Is this OK? (yes)  // ယခုကဲ့သို့ေပၚလာလွ်င္ ေအာက္ပါ အတိုင္း yes လို့ ေရးေပးျပီး enter နွိပ္ပါ ။
Is this OK? (yes) yes
```

ေအာက္ပါ ပံုထဲက အတိုင္း package.json ဆိုတဲ့ javaScript object notation file တစ္ခု ေပးလာပါမယ္ ။



Validator

Validator ဆိုတဲ့ library ကေတာ့ string ေတြကို ခ်ဥ္းွ သီးသန့္ မွန္ မ မွန္ စစ္ေပးျခင္း နွင့္ code ထဲက မလိုအပ္တဲ့ အရာေတြကို ရွင္းလင္းေပးပါတယ္ ။ Validator ကို install လုပ္ဖို့ npm i validator@<version> version ေနရာမွာ မိမိ install လုပ္မယ့္ version ကို ထည့္ေပးရပါမည္ ။ version ကို သိဖို့ https://www.npmjs.com/package/validator ယခု link အတိုင္းသြားျပီး ျကည့္နုိင္ပါတယ္ ။ npm သည္ node package module ကို ဆိုလိုျခင္းျဖစ္ျပီး i သည္ install ကို ဆိုလိုျခင္းျဖစ္ပါသည္ ။ validator ကို install လုပ္ရန္ npm i validator@11.0.0 ဟု console တြင္ေရးျပီး enter ။ထို့ေနာက္ ေအာက္ပါ အတိုင္း ေပၚလာလွ်င္ validator ကို ေအာင္ျမင္စြာ install လုပ္ျပီး ျဖစ္ပါသည္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>npm i validator@11.0.0
npm WARN nodetest@1.0.0 No description
npm WARN nodetest@1.0.0 No repository field.

+ validator@11.0.0
updated 1 package and audited 1 package in 3.766s
found 0 vulnerabilities


C:\Users\MAT\Documents\nodeprogram\nodetest>
```

ထိုသို့ package ကို install လုပ္ျပီး လွ်င္ မိမိတို့ node folder ထဲတြင္ package-lock.json file တစ္ခုနွင့္ node_modules စတဲ့ directory အသစ္ တစ္ခုေပၚလာပါမည္ ။

Using Validator

Validator ကိုအသံုးျပဳရန္ require(‘ validator ’) ကို ေျကျငာေပးရန္ လိုအပ္ပါသည္ ။ validator ထဲကမွ isEmail ဆိုသည့္ function တစ္ခုကို စတင္ အသံုးျပဳပါမည္ ။ isEmail သည္ သူ႔ေနာက္ string argument ကို email format နွင့္ ကိုက္ညီ ျခင္း ရွိမရွိ ကို စစ္ေဆးပါသည္ ။ ကိုက္ညီပါက true ကို return ျပန္ေပးျပီး မကိုက္ညီပါက fasle ကိုသာ return ျပန္ေပးပါသည္။

```
const validator = require('validator')

console.log('checking')
console.log(validator.isEmail('winhtut@gh.com'))
```

အထက္ပါ အတိုင္း code ကိုေရးျပီး run ျကည့္မည္ဆိုလွ်င္

Checking

true ဆိုသည့္ output ကို ရပါမည္ ။

```
console.log(validator.isEmail('winhtut@gh'))
```

.com ကို ျဖုတ္ျပီး အထက္ပါcode ကို run ျကည့္ပါက false ကို ရပါလိမ့္မယ္။

Using isURL

isURL သည္ url ျဖစ္သားဆိုတာကို စစ္ပါတယ္ ။ ulr format နွင့္ကိုက္ညီလွ်င္ true လို႕return ျပန္ေပးျပီး မကိုက္ညီလွ်င္ false ကို return ျပန္ ေပးပါသည္ ။ example code ကို ေအာက္တြင္ ေဖာ္ျပထားပါသည္ ။

```
console.log(validator.isURL('https://www.facebook.com/WinHtutEquation'))
```

return ture

```
console.log(validator.isURL('https www.facebook.com/WinHtutEquation'))
```

return false

Printing in Color Using Chalk

Chalk ကို အသံုးျပဳရန္ ပထမဦးဆံုး install ျပဳလုပ္ေပးရန္ လိုအပ္ပါသည္ ။ validator နည္း အတိုင္း ပဲ ျပဳလုပ္ရုံပါပဲ ။ npm I chalk ယခု အတိုင္းလည္း ျပဳလုပ္ရုံျပီး chalk ေနာက္ဆံုး version ကို install ျပဳလုပ္ေပးမည္ျဖစ္ပါသည္ ။

```
npm WARN nodetest@1.0.0 No description
npm WARN nodetest@1.0.0 No repository field.

+ chalk@2.4.2
added 7 packages from 3 contributors and audited 8 packages in 2.435s
found 0 vulnerabilities
```

အထက္ပ အတိုင္း စာသာမ်ား ေပၚလွ်င္ chalk ကို ေအာင္ျမင္စြာ install လုပ္ျပီး ျဖစ္ပါသည္ ။ chalk ကို အသံုးျပဳရန္ validator နည္း အတိုင္း ေျကျငာေပးရန္ လိုအပ္ပါသည္ ။

```
const chalk=require('chalk')
console.log(chalk.blue('Hello World'))
```

Hello World ဆိုသည့် စာသားကို အျပာေရာင္ျဖင့္ ေဖာ္ျပရန္ chalk.blue ဆိုျပီး ေရးသားထားျခင္းျဖစ္ပါသည္ ။ chalk ေနာက္မွာ ေရးထားေသာ color သို႕မဟုတ္ instruction အတိုင္း လုပ္ေဆာင္ ေပးသြားပါမည္ ။ chalk package ကိုပိုမို နားလည္ေစရန္ ေအာက္ပါ code မ်ားကို ေလ့လာနိုင္ပါသည္ ။

```
const chalk=require('chalk')
const log = console.log;
log(chalk.blue('Hello') + ' World' + chalk.red('!'));

// Compose multiple styles using the chainable API
log(chalk.blue.bgRed.bold('Hello world!'));

// Pass in multiple arguments
log(chalk.blue('Hello', 'World!', 'Foo', 'bar', 'biz', 'baz'));

// Nest styles
log(chalk.red('Hello', chalk.underline.bgBlue('world') + '!'));

// Nest styles of the same type even (color, underline, background)
log(chalk.green(
    'I am a green line ' +
    chalk.blue.underline.bold('with a blue substring') +
    ' that becomes green again!'
));

// ES2015 template literal
log(`
CPU: ${chalk.red('90%')}
RAM: ${chalk.green('40%')}
DISK: ${chalk.yellow('70%')}
`);

// Use RGB colors in terminal emulators that support it.
log(chalk.keyword('orange')('Yay for orange colored text!'));
log(chalk.rgb(123, 45, 67).underline('Underlined reddish color'));
log(chalk.hex('#DEADED').bold('Bold gray!'));
```

Global npm Modules and nodemon

Validator and Chalk တို႕ဟာ locally package ျဖစ္ျကျပီး code file ထဲမွာ တိုက္ရိုက္ ေနရာယူ install လုပ္ျကပါတယ္ ။ nodemon ဆိုတဲ့ tool ကေတာ့ global package တစ္ခုျဖစ္ျပီး source file မွာ changes ျဖစ္လာတာကို ေစာင့္ျကည့္ ေပးတဲ့ tool တစ္ခုျဖစ္ပါတယ္။ node.js based application ေတြရဲ႕ source code file ထဲမွာ changes တစ္ခုခု ေတြ႕တာနဲ႕ server ကို automatic restart လုပ္ေပးပါတယ္။ အရမ္းကို အသံုးဝင္တဲ့ tool တစ္ခု ျဖစ္ျပီး သူ႕ကို အသံုးျပဳရန္ node app.js ေနရာတြင္ nodemon app.js လို႕ေရးျပီး nodemon ကို အသံုးျပဳနိုင္ပါတယ္ ။

nodemon ကို အသံုးျပဳဖို့နဲ့ အျခားေသာ package မ်ားကဲ့သုိ့ ပထမဦးဆံုး intall လုပ္ေပးရန္ လိုအပ္ပါတယ္။

npm i –g nodemon ဟုေဂၚျပီး install လုပ္နိုင္သလို npm install nodemon –g ဟို ေရးျပီးလည္း install လုပ္နုိင္ပါတယ္။ install လုပ္ျပီးလွ်င္ စစ္ေဆးဖို့အတြက္ nodemon –v ဟို ရိုက္ေပးပါ ထို့ေနာက္ terminal မွာ nodemon version ေပၚလာရင္ အသံုးျပဳလို့ရပါတယ္ ။ ေအာက္မွာ example code ကိုေဖာ္ျပထားပါတယ္ ။

```
const chalk=require('chalk')
const testMsg=chalk.blue.bold('WinHtut!');
console.log(testMsg)
```

node app.js (ယခုနည္း အတုိင္း run ပါ)။

အထက္ပါ code ကို run ရင္သာမန္ အတိုင္းပဲ အျပာေရာင္စာသားျဖင့္ WinHtut! ဆိုသည့္ စာသားကို ေတြ့ျမင္ရျပီး program အဆံုးသတ္သြားပါမယ္။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js
WinHtut!

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

အထပ္တြင္ ေဖာ္ျပထားေသာ နည္းလမ္းသည္ သာမန္ အတုိင္းပဲ run တာျဖစ္ျပီး nodemon ျဖင့္ run ျခင္း မဟုတ္ပါ။ nodemon ျဖင့္ run ရန္ nodemon app.js ယခုကဲ့ သို့ run ရပါမည္ ။ program အဆံုး သတ္မသြားပဲ ရပ္ေနသည္ကိုေအာက္ပါ အတိုင္း ျမင္ရပါမည္။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js
WinHtut!

C:\Users\MAT\Documents\nodeprogram\nodetest>nodemon app.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
WinHtut!
[nodemon] clean exit - waiting for changes before restart
```

Program run ေနတုန္းကို ျမင္ရျဖစ္ျပီး source code file ထဲတြင္ ေအာက္ပါ အတိုင္း ျပင္ျပည့္ကာ control+s ျဖင့္ code ကို save လုိ့ပါ ။

```
const chalk=require('chalk')
const testMsg=chalk.blue.inverse.bold('WinHtut!');
```

```
console.log(testMsg)
```

line number 2 တြင် inverse ဆိုသည့်စာသားကို ထပ္ထည့္လိုက္ပါသည် ထို႕ေနာက် control+s နွိပ္ျပီး လွ်င် output သည် ခ်က္ခ်င္း ခိန္းသြားသည္ကို ျမင္ရပါမည္။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>nodemon app.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
WinHtut!
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
WinHtut!
[nodemon] clean exit - waiting for changes before restart
```

အထက္ပါ ေဖာ္ျပထားေသာ နည္းအတိုင္းပဲ code မ်ားကို မိမိလုိအပ္သလို ျပဳျပင္းျပီး control+s နွိပ္လိုက္ပိုင္း output မ်ား အလို အေလ်ာက္ ခိန္းသြားသည္ကို ျမင္ရမွာပါ အဘယ္ေၾကာင့္ဆိုေသာ့ nodemon သည္ server ကို အလိုအေလ်ာက္ restart ခ် ေပးေသာေၾကာင့္ ျဖစ္ပါသည္။ control+c ကိုနွိပ္ျပီး program ကို အဆံုးသတ္နုိင္ပါသည္။

Standard Input

ပထမဆံုး အေနျဖင့္ app.js ဆိုသည့္ program file တစ္ခု တည္ေဆာက္ပါ ထိုေနာက္ေအာက္ပါ code မ်ားကို ေရးပါ။

```
console.log('Hello I am Win Htut')
```

အရင္ program မ်ားတြင္ app.js ဆိုသည့္ program ကို run ၾန္ node app.js ဆိုျပီး ေရးသည္။ ယခု program တြင္ node app.js WinHtut ဟု run ပါ ေနာက္ထည့္ေပးလိုက္ေသာ အပိုစာသားျဖစ္သည့္ WinHtut သည္ user input ျဖစ္သည္။

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js winhtut
Hello I am Win Htut

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

User input ျဖစ္သည့္ winhtut ဆိုသည့္ စာသားကို မျမင္ရေသးပါ။ ထိုစာသားကို ျမင္ရရင္ program တြင္ process.argv ဆိုသည့္ argument vector ကို ေရးေပးရန္ လိုအပ္ပါသည္ ။ argument vector သည္ user ထည့္ေပးလိုက္ေသာ data သို႕မဟုတ္ စာသားကို ဖမ္ေပးၾန္ျဖစ္သည္ ။

```
console.log('Hello I am Win Htut')

var data=process.argv
console.log(data)
```

process.argv သည် user မွ ထည့္ေပးလိုက္ေသာ စာသားမ်ားကို ရယူျပီး data ဆိုသည့္ variable ထဲသို႕ ထည့္ေပးပါသည္။ output အေနျဖင့္ ေအာက္ပါ အတိုင္း ျမင္ရပါမည္။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js winhtut
Hello I am Win Htut
[ 'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\MAT\\Documents\\nodeprogram\\nodetest\\app.js',
  'winhtut' ]

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

ပထမ အစိမ္းေရာင္ စာတန္းသည္ node ရွိေသာ path လမ္းေၾကာင္းကို ေဖာ္ျပျခင္းျဖစ္ျပီး ဒုတိယ အစိမ္းေရာင္ စာတမ္းသည္ မိမိတို႕ေရးသားထားေသာ app.js ဆိုသည့္ program file ရွိသည့္ ေနရာကို ေဖာျပျခင္း ျဖစ္သည္ ။ ေနာက္ဆံုးမွာ ေဖာ္ျပထားေသာ winhtut သည္ မိမိတို႕ ထည့္ေပးလိုက္ေသာ input ျဖစ္သည္။အာ့ကြင့္ sample program တစ္ပုဒ္ေရးျပထားပါသည္ ။

```
console.log('Hello I am Win Htut')

var data=process.argv
console.log(data)


if(data ==='add'){
   console.log('Adding Note!')
}else if(data === 'remove'){
   console.log('Removing note!')
}
```

User က ထည့္ေပးလိုက္ေသာ data မွာ add ျဖစ္လွ်င္ Adding Note! ဆိုသည့္ စာတမ္းကို ေဖာျပေပးမည္ ျဖစ္ျပီး remove ျဖစ္လွ်င္ Removing note! ဆိုသည့္ စာတမ္းကို ေဖာျပေပးရန္ ရည္ရြယ္ခ်က္ထားျခင္းျဖစ္သည္။

Yargs သည် node.js library file တစ်ခုျဖစ်ျပီး interactive command line tools မ်ား တည္ေဆာက်ို႕ အတြက် အသံုးဝင်ပါတယ်။ yargs ကို အသံုးျပဳရန် ပထမဆံုး install ျပဳလုပ်ေပးဂို့ လုိအပ်ပါသည် ။ install ျပုလုပ်ရန် terminal တြင် npm i yargs ကိုေရးေပးရပါမည် ။ ေအာက်ပါ code မ်ားကို app.js ဆိုသည့် program file တစ်ခု တည္ေဆာက်ျပီး run ျကည့်ပါ

```
const yargs=require('yargs')
console.log(process.argv)
console.log(yargs.argv)
```

output အေနျဖင့်ေအာက်ပါ အတိုင်း ျမင္ရပါမည်။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js
[ 'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\MAT\\Documents\\nodeprogram\\nodetest\\app.js' ]
{ _: [], '$0': 'app.js' }

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

အေပၚက အစိမ်းေကာင် စာတန်းနွစ်ခုသည် process.argv မွ output ျဖစ်ျပီး ေအာက်ဆံုး တစ်ေျကာင်းသည် yargs.argv မွ output ျဖစ်ပါသည် ။ yargs သည် object နွစ်ခု ကို Output ေပးပါသည် ပထမ တစ်ခုသည် _:[] , ျဖစ်ျပီး ဒုတိယ တစ်ခုသည် '$0': 'app.js' ျဖစ်ပါသည် ။ ပထမဆံုး object တစ်ခုကို အရင် ရွင်းျကပါစို့ ။ _:[ ] ယခု Object ထဲတြင် command နွင့် description ကိုေရးသားနိုင်ပါသည် ။ node app.js add --title="This is title" ယခု အတိုင်း run ျကည့်ပါက ေအာက်ပါ result ကို ျမင္ရပါမည် ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js
[ 'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\MAT\\Documents\\nodeprogram\\nodetest\\app.js' ]
{ _: [], '$0': 'app.js' }

C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js add --tile="This is title"
[ 'C:\\Program Files\\nodejs\\node.exe',
  'C:\\Users\\MAT\\Documents\\nodeprogram\\nodetest\\app.js',
  'add',
  '--tile=This is title' ]
{ _: [ 'add' ], tile: 'This is title', '$0': 'app.js' }

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

ယခုနည်းအတိုင်း yargs ကို အသုံးပြုနိုင်သည် ။ ပိုပြီး နားလည်စေရန် ဥပမာတို့ကို ဆက်လက် လေ့လာကြည့်ပါ ။ program တွင် process.argv ကို ဖျတ်ပြီး yargs.argv ကိုသာ အသုံးပြုပါမည် ။

```
const yargs=require('yargs')
console.log(yargs.argv)
```

အထက်ပါ code ကို run ကြည့်ပါက ဥပမာပါ အတိုင်း yargs output ကိုသာ ရှင်းလင်းစွာ ပြမရမည်

{ _: ['add'], tile: 'This is title', '$0': 'app.js' }

ဆက်လက်ပြီး yargs default option ဖြစ်သည့် --help and --version ကို လေ့လာကြည့်ပါမည် ။ ဥပမာပါ အတိုင်း run ရပါမည် ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --help
Options:
  --help     Show help                                                 [boolean]
  --version  Show version number                                       [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

node app.js –version ကို run ကြည့်လျှင် ဥပမာပါ အတိုင်း version number ကို ပြမရပါမည် ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --help
Options:
  --help     Show help                                                 [boolean]
  --version  Show version number                                       [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --version
1.0.0

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

ထို version number အား yargs version ကို အသံုးျပဳျပီး မိမိတို့စိတ္ျကိုက္ေျပာင္းလဲ နုိင္ပါ သည္။version number ေျပာင္းလဲ ဂုန္ program ထဲတြင္ေအာက္ပါအတိုင္းေရးေပးရမည္ ။

```
const yargs=require('yargs')
yargs.version('1.2.0')
console.log(yargs.argv)
```

program ကို run ျကည့္ခ်င္ version number မွာ မိမိတို့ ေပးလိုက္သည့္ number အတိုင္းျဖစ္ေနသည္ကို ျမင္ရပါမည္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --help
Options:
  --help     Show help                                                 [boolean]
  --version  Show version number                                       [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --version
1.0.0

C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --version
1.2.0

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

မိမိတို့ကိုယ္ပိုင္ add ဆိုသည့္ command တစ္ခုကို စတင္ဖန္တီးျကပါမည္။ program မွာ ေအာက္ပါ အတုိင္းျဖစ္သည္ ။

```
const yargs=require('yargs')

yargs.version('1.2.0')

yargs.command({
   command: 'add',
   describe: 'Adding a new note',
   handler: function(){
      console.log('This is a note')
   }

})
console.log(yargs.argv)
```

yargs.command သည္ yargs ထဲမွ command ဆိုသည့္ function ကို အသံုးျပဳျပီးပီး ထို function ထဲတြင္ မိမိတို့စိတ္ျကိုက္ command မ်ား descrption မ်ား function မ်ားကို တည္ေဆာက္နိုင္သည္။

```
command: 'add', program run ေသာ အခါ၌ add ဆိုသည့် command ကိုေခၚလိုက္သည့္
ေနာက္ တစ္ျပိုင္နက္ သူ႕ေအာက္ description တြင္ေဖာျပထားေသာ စာမ်ား function မ်ားကို
အလုပ္လုပ္ေပ းမည္ ျဖစ္ပါသည္ ။
describe: 'Adding a new note',
```

describe သည္ မိမိတို့ ေဖာ္ျပလိုေသာ အေျကာင္းအရာမ်ားတြက္ အသံုးျပဳပါသည္ ။ hanldler ေနာက္တြင္ function မ်ားေရးသားနိုင္သည္ ။

Terminal တြင္ node app.js –help ဟု run ျကည့္လ်ွင္ ေအာက္ပါ အတုိင္း အခ်က္အလက္မ်ား တိုးလာသည္ကို ျမင္ရပါမည္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --help
app.js [command]

Commands:
  app.js add  Adding a new note

Options:
  --help     Show help                                         [boolean]
  --version  Show version number                               [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

App.js ထဲတြင္ add ဆိုသည့္ command ႏွင့္ Adding a new note ဆိုသည့္ စာသားမ်ား ပိုလာသည္ကို ျမင္ရပါမည္ ။ node app.js add ဆိုျပီး ေနာက္ တစ္ျကိမ္ run ျကည့္လ်ွင္ add command ႏွင့္ သက္ဆိုသည့္ ေဖာ္ျပခ်က္မ်ားသာ က်လာသည္ကို ျမင္ရပါမည္ ။

```
app.js [command]

Commands:
  app.js add  Adding a new note

Options:
  --help     Show help                                         [boolean]
  --version  Show version number                               [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js add
This is a note
{ _: [ 'add' ], '$0': 'app.js' }

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

မူရင္း default ပါဝင္ေသာ --help and version ဆိုသည့္ command မ်ား အျပင္ ယခုဆိုလ်ွင္ add ဆိုသည့္ command တစ္ခုကိုပါ မိမိတို့ကိုယ္တိုင္ ဖန္တီးနိုင္ျပီျဖစ္သည္ ။ ေနာက္ command တစ္ခု အေနျဖင့္ remove ကို ေရးသားျကည့္ျကပါစို့ ။ ေအာက္တြင္ remove အတြက္ command တစ္ခု ေဂၚျပထားပါသည္ ေရးခဲ့ျပီးေသာ add command ႏွင့္ သေဘာ တရား ခ်င္း အတူတူပင္ ျဖစ္ပါသည္ ။

```
const yargs=require('yargs')
yargs.version('1.2.0')

yargs.command({
   command: 'add',
   describe: 'Adding a new note',
   handler: function(){
      console.log('This is a note')
   }
})
yargs.command({
   command: 'reomve',
   describe: 'Removing  a note',
   handler: function(){
      console.log('We are created remov command')
   }
})
console.log(yargs.argv)
```

node app.js –help ဟု run ျကည့္ခ်င္ ေအာက္ပါ အတုိင္း remove command တိုးလာသည္ကို ျမင္ရပါမည္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --help
app.js [command]

Commands:
  app.js add     Adding a new note
  app.js reomve  Removing  a note

Options:
  --help     Show help                                         [boolean]
  --version  Show version number                               [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

Node app.js add သို႕မဟုတ္ node app.js reomve(remove) တို႕ကို run ျကည့္ခ်င္ ထို command ေနာ့ သက္ဆိုင္သည့္ output တုိ႕ကိုသာ ျမင္ေတြ႕ရပါမည္ ။ ေအာက္တြင္ add , remove , read and list စသည့္ command မ်ားအား ဥပမာ အေနျဖင့္ ေရးသားေပးထားပါသည္။

```
const yargs=require('yargs')

yargs.version('1.2.0')

yargs.command({
```

```
    command: 'add',
    describe: 'Adding a new note',
    handler: function(){
        console.log('This is a note')
    }
})
yargs.command({
    command: 'remove',
    describe: 'Removing  a note',
    handler: function(){
        console.log('We are created remov command')
    }
})

yargs.command({
    command: 'read',
    describe: 'Reading  a note',
    handler: function(){
        console.log('We are created read command')
    }
})

yargs.command({
    command: 'list',
    describe: 'Listing  a note',
    handler: function(){
        console.log('We are created list command')
    }
})
console.log(yargs.argv)
```

အထက္ပါ code မ်ားကို ေလ့လာျပီးပါက ေနာက္တစ္ဆင့္ အေနျဖင့္ Option ထဲမွာ ေနာက္ထပ္ မိမိ ထည့္လိုေသာ title မ်ားကို ထည့္ပါမယ္ ။ title အသစ္ တစ္ခု ထည့္ခ်ိန္ command object ထဲတြင္ builder ဆိုသည့္ object တစ္ခုကို ထည့္ေပါမယ္ ထို builder ထဲမွာမွ မိမိ ထည့္လိုေသာ title ေခါင္းစဥ္ႏွင့္ instruction မ်ားကို ေရးသားရပါမယ္ ။

```
const yargs=require('yargs')
yargs.version('1.2.0')
console.log(yargs.argv)
```

အထက္ပါ code သံုးေျကာင္းအား run ျကည့္လွ်င္ output အေနျဖင့္ default option ႏွစ္ခုကိုသာ ျမင္ေတြ့ရပါမယ္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js --help
Options:
  --help     Show help                                                 [boolean]
  --version  Show version number                                       [boolean]

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

Node app.js –version လို႕ run ျကည့္ရင္ version ကို ေဖာျပ ေပးမည္ ။ မိမိတို႕ အေနျဖင့္ ေနာက္ ထပ္ အသစ္ တစ္ခု ထပ္ထည့္ပါမည္ ။အာကၠါ code မ်ားအား run ျကည့္ရင္ Options ထဲမွာ title တစ္ခု တိုးလာပါမည္ ။

```
const yargs=require('yargs')
yargs.version('1.2.0')

yargs.command({
   command: 'add',
   describe: 'Adding a new note',
   builder:{
      title:{
         describe: 'Title Note',
         demandOption:true,
         type:'string'
      }
   },
   handler: function(argv){
      console.log('This is a note')
   }
})
console.log(yargs.argv)
```

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js add
app.js add

Adding a new note

Options:
  --help     Show help                                                 [boolean]
  --version  Show version number                                       [boolean]
  --title    Title Note                                  [string] [required]

Missing required argument: title

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

Program ရွင်းလင်းခ်က်

```
yargs.command({
   command: 'add',
   describe: 'Adding a new note',
   builder:{ option အသစ် တစ်ခု ထပ်ဆောက်ဂုန် builder ကို သံုးပါသည်
      title:{ မိမိ ထည့်လိုသော နံမည်
         describe: 'Title Note', ဖော်ပြလိုသောအခ်က်အလက်
         demandOption:true, true ပေးထားမွ သာလွ်င် title ပေၚမည် ျဖစ်သည်
         type:'string'     အမ်ိဳးအစားကိုပါ ထည့်ေပးရမည် ။
      }
```

အထက်ပါ program ထဲသို့ title ထည့်ျကည့်လွ်င် ေအာ်ပါ အတိုင်းျမင်ရမည်

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js add --title="My Title"
This is a noteMy Title
{ _: [ 'add' ], title: 'My Title', '$0': 'app.js' }
```

Output တြင် array ပံုစံမ်ိဳး မေဖာ်ျပလိုေတ့ပဲ စာသား object အေနျဖင့်သ့ ေဖာ်ျပလိုေသာေျကာင့် program တြင် console.log(yargs.argv) ကို မသံုးေတ့ပဲ yargs.parse() ဟု parse() function ကိုသံုးပါမည်။ parse function ကိုသံုးရာတြင် return အေနျဖင့် argument vector object ကို ျပန်ေပးပါသည် ။

```
console.log('This is a note')
```

အထက်ပါ code ေနရာတြင်လည်း console.log('This is a note' + argv.title)ဟိုေျပာင်းေရး ရပါမည် သို့မွသာ user ထည့်ေပးလိုက်ေသာ စာသားကို အတိအက် string ပံုစံျဖင့် ေဖာ်ျပေပးမည် ျဖစ်သည်။ example source code ကိုေအာက် တြင် ေဖာ်ျပေပးထားပါသည် ။

```
const yargs=require('yargs')
yargs.version('1.2.0')

yargs.command({
   command: 'add',
   describe: 'Adding a new note',
   builder:{
      title:{
         describe: 'Title Note',
         demandOption:true,
         type:'string'
      }
   },
   handler: function(argv){
      console.log('This is a note' + argv.title)
```

```
    }
})
console.log(yargs.argv)
```

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js add --title="My Title"
This is a noteMy Title

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

အထက်ပါ program တြင် title option အျပင် body ထပ်ထည့်ပါမည်။ body ထပ်ထည့်ျခင်းမွာလည်း title ထပ်ထည့်ျခင်း အတိုင်းသာျဖစ်ျပီး လြယ်ကူပါသည်။ example source code ကို ေအာက်တွင် ေဖာ်ျပထားသည်။

```
const yargs=require('yargs')
yargs.version('1.2.0')

yargs.command({
    command: 'add',
    describe: 'Adding a new note',
    builder:{
        title:{
            describe: 'Title Note',
            demandOption:true,
            type:'string'
        }
    },
    body:{

        describe: 'Note Body',
        demandOption: true,
        type: 'string'

    },
    handler: function(argv){
        console.log('This is a note ' + argv.title)
        console.log(' Body description ' + argv.body)
    }
})
yargs.parse();
```

သတ်ျပီးရန္ run သည့် အခ်ိန္တြင် input အေနျဖင့် title အတြက္ေရာ body အတြက္ေရာပါ ထည့္ေပးရပါမည်။

```
C:\Users\MAT\Documents\nodeprogram\nodetest>node app.js add --title="My Title" --body="This is some body"
This is a note My Title
 Body description This is some body

C:\Users\MAT\Documents\nodeprogram\nodetest>
```

Storing Data With JSON

First Step အေနျဖင့္ object တစ္ခု ဖန္တီးပါမယ္ ။ ထို object ထဲတြင္ properties မ်ားထည့္ပါမယ္ ။ ေအာက္ပါ program တြင္ book ဆိုသည့္ object တစ္ခု ဖန္တီးလိုက္ျပီး ထို object ထဲတြင္ title and author ဆိုသည့္ properties ႏွစ္ခု ဖန္တီးလိုက္ပါတယ္ ။

```
const book={
   title: 'Nodejs',
   author: 'Win Htut Green Hackers Team'
}
```

Second step အေနျဖင့္ book ဆိုသည့္ object ထဲက properties or data ေတြကို JSON format သို႔ေျပာင္းပါမယ္ ။ ထိုသို႔ေျပာင္းရန္ JSON.stringify() ဆိုသည့္ function ကိုေခၚသံုးရပါတယ္။ ထို function သည္ သူ႕ထဲတြင္ ျဖတ္သာေသာ javascript value or object or array ကို JSON string အျဖစ္ return ျပန္ေပးပါတယ္ ။

```
const bookJSON=JSON.stringify(book)
```

return ျပန္ေပးေသာ JSON string ကို bookJSON ဆိုသည့္ variable ထဲမွာ stored လုပ္ပါတယ္။ သတိျပဳရန္မွာ bookJSON သည္ string ျဖစ္ေနပါျပီ။

Third Step အေနျဖင့္ bookJSON ကို output ထုတ္ပါမယ္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>node 1-json.js
{"title":"Nodejs","author":"Win Htut Green Hackers Team"}

C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>
```

သတိျပဳရန္အခ်က္မွာ output ထြက္လာေသာ စာသားမ်ားသည္ single quote မဟုတ္ဘဲ (“ ”) double quotes ထဲတြင္ ေဖာျပေပးခဲ့ျခင္းျဖစ္သည္ ။ အဘယ္ေျကာင့္ဆိုေသာ့ ထိုအခ်က္အလက္မ်ားသည္ JSON string အျဖစ္ ေျပာင္းလဲ ခံထားရေသာေျကာင့္ ျဖစ္ မူရင္း object မဟုတ္ေတာ့ပါ ။ object ဟုတ္မဟုတ္ စစ္ေဆးရန္ ေအာက္ပါ အတိုင္း code line တစ္ေျကာင္း ပိုထည့္ျပီး run ျကည့္ပါ ။

```
console.log(bookJSON.title)
```

output အေနျဖင့္ undefined ဆိုသည့္ စာသား ပါလာသည္ကို ေတြ့ျမင္ရပါလိမ့္မည္ ။ အဘယ္ေျကာင့္ ဆိုေသာ့ bookJSON သည္ object မဟုတ္ေတာ့ပါ ။

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>node 1-json.js
{"title":"Nodejs","author":"Win Htut Green Hackers Team"}
undefined
```

Fourth Step အေနျဖင့္ ထြက္လာေသာ JSON string မ်ားကို object သို့ ျပန္ေျပာင္းပါမည္။ parse function ကိုသံုးပါမည္ parse() function သည္ JSON string ကို object အျဖစ္ return ျပန္ေပးပါသည္။

```
const bookObject=JSON.parse(bookJSON)
```

ယခု အခ်ိန္တြင္ bookObject သည္ ယခု အခ်ိန္တြင္ object ျပန္ျဖစ္သြားပါျပီ ။ object ဟုတ္မဟုတ္ ေသခ်ာ ေစရန္ bookObject.title ဆိုျပီး bookObject ထဲမွ title ကို ထုတ္ျကည့္ပါမည္ ။

```
console.log(bookObject.title)
```

ေအာက္ပါ အတိုင္း program အျပည့္အစံုအား run ျကည့္ျပီး output ကို စစ္ေဆးနိုင္ပါသည္။

```
const book={
    title: 'Nodejs',
    author: 'Win Htut Green Hackers Team'
}
const bookJSON=JSON.stringify(book)
console.log(bookJSON)
const bookObject=JSON.parse(bookJSON)
console.log(bookObject.title)
```

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>node 1-json.js
{"title":"Nodejs","author":"Win Htut Green Hackers Team"}  JSON string
undefined    return undefined because it is not a object

C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>node 1-json.js
{"title":"Nodejs","author":"Win Htut Green Hackers Team"}  JSON string
Nodejs  return titile name because it is a object now

C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>
```

Fifth Step အေနျဖင့္ json file တစ္ခု တည္ေဆာက္ျပီး ထို json file ထဲတြင္ second step မွ ရ ရွိလာေသာ JSON string မ်ားကို သြားေရာက္သိမ္းဆည္းပါမည္ ။ json file တစ္ခု တည္ေဆာက္ရန္ file system package (fs) ကို အသံုးျပဳပါမည္ ။ source code မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္

```
const fs=require('fs')
const book={
    title: 'Nodejs',
    author: 'Win Htut Green Hackers Team'
}
const bookJSON=JSON.stringify(book)
fs.writeFileSync('1-json.json',bookJSON)
```

အထက်ပါ program ကို run ပြီးပါက မိမိတို့ source code folder ထဲတြင် 1-json.json ဆိုသည့် file တစ်ခု အသစ် တိုးလိုသည်ကို ပြမှုပါမည်။ ထို file ကို ဖြင့်ကြည့်ပါက အခ်က် အလက်မ်ားကို JSON string အနေဖြင့်ပြမှုပါမည် ။

```
{"title":"Nodejs","author":"Win Htut Green Hackers Team"}
```

Sixth Step အနေဖြင့် Create လုပ်ထားသော ထို JSON file ထဲမှ အခ်က်အလက် မ်ားကို ဖတ်ပါမည်။ ဖတ်လို့ရလာသော data မ်ားကို object အနေဖြင့်ပြန်လည် ပြောပြပါမည်။ ထို့သို့ ဖတ်ရန် fs.readFileSync ဆိုသည့် function ကိုသုံးပါမည် ထို function ကို သုံးလိုင် return အနေဖြင့် buffer data မ်ား ပြန်ပေးပါသည်။

```
const fs=require('fs')

const bufferData = fs.readFileSync('1-json.json')
console.log(bufferData)
```

output အနေဖြင့် အောက်ပါ အတိုင်း ပြမှုပါမည် ။

<Buffer 7b 22 74 69 74 6c 65 22 3a 22 4e 6f 64 65 6a 73 22 2c 22 61 75 74 68 6f 72 22 3a 22 57 69 6e 20 48 74 75 74 20 47 72 65 65 6e 20 48 61 63 6b 65 72 73 ... >

ထို ထြက်လာသော buffer data မ်ားကို string သို့ ပြန်ပြောင်းပါမည် ။ ထိုသို့ပြောင်းရန် .toString() ဆိုသည့် function ကိုအသုံး ပြုရပါမည်။

```
const dataJSON=bufferData.toString()
```

ရရှိလာသော JSON string မ်ားကို object အဖြစ်သို့ ပြန်ပြောင်းပါမည် ။

```
const dataOBJ = JSON.parse(dataJSON)
```

ယခု အဆင့်ပြီးလိုင် dataOBJ ဆိုသည့် variable သည် object တစ်ခု ဖြစ်သွားပါပြီ object တစ်ခု ဖြစ်သည့်အလျောက် ထို object ထဲမှ properties မ်ားကို ပြောင်းပြန်လှန်လို update လည်း လုပ်နိုင်ပါသည်။ dataOBJ ကို Output အနေဖြင့် ထုတ်ပြကည့်လိုင် 1-json.json ထဲမှ data မ်ားကို object အနေဖြင့် ပြန်ပြောပြပါမည် ။

```
const fs=require('fs')

const bufferData = fs.readFileSync('1-json.json')
const dataJSON=bufferData.toString()
const dataOBJ = JSON.parse(dataJSON)
console.log(dataOBJ)
```

output

{ title: 'Nodejs', author: 'Win Htut Green Hackers Team' }

Object file ထဲမှ properties တစ်ခုကို ထုတ်ပြကည့်ပါမည် ။

```
console.log(dataOBJ.author)
```

output အေနျဖင့္ ေအာက္ပါ အတိုင္း ထြက္လာပါသည္ ထို႔ေျကာင့္ dataOBJ သည္ object ျဖစ္ပါသည္ ။

```
{ title: 'Nodejs', author: 'Win Htut Green Hackers Team' }
```

```
Win Htut Green Hackers Team
```

Seven step အေနျဖင့္ JSON file ထဲက properties မ်ားကို update လုပ္ပါမည္ ။ ထိုသို႕ update လုပ္ရန္ JSON file ထဲက အခ်က္ အလက္မ်ားကို ေအာက္ပါအ တိုင္း ျပင္ဆင္ပါမည္။ 1-json.json ထဲတြင္

```
{"title":"Nodejs",
"author":"Win Htut Green Hackers Team",
"page":"1000",
"language":"English"}
```

ယခု အတိုင္း ျပင္ဆင္ပါ ။ ထို႕ေနာက္ 1-json.js program file ထဲတြင္ ေအာက္ပါ အတိုင္းေရးျပီး run ျကည့္ပါ။

```
const fs=require('fs')

const bufferData = fs.readFileSync('1-json.json')
const dataJSON=bufferData.toString()
const dataOBJ = JSON.parse(dataJSON)
console.log(dataOBJ)
```

output အေနျဖင့္ ေအာက္ပါ object မ်ားကို ျမင္ရပါမည္ ။

```
{ title: 'Nodejs',
  author: 'Win Htut Green Hackers Team',
  page: '1000',
  language: 'English' }
```

ထို object ထဲတြင္ ရွိေသာ အခ်က္ အလက္မ်ားကို update ျပင္ဆင္ပါမည္ ။ Nodejs ေနရာတြင္ C programming ဟုလည္းေကာင္း author ေနရာတြင္ Win Htut ဟုလည္းေကာင္း page ေနရာတြင္ 300 ဟုလည္းေကာင္း language ေနရာတြင္ Myanmar ဟုလည္းေကာင္း ျပင္ဆင္ပါမည္ ။ title ျပင္ဆင္ရန္ program ထဲတြင္

```
dataOBJ.title='C Programming'
```

ယခု အတိုင္းျပင္ဆင္ပါမည္။ အျခားေသာ author ,page and language မ်ားသည္ လည္း ထိုနည္း အတိုင္း ပင္ျဖစ္သည္ ။ program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const fs=require('fs')
const bufferData = fs.readFileSync('1-json.json')
const dataJSON=bufferData.toString()
const dataOBJ = JSON.parse(dataJSON)
dataOBJ.title='C Programming'
dataOBJ.author='Win Htut'
```

```
dataOBJ.page = 300
dataOBJ.language='Myanmar'
console.log(dataOBJ)
```

အထက်ပါ program ကို run ကြည့်ခြင့် output အနေဖြင့် JSON file ထဲရှိ အခ်က်အလက်များသည် မိမိတို့ ပြပြင်လိုက်သည့် အတိုင်း ေပြာင်းလဲ သြားသည်ကို ြမင်ရပါမည် ။

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>node 1-json.js
{ title: 'Nodejs',
  author: 'Win Htut Green Hackers Team',
  page: '1000',
  language: 'English' }

C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>node 1-json.js
{ title: 'C Programming',
  author: 'Win Htut',
  page: 300,
  language: 'Myanmar' }

C:\Users\MAT\Documents\nodeprogram\nodeprogram\playground>
```

Introduction to Web Server

Express

ယခု သင်ခန်းစာတြင် Express web Framwork အားအသုံးြပုပြီး web server တစ်ခု ေရးသားသြားမှာ ြဖစ်ပါတယ်။

Fist Step web-server ဆိုသည့် folder တစ်ခု တည်ေဆာက်ပါမည် ။ ထို folder ထဲတြင် npm init ကိုသုံးြပီး npm ကို စတင် ပါမယ် ။ ထို့ေနာက် npm i express ကိုသုံးြပီး express package ကို install လုပ်ပါမည်။ ထို့ေနာက် web-server folder ထဲတြင်ငင် src ဆိုသည့် folder တစ်ခု တည်ေဆာက်ပါမည် ထို folder ထဲတြင် app.js ဆိုသည့် js source code file တစ်ခု တည်ေဆာက်ပါမည် ။ app.js ထဲတြင် express library အား စတင်အသုံးြပရန် ေြကာငေပးပါမည် ။

```
const express=reqire('express')
```

express function အား variable တစ်ခု တည်ေဆာက်ြပီး ထည့်ထားပါမည် ။

```
const app=express()
```

express() ကို ကိုယ်စားြပုထားေသာ app ထဲမှ GET request အား စတင်အသုံးြပုရန် app.get ကို ေရးသားပါမည် get function တြင် arguments နှစ်ခု ပါဝင်ပါသည် get(‘ ’ , (req,res)) req (request) ြဖစ်ြပီး res (respon) ြဖစ်သည် ။ client မှ request လုပ်လာလ်င် res ကို အသုံး ြပုပြီး respon ြပန်ေပးပါမည် ။ ‘ ’ သည် directory တည်ေနရာ ြဖစ်သည် ။

```
app.get('',(req,res)=>{

   res.send('Hello Express World!')
```

```
})
```

(req ,res)=> သည် arrow function ကို အသံုးျပဳထားျခင္းျဖစ္သည္ ။ respon ျပန္ေပးရာတြင္ res.send() ကို သံုးျပီး send function ထဲတြင္ မိမိ ေဖာ္ျပလိုေသာ အခ်က္ အလက္မ်ားကို ေဂ်းသားပါသည္ ။

Second Step server အား စတင္ run ရန္ listen ဆိုသည့္ function အားအသံုးျပဳျပီပါမည္ ။ listen ထဲတြင္ argument ႏွစ္ခု ထည့္ေပးရမည္ ပထမ တစ္ခုသည္ listen လုပ္မည့္ port number ျဖစ္ျပီး ဒုတိယ တစ္ခုသည္ call back function ျဖစ္သည္ ။

```
app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

Port number 3000 ကို အသံုးျပဳမည္ျဖစ္ျပီး call back function ထဲတြင္ ေဂ်းသားထားေသာ console.log ေနာက္ အခ်က္အလက္မ်ားသည္ client ဘက္တြင္ လံုးဝ ေပၚမည္ မဟုတ္ပဲ server ဘက္တြင္သာ ျမင္ရပါမည္ ။ program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္ပါသည္ ။

```
const express=require('express')
const app=express()

app.get('',(req,res)=>{
   res.send('Hello Express World!')
})
app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

Program အား run ျကည့္ပါက ေအာက္ပါ အတိုင္း port 3000 မွာ listen လုပ္ခုိင္ေျကာင္း ျပေနပါမည္ ။ ထိုေနာက္ ေပၚလာေသာ windowform တြင္လည္း allow ေပးရန္ လိုအပ္ပါသည္။

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\src>node app.js
Server is up on port 3000.
```

မိမိ ႏွစ္သက္ရာ browser တစ္ခုကို ဖြင့္ျပီး http://localhost:3000/ ယခု အတိုင္းေရးျကည့္ပါ က Hello Express World! ဆိုသည့္ စာသားကို ျမင္ေတြ့ရပါမည္ ။ server အား shutdown ပိတ္ရန္ control+C ကိုနွိပ္ေပးရပါမည္ ။

Third Step အေနျဖင့္ help and about page မ်ားကိုထပ္ထည့္ပါမည္ ။ ပထမ ေရးခဲ့သည့္ အတိုင္းပင္ ျဖစ္သည္ မတူသည့္ ေနရာမွာ directory ေနရာတြင္ help ကို ထပ္ထည့္ေပးရန္လိုအပ္ပါသည္ ။ ေရွ့တြင္ / ထည့္ေပးရန္ လိုအပ္ပါသည္ ။

```
app.get('/help',(req,res)=>{
   res.send('This is help page')
```

```
})
```

အထက္ပါ ေရးထားခ်က္ မ်ားသည္ help page အတြက္ျဖစ္ပါသည္ ။ about page အတြက္လည္း ထိုနည္းအတိုင္းပင္ ျဖစ္သည္ ။ program အျပည့္အစံုကို ေအာက္တြင္ေဖာ္ျပထားပါသည္ ။

```
const express=require('express')
const app=express()

app.get('',(req,res)=>{
   res.send('Hello Express World!')
})

app.get('/help',(req,res)=>{
   res.send('This is help page')
})
app.get('/about',(req,res)=>{
   res.send('This is about page')
})

app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

အထက္ပါ program အား run ျပီး browser တြင္ http://localhost:3000/about ယခုအတိုင္း ေရးျကည့္ပါက This is about page ဆိုသည့္ စာသား ကို ျမင္ေတြ့ရပါမည္ ။ http://localhost:3000/help ယခု အတိုင္း ေရးျကည့္ပါက This is help page ဆိုသည့္ စာသားကို ျမင္ရပါမည္ ။

Fourth Step အေနျဖင့္ Client ဘက္သို့ HTML (Hyper text markup language) and JSON (javascript object notation) မ်ားကို ျပန္ပို့ ေပးပါမည္ ။ ပထမဆံုးူး home page သို့ HTML မ်ား ပို့ေပးပါမည္ ။ res.send ထဲတြင္ မိမိ ပို့လိုေသာ HTML မ်ား ပို့ေပးနိုင္သည္ ။

```
app.get('',(req,res)=>{
   res.send('<h1>Hello World I am Win Htut</h1>')
})
```

About page သို့ JSON file မ်ား ပို့ေပးပါမည္ ။ res.send ထဲတြင္ငင္ မိမိ ပို့လိုေသာ JSON မ်ား ပို့ေပးနိုင္ပါသည္ ။ program အျပည့္အစံုမွာ ေအာက္ပါအတိုင္းျဖစ္ျပီး run ျကည့္နိုင္သည္ ။

```
const express=require('express')
const app=express()

app.get('',(req,res)=>{
```

```
    res.send('<h1>Hello World I am Win Htut</h1>')
})
app.get('/help',(req,res)=>{
    res.send('This is help page')
})
app.get('/about',(req,res)=>{
    res.send({
        Name: 'WinHtut',
        Age:24,
        Tall:5.9
    })
})

app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

http://localhost:3000/ **Home page ကို run ျကည့္ပါက Hello World I am Win Htut ဆိုသည့္ စာသားမ်ားကို စားလံုး အျကီးျဖင့္ ျမင္ရမည္ျဖစ္ျပီး** http://localhost:3000/about about page **ကို run ျကည့္ပါက JSON စာသားမ်ားကို ျမင္ရေပမည္ ။**

{"Name":"WinHtut","Age":24,"Tall":5.9}

Serve up Static

Fitst Step အေနျဖင့္ မိမိတို့ တည္ေဆာက္ထားေသာ web-server folder ထဲတြင္ public ဆိုသည့္ folder တစ္ခုကို တည္ေဆာက္ပါမည္ ။ ထို public folder ထဲတြင္ index.html ဆိုသည့္ html file တစ္ခု တည္ေဆာက္ပါမည္ ။ ထို html file ထဲတြင္ ေအာက္ပါ code မ်ားအား ေရးသားပါမည္ ။

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <h1>This is From Static File</h1>
    </body>
</html>
```

ထို့ေနာက္ src folder ထဲက app.js file ရွိသည့္ေနရာတြင္ nodemon ကို install လုပ္ပါမည္ nodemon ကို install လုပ္ရန္ npm i nodemon ဟုေရးရပါမည္ ။ ထို့ေနာက္ nodemon app.js ဆိုျပီး run ပါ ။ ထို့ေနာက္ console မ်ာ directory and file name ကို ျကည့္ပါမည္။ ထိုသို့ ျကည့္ရန္ console.log(__dirname) ႏွင့္ console.log(__filename) တို့ကို ေရးရန္လိုအပ္သည္ ။ control s ႏွိပ္လိုက္သည့္အခ်ိန္ console မွာ directory and file name ကို ျပေပးပါလိမ့္မည္ ။ code အျပည့္အစံုကို ေအာက္တြင္ ေဖာ္ျပထားပါ သည္ ။

```
const express=require('express')
const app=express()
```

```
console.log(__dirname)
console.log(__filename)
app.get('',(req,res)=>{
   res.send('<h1>Hello World I am Win Htut</h1>')
})
app.get('/help',(req,res)=>{
   res.send('This is help page')
})
app.get('/about',(req,res)=>{
   res.send({
      Name: 'WinHtut',
      Age:24,
      Tall:5.9
   })
})
app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\src>nodemon app.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\src
C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\src\app.js
Server is up on port 3000.
```

ပထမ စာတန္းသည္ directory လမ္းေၾကာင္းျဖစ္ျပီး ဒုတိယ စာတန္းသည္ file name ကို ေဖာျပေပးျခင္း ျဖစ္သည္ ။

Second Step အေနျဖင့္ static ျဖစ္ခ်င္ေသာေၾကာင့္ express.static ကို ေခၚသံုးရပါမည္ ထို express.static ထဲတြင္ argument အေနျဖင့္ index.html ရွိသည့္ ေနရာကို ေဖာျပေပးရပါမည္။ ထို႕သို႕ ေဖာျပေပးရန္ path.join ကို သံုးရပါမည္ ထို join function ထဲတြင္ argument ႏွစ္ခု ပါဝင္ပါသည္ ပထမ တစ္ခုသည္ __dirname ဆိုသည့္ directory name ျဖစ္ျပီး ဒုတိယ တစ္ခုသည္ index.html ရွိသည့္ folder ေနရာျဖစ္သည္ ။ path ကို ေခၚသံုးရန္ require(‘ path ’) ဆိုျပီး ေၾကျငာေပးၾကန္ လိုအပ္သည္ ။ ထို႕ေနာက္ path.join (‘__dirname, ‘ html file ရွိသည့္ေနရာ’) ကိုေရးေပးရမည္ ။

```
const express=require('express')
const path=require('path')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
```

path.join ကို publicDirectoryPath ဆိုသည့် variable ထဲတြင် ထည့်ထားပါသည် ။ ထို variable ကိုမှ express.static ဆိုသည့် static function ထဲတြင် argument အနေဖြင့် ပေးရသည် အသံုးပြုပံပါမည် ။

```
app.use(express.static(publicDirecotryPath))
```

program အပည့်စံုမှာ ေအာက်ပါ အတိုင်းပြဖွဲ့သည် ။

```
const express=require('express')
const path=require('path')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
app.use(express.static(publicDirecotryPath))


app.get('',(req,res)=>{
    res.send('<h1>Hello World I am Win Htut</h1>')
})
app.get('/help',(req,res)=>{
    res.send('This is help page')
})
app.get('/about',(req,res)=>{
    res.send({
        Name: 'WinHtut',
        Age:24,
        Tall:5.9
    })
})

app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

Nodemon ဖြင့် run ထားသောေကာင့် control+s နှိပ်လိုက်သည့်နှင့် တစ်ပုံုနက် server တြင် changes ဖြစ်သွားပါမည် ။ ထို့နောက် localhost:3000 ဖြင့် browser တြင် ကည့်ပါက This is From Static File ဆိုသည့် စာသားမ်ားကို ပြင်ေတြ့ရေပမည် ။ ထို့ေကာင့် နဂိုမူရင်း ေဂးထားေသာ အခ်ိ့ code မ်ားဖြစ်သည့် root page ,help page and about page တို့မှာ မလိုအပ်ေတာ့ ေသာေကာင့် ဖက်ပစ်နိုင်ပါသည် ။ လိုအပ်ေသာ code မ်ားမှာ ေအာက်ပါအတိုင်းသာ ဖြစ်သည် ။

```
const express=require('express')

const path=require('path')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
app.use(express.static(publicDirecotryPath))
```

```
app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

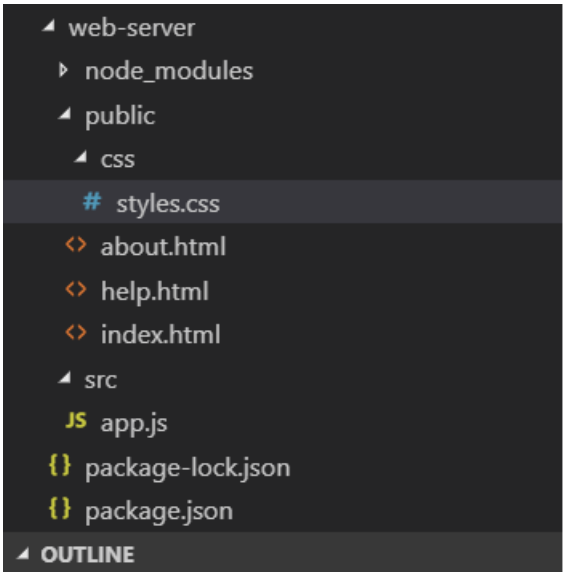
ျပန္ run ျကည့္ပါက public ေအာက္တြင္ ရွိေသာ index.html file မွ စာသားမ်ားကိုသာျမင္ရမည္ ျဖစ္သည္။

Third Step အေနျဖင့္ about and help page မ်ား ထပ္ထည့္ရန္ public folder ထဲတြင္ about.html file နွင့္ help.html file မ်ားကို တည္ေဆာက္ပါမည္။ about.html page တြင္ about ဆိုသည့္ စာတစ္ေျကာင္းေပၚေအာင္ ေရးထားျပီး help.html တြင္လည္း help ဆိုသည့္ စာတန္းေပၚေအာင္ ေရးသားထားပါမည္ ။ ထို႕ေနာက္ File အားလံုးအား ေသခ်ာ save ျပီး control+s နွိပ္ျပီး http://localhost:3000/about.html ယခုအတိုင္းသြားျကည့္ပါက about ဆိုသည့္ စာသားကို ျမင္ရပါမည္။ ထိုနည္းအတိုင္းပင္ help.html ကိုလည္းျကည့္နိုင္ပါသည္ ။

Serving up CSS, JS, Images and More

Web page မွာ စာသားမ်ား ပံုမ်ား script မ်ားထည့္ဖို႕ အတြက္ CSS , JS , Images တို႕ကို သံုးသြားပါမည္။

First Step အေနျဖင့္ Public Folder ေအာက္တြင္ css file မ်ား သိမ္းရန္ CSS ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါမည္ ။ ထို css folder ထဲတြင္ styles.css ဆိုသည့္ css file တစ္ခု တည္ေဆာက္ပါမည္ ။ Directory မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။



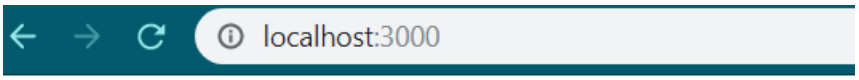
ထို styles.css file ထဲတြင္ index.html file မွ h1 ျဖင့္ ေရးသားထားေသာ စာသားမ်ားကို အေရာင္ ေျပာင္းလဲ ရန္ ေအာက္ပါအတိုင္း code ေရးသားပါမည္ ။

```
h1{
   color:grey;
}
```

ထို႕ေနာက္ index.html file ထဲတြင္ styles.css file အား link ခ်တ္ေပးရန္ ေအာက္ပါ code မ်ားကို ေရးသားရပါမည္ ။ program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

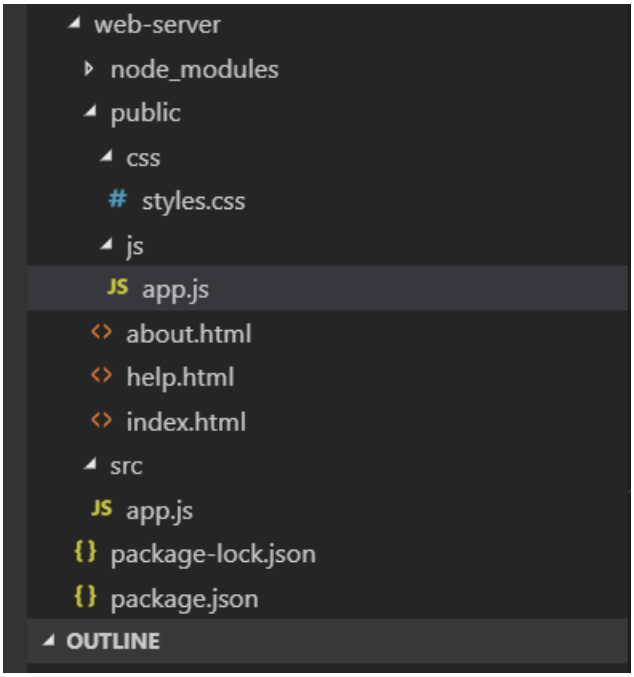
```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="./css/styles.css">
  </head>
  <body>
    <h1>This is From Static File</h1>
  </body>
</html>
```

Link ခံတူမည့္ code ကို <head> tag ထဲတြင္ ေရးသားရမည္ျဖစ္ျပီး rel ႏွင့္ href ႏွစ္ခု ပါဝင္ပါမည္။ တစ္ခုႏွင့္ တစ္ခုျကားတြင္ space ျခားဂမည္ျဖစ္ျပီး rel သည္ relation of the linked document ကို ကိုယ္စားျပဳပါသည္ ။ href တြင္ css file directory ကို အတိအက် ထည့္ေပးဂပါ မည္။ ထို႕ေနာက္ program မ်ားအား save ျပီး nodemon app.js ကို ျပန္ run ျကည့္ပါ။ ေအာက္ပါ အတိုင္း စာသား color ေျပာင္းသြားသည္ကို ျမင္ရပါမည္ ။



This is From Static File

Second Step အေနျဖင့္ javascript code မ်ားကို ထည့္သြားပါမည္။ ထိုသို႕ ထည့္ဂုန္ js ဆိုသည့္ folder တစ္ခုကို public folder ေအာက္တြင္ ထပ္ေဆာက္ပါမည္ ။ ထို js ဆိုသည့္ folder ထဲတြင္ app.js ဆိုသည့္ javascript file တစ္ခု တည္ေဆာက္ထားပါမည္ ။ Driectory မွာ ေအာက္ပါ အတိုင္း ျဖစ္သည္ ။



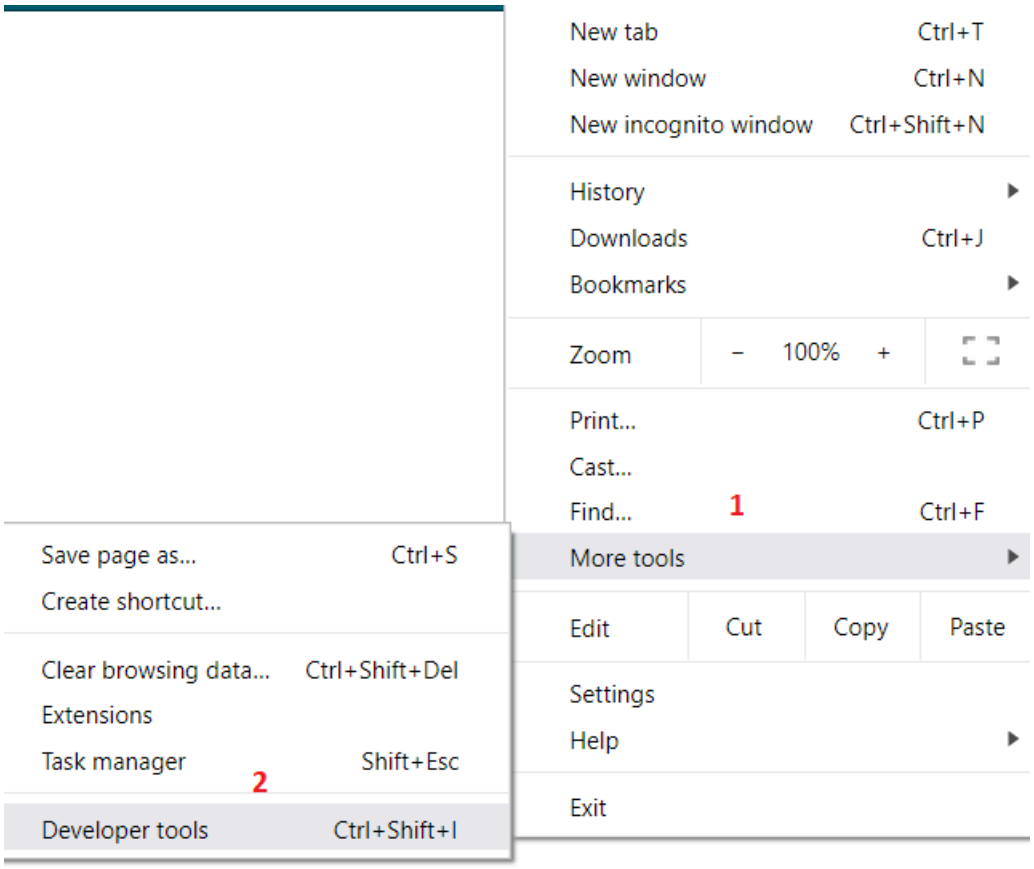
js folder ထဲမွ app.js ဆိုသည့္ file ထဲတြင္ javascript code မ်ားကို ေရးသားပါမည္ ။ ေရးသားလိုေသာ code မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။ ထို javascript code မ်ားသည္ Client Side အတြက္ ျဖစ္ပါသည္ ။

```
console.log('Client side javascript file is loaded!')
```

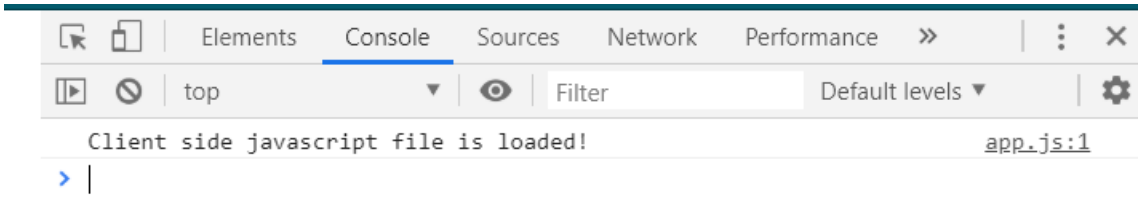
ထိုသို႕ ေရးသားျပီးေသာ code မ်ားကို effective ျဖစ္ေစဂုန္ index.html file ထဲတြင္ script တစ္ခု ပါဝင္မည္ ျဖစ္ေျကာင္းကို ေအာက္ပါအတိုင္းေရးသားေပးရပါမည္။ ထို႕ သို႕ ေရးသားရာတြင္ <head> tag ထဲတြင္သာ ေရးသားရပါမည္ ။ script ေရးသားမည္ ျဖစ္ရဲ့အတြက္ script ဆိုသည့္ tag ျဖင့္ စရပါမည္ ။ ထို script tag ထဲတြင္ src ပါဝင္ပါမည္ src သည္ external မွ run လိုေသာ scripit file location ကို ထည့္ ေပးရပါမည္ ။

```
<head>
    <link rel="stylesheet" href="./css/styles.css">
    <script src ="/js/app.js"></script>
  </head>
```

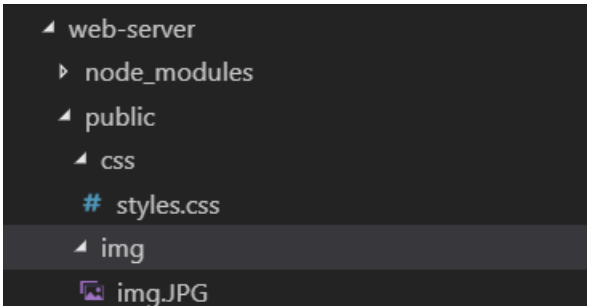
ထို့ေနာက္ မိမိ အသံုးျပဳေသာ browser ထဲမွ setting မွ တစ္ဆင့္ developer tools ထဲကို ဝင္ပါ။



ထို့သို့ သြားျပီးသည္ခ်င့္ တစ္ျပိုင္နက္ console ေအာက္တြင္ ေအာက္ပါ အတိုင္း javacript ျဖင့္ေရးသားထားေသာ code မ်ားကို ျမင္ရပါမည္ ။



Third Step အေနျဖင့္ Image တစ္ခုထည့္ပါမည္ ထိုသို့ ျပဳလုပ္ရန္ public folder ေအာက္တြင္ img ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါ ထို folder ေအာက္တြင္ မိမိ ထည့္လိုေသာ ပံုတစ္ပံု ထည့္ပါ။ ပံုထည့္ရန္ my computer ကေန ျပီး မိမိ node program မ်ားေရးသားထားေသာ folder ထဲ ထိ ဝင္ပါ ထို့ေနာက္ img folder ထဲတြင္ ထည့္ထားပါ ။Directory မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္



ထို့ေနာက္ index.html file ထဲတြင္ image အတြက္ link ခ်ိတ္ေပးဂ့ွန္ လိုအပ္ပါသည္ ။ <img ဆိုသည့္ tag ကိုသံုးရမည္ ျဖစ္ျပီး ထို tag ထဲတြင္ src=" image location "ကို ထည့္ေပးရပါမည္။ image ထည့္ေသာ အခါတြင္ ထို image extension ကို အတိအက် ထည့္ေပးရပါမည္ ။ example program မွာ ေအာက္ပါ အတိုင္း ျဖစ္သည္။

```
<head>
      <link rel="stylesheet" href="./css/styles.css">
      <script src ="/js/app.js"></script>
   </head>
   <body>
      <h1>This is From Static File</h1>
      <img src="/img/img.JPG">
   </body>
```

ထို့ေနာက္ program အား save ျပီး browser ကို refresh လုပ္ျကည့္လွ်င္ မိမိတို့ ထည့္ထားေသာ Image ေပၚသည္ကိုေတြ့ျမင္ရုံပါသည္ ။ ထို image အား မိမိလိုခ်င္ေသာ size အေနအထားကို styles.css ထဲတြင္ ျပင္ဆင္ပါသည္။ ထိုသို့ ျပဳလုန္ styles.css ထဲသို့ သြားျပီး image အတြက္ tag ျဖစ္ေသာ img ကိုသံုးျပီး curly bracket ထဲတြင္ မိမိ လိုခ်င္ေသာ width အား ထည့္ေပးရုံသည္ ။ example program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
h1{
   color:grey;
}
img{
   width: 500px;
}
```

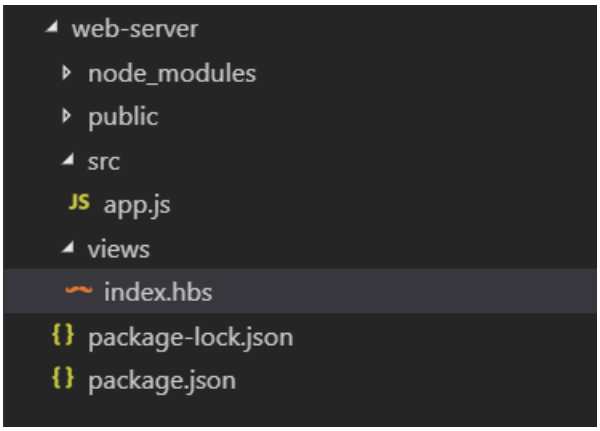
အထက္ပါ အတိုင္းေရးျပီး save က brwser အား refresh လုပ္ျကည့္ပါက မိမိ image မွာ width အေနျဖင့္ 500px ရွိသည္ကို ျမင္ရပါမည္။

Dynamic Pages with Templating

ယခင္သင္ခန္းစာတြင္ static webpage တည္ေဆာက္ခဲ့ျပီး ယခု သင္ခန္းစာတြင္ Dynamic pages အျဖစ္ တည္ေဆာက္သြားပါမည္။ထို့ေသာက္တည္ေဆာက္ hbs ကို အသံုးျပဳရန္ လိုအပ္ပါ သည္။ handlebar hbs version ကို https://www.npmjs.com/package/hbs ယခု link တြင္ ျကည့္ရုံျပီး ယခု current version သည္ 4.0.4 ျဖစ္သည္ ။ hbs ကို install လုပ္ဂ့ွန္ npm i hbs@4.0.4 ကို run ေပးပါ ျပီးလွ်င္ hbs ကို express နွင့္ ေပါင္းျပီး စတင္ အသံုးျပဳရန္ program ထဲတြင္ app.set() ကို ေရးရမည္ျဖစ္ျပီး ထို set function ထဲတြင္ setting နွင့္ value နွစ္ခု ပါဝင္ပါမည္ ။ setting တြင္ view engine ဟု အတိအက် ေရးရပါမည္ ထို့သို့ေရးမွသာ erpress က သိမည္ျဖစ္သည္ ။ value ေနရာတြင္ မိမိတို့ အသံုးျပဳမည္ျဖစ္ေသာ hbs ကို ထည့္ေပးရပါမည္။ program မွာ ေအာက္ပါ အတိုင္း ျဖစ္သည္။

```
app.set('view engine','hbs')
```

ထို့ေနာက္ handlebars file မ်ား ထည့္ဝွန္web-server ဆိုသည့္ folder ေအာက္တြင္ views ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါမည္။ ထို views fodler ထဲတြင္ index.hbs ဆိုသည့္ handlebars extension ျဖင့္ hbs file တစ္ခု တည္ေဆာက္ပါမည္ ။ Directory မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။



ထို့ေနာက္ index.html ထဲမွ code မ်ားအားလံုးကို copy လုပ္ျပီး index.hbs ထဲတြင္ ထည့္ပါ။ ျပီးလွ်င္ src folder ထဲမွ app.js file ထဲတြင္ app.get function ကိုသြားေရးပါမည္ ။ get functon ကို အထက္တြင္ အေသးစိတ္ ေဖာ္ျပခဲ့ျပီးျဖစ္သည္ ။ ယခင္ သင္ခန္းစာမ်ားက app.get ထဲတြင္ res.sendကို သံုးေသာ္လည္း ယခု သင္ခန္းစာတြင္ res.render ကို အသံုးျပဳပါမည္ ။ program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္ပါသည္ ။ index.html ကိုဖ်က္ပါ။

index.hbs ထဲတြင္ ေရးထားေသာ example program

```
<!DOCTYPE html>
<html>
   <head>
      <link rel="stylesheet" href="./css/styles.css">
      <script src ="/js/app.js"></script>
   </head>
   <body>
      <h1>Handlebars</h1>
   </body>
</html>
```

app.js ထဲတြင္ေရးထားေသာ program

```
const express=require('express')
const path=require('path')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
app.set('view engine', 'hbs')
app.use(express.static(publicDirecotryPath))
app.get('', (req, res) => {
   res.render('index')
})

app.listen(3000,()=>{
```

```
console.log('Server is up on port 3000.')
})
```

Localhost:3000 တြင် run ျကည့ပါက Handlebars ဆိုသည့ စာသားကိုျမင္ရပါမည္။

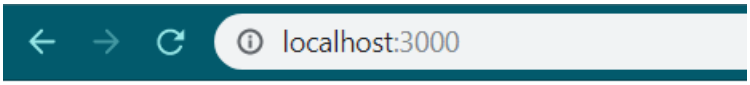
First Step အေနျဖင့ render ထဲတြင် ဒုတိယ argument အေနျဖင့ object တစ္ခု ထပ္ထည့ပါမည္ ။ ထို object ထဲတြင္ access လုပ္ရန္ values မ်ားကိုေရးသားေပးထားပါမည္ ။ program တြင္ ေအာက္ပါ အတိုင္း ျဖည့စြက္ေပးရန္လိုအပ္ပါသည္ ။ title name age high မ်ားကို မိမိ စိတ္ျကိုက္ထည့္နိုင္ သက့ဲသို႕ အျခားေသာ value မ်ားလည္း ထပ္ထည့္နုိ္ပါသည္ ။

```
app.get('', (req, res) => {
    res.render('index',{
        title:'Weather App',
        name:'Win Htut',
        age:24,
        high:5.9
    })
})
```

index.hbs ထဲတြင္ အထက္ပါေဖာ္ျပခဲက့တြင္ ပါဝင္ေသာ object မွ value မ်ားကို ေခၚသံုးရန္ ေအာက္ပါ အတိုင္း ေရးေပးရပါမည္။ curly bracket တြဲနွ႕ကြင္း နွစ္ခု ျကားတြင္ မိမိ ေခၚခ်င္ေသာ value ကို ေရးေပးရပါမည္ ။

```
<body>
        <h1>{{title}}</h1>
        <p>Create by{{name}}</p>
        <p>At the age of {{age}}</p>
        <p>More Infor High{{high}}</p>
    </body>
```

အထက္ပါ program မ်ားအား သက္ဆိုရာ file မ်ားတြင္ျဖည့စြက္ျပီး run ျကည့ပါက ေအာက္ပါ အတိုင္း ျမင္ေတြ႕ရပါမည္။



Weather App

Create byWin Htut

At the age of 24

More Infor High5.9

program run ေသာ အခါခ်ိန္တြင္ error တစ္စံုတစ္ရာ တက့ဲ့ပါက terminal မွ web-server folder ေအာက္သို႕သြားျပီး nodemon src/app.js ဟု ေရးျပီး run ေပးပါက အဆင္ေျပမည္ျဖစ္သည္ ။ ေအာက္တြင္ ပံုနွင့ တကြ ေဖာ္ျပထားသည္ ။

```
C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server>nodemon src/app.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node src/app.js`
Server is up on port 3000.
[nodemon] restarting due to changes...
[nodemon] starting `node src/app.js`
```

Second Step အေနျဖင့္ နဂိုမူရင္းရွိေသာ index.html ,about.html , help.html မ်ားကို ဖ်က္ျပီး ထိုေနရာတြင္ about.hbs and help.hbs file မ်ားကို အစားထိုးေရးသားပါမည္။ ေရးသားနည္းမွာ အထက္က index.hbs နွင့္ဆင္တူပါသည္ ။ example source code မ်ားကို ေအာက္တြင္ ေဖာ္ျပထားပါသည္။

```
const express=require('express')
const path=require('path')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
app.set('view engine', 'hbs')
app.use(express.static(publicDirecotryPath))
app.get('', (req, res) => {
    res.render('index',{
        title:'Weather App',
        name:'Win Htut',
        age:24,
        high:5.9
    })
})
app.get('/about', (req, res) => {
    res.render('about',{
        title:'Chief Instructor at Green Hackers',
        name:'Win Htut',
        age:24,
        high:5.9
    })
})
app.get('/help', (req, res) => {
    res.render('help',{
        title:'This is about Help Information',
        location:'Myanmar',
        region:'Yangon,Mandalay,PyinOoLwin'
    })
})
```

```
app.listen(3000,()=>{
console.log('Server is up on port 3000.')
})
```

Custominzing the Views Directory

Views directory ကို မိမိတို့ လိုအပ္သလို ျပဳျပင္ပါမည္ ။ ပထမဆံုး အေနျဖင့္ views folder name အား rename လုပ္ျပီး templates ဟု နံမည္ေျပာင္းေပးပါမည္ ။ ထို့ေနာက္ program အား run ျကည့္ပါက error တက္ပါမည္ ။ ျပန္လည္ေျဖရွင္းရန္ ခိန္းလုံက္ေသာ templates folder အား directory ေပးဖို့လိုအပ္ပါသည္။ public folder အား path ေပးသကဲ့သို့ path.join ကိုအသံုျး ျပဳပါမည္။ join function ထဲတြင္ __dirname တစ္ခုနွင့္ '../templates' ဆိုသည့္ arguments နွစ္ခု ပါဝင္ပါမည္ ။

```
const viewsPath=path.join(__dirname,'../templates')
```

ထို့ေနာက္ app.set တြင္ hbs ကို directory ေပးသကဲ့သို့ views နွင့္ viewPath အား ထည့္ေပးရန္ လိုအပ္ပါသည္။

```
app.set('views',viewsPath)
```

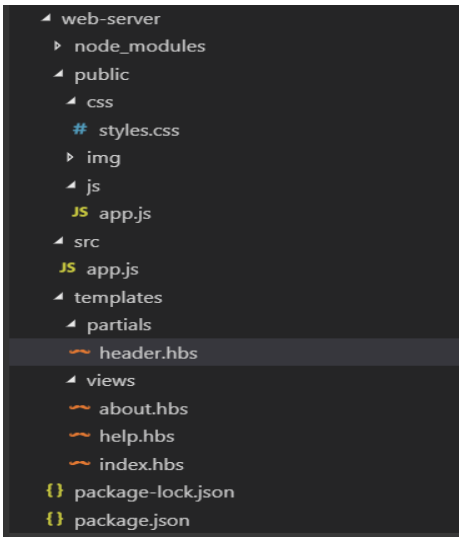
အေပၚပိုင္း program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const express=require('express')

const path=require('path')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
const viewsPath=path.join(__dirname,'../templates')

app.set('view engine', 'hbs')
app.set('views',viewsPath)
app.use(express.static(publicDirecotryPath))
```

ယခု အတိုင္းေရးျပီး run ျကည့္ပါက error တက္သည္ကို ျမင္ရပါမည္ ။ error တက္သျမ်ား အေနျဖင့္ web-server/src/ ထဲထိ သြားျပီး nodemon app.js ကို run ေပးလွ်င္ ျပန္လည္ အဆင္ေျပပါမည္။

Advanced Templating

First Step အေနျဖင့္ templates folder ေအာက္တြင္ partials and views folder နွစ္ခု တည္ေဆာက္ပါ views ထဲသို့ about.hbs , help.hbs,index.hbs မ်ားကို ေရွ့ပါ။ partials folder ေအာက္တြင္ header.hbs ဆိုသည့္ hbs file တစ္ခု တည္ေဆာက္ပါ။ directory မွာ ေအာက္ပါ အတိုင္း ျဖစ္သည္ ။



ထို႕ေနာက္ header.hbs file ထဲတြင္ h1 element တစ္ခု တည္ေဆာက္ျပီး ထို element ထဲတြင္ မိမိ နွစ္သက္ရာ ကိုေရးသားပါ ။

```
<h1>Static Header.hbs</h1>
```

ျပီးလွ်င္ help.hbs ထဲတြင္ title ေရးထားေသာ h1 element ကိုဖ်က္ျပစ္ကာ {{>header}} ဆိုျပီး ေရးသားပါမည္ ။ ေအာက္တြင္ example program အျဖစ္ ျဖည့္စြက္ေရးသားထားပါသည္ ။

```
<body>
    {{>header}}
    <p>Our service at  {{location}}</p>
    <p>we are at {{region}}</p>

</body>
```

ယခု program အားလံုးအား save ျပီး src directory ေအာက္တြင္ nodemon app.js ကို run ျကည့္ပါ။ထို႕ေနာက္ localhost:3000/help ကို သြားျကည့္ပါက ေအာက္ပါ အတိုင္း erro messages မ်ားကို ျမင္ေတြ႕ ရေပမည္။


```
Error: Failed to lookup view "help" in views directory "C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\templates"
    at Function.render (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\application.js:580:17)
    at ServerResponse.render (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\response.js:1012:7)
    at app.get (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\src\app.js:27:9)
    at Layer.handle [as handle_request] (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\layer.js:95:5)
    at next (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\route.js:137:13)
    at Route.dispatch (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\route.js:112:3)
    at Layer.handle [as handle_request] (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\layer.js:95:5)
    at C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\index.js:281:22
    at Function.process_params (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\index.js:335:12)
    at next (C:\Users\MAT\Documents\nodeprogram\nodeprogram\web-server\node_modules\express\lib\router\index.js:275:10)
```

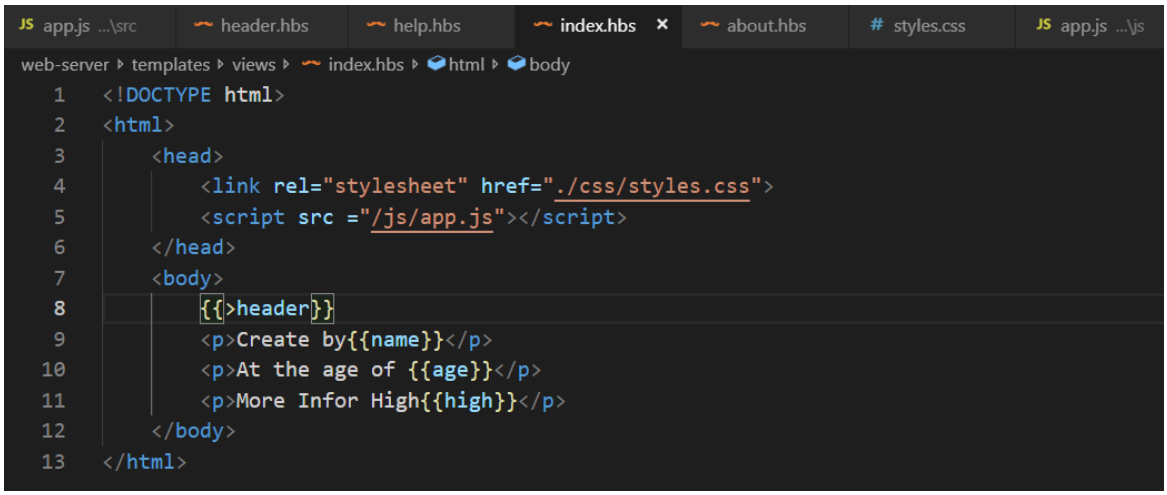
အဘယ္ေျကာင့္ ဆိုေသာ folder မ်ားေအာက္သို႕ hbs file မ်ား ေရွ႕လိုက္ေသာ္လည္း path.join ထဲတြင္ directory မ်ား အတိအက် မေရးေပးရေသးေသာ ေျကာင့္ျဖစ္သည္။ ထိုကေျကင့္ app.js ထဲတြင္ templates ထဲမွ အသစ္ေဆာက္လိုက္ေသာ views fodler အတြက္ ျပင္ဆင္ခ်က္ တစ္ခုနွင့္ partials fodler အတြက္လည္း path အသစ္ ေရးေပးရပါမည္ ။ ထို႕ျပင္ hbs ထဲမွ registerPartials function ကိုေခၚသံုးမည္ ျဖစ္တဲ့ အတြက္ const hbs=require('hbs') ကိုလည္း အထက္တြင္ ေျကျငာေပးဖို႕ လိုအပ္ပါသည္။ အရင္ path မ်ားတြင္ express() ကို ကိုယ္စားျပဳေသာ app ထဲမွ set function ကိုသံုးျပီး path ေပးခဲ့ျကသည္။ ယခုအခ်ိန္တြင္ handlebars ကိုသံုးမည္ျဖစ္တဲ့အတြက္ hbs ကို ေျကျငာေပးရျပီး ထို hbs ထဲမွ registerPartials ဆိုသည့္ function ကို ေခၚသံုးပါသည္ ။ registerPartials function

သည် hbs path ကို ေ့ျကျငာရာတြင် အသံုးျပဳပါသည်။ ထပ်မံ ျဖည့်စြက်ထားေသာ program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည် ။

```
const express=require('express')
const path=require('path')
const hbs=require('hbs')
const app=express()
const publicDirecotryPath=path.join(__dirname,'../public')
const viewsPath=path.join(__dirname,'../templates/views')
const partialPath=path.join(__dirname,'../templates/partials')

app.set('view engine', 'hbs')
app.set('views',viewsPath)
hbs.registerPartials(partialPath) // for handlebars path
app.use(express.static(publicDirecotryPath))
```

file အား လံုးအား save ျပီး web-server directory ေအာက္တြင္ nodemon src/app.js ကို run ျကည့္ပါ ။ localhost:3000/help ကို ဆက္သြားျပီးလွ်င္ Static header.hbs ဆိုတာကို စာလံုး အျကီးျဖင့္ျမင္ရလွ်င္ ေအာင္ျမင္ပါသည္ ။ ထိုသို့ မဟုတ္ပဲ error ဆက္တက္ေနပါက variable name မ်ား path မ်ားကို ေသခ်ာ အတိအက် စစ္ေဆးေပးရပါမည္။

Second Step အေနျဖင့္ index.hbs ႏွင့္ about.hbs မွ h1 element မ်ားကို ဖ်က္ျပစ္ျပီး help.hbs မွာကဲ့သို့ partials header မ်ားကိုလိုက္ထည့္ပါမည္။



app.js ...\src | header.hbs | help.hbs | index.hbs | about.hbs | styles.css | app.js ...\j

web-server ▸ templates ▸ views ▸ about.hbs ▸ html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <link rel="stylesheet" href="./css/styles.css">
5      </head>
6      <body>
7          {{>header}}
8          <p>My Name is {{name}}</p>
9          <p>My Age is {{age}}</p>
10         <p>My High is {{high}}</p>
11         <img src="img/img.JPG">
12     </body>
13 </html>
```

အထက္ပါအတိုင္းေရးသားျပီး program အားလံုးအား save ပါ ထို႕ေနာက္ localhost တြင္ about and index ကိုစစ္ျကည့္ပါက စာသားမ်ား ေျပာင္းလဲသြားသည္ကို ျမင္ရပါမည္။ page တစ္ခုခ်င္းစီရဲ႕ သက္ဆိုင္ရာ title ကိုသာ ေပၚေစခ်င္သည့္ အတြက္ header.hbs ထဲတြင္ title ကို ေရးပါမည္။အဲ့ဒါကို အတိုင္းျဖစ္ပါသည္ ။

```
<h1>{{title}}</h1>
```

ထို႕ေနာက္ page တြင္ link မ်ား ခံတြဲပါမည္ ။

localhost:3000/about

Chief Instructor at Green Hackers

RootPage About Me Help

My Name is Win Htut

My Age is 24

My High is 5.9

Header.hbs file ထဲတြင္ ေရးရမည့္ example code မ်ားကို ေအာက္တြင္ေဖာျပထားပါသည္။

```
<h1>{{title}}</h1>
<div>
  <a href="/">RootPage</a>
```

```
  <a href="/about">About Me</a>
  <a href="/help">Help</a>
</div>
```

Header ျပီးေနာက္ footer အတြက္ ထပ္ထည့္ရန္ partial folder ေအာက္တြင္ footer.hbs ဆိုသည့္ file တစ္ခုကို ထပ္ေဆာက္ပါမည္။ထို file ထဲတြင္ မိမိ ေရးသားလိုေသာ စာသားမ်ားကို ထည့္သြင္း နိုင္ပါသည္။ example အေနျဖင့္ name ကို ထည့္ပါမည္။

```
<h3>Created by{{name}}</h3>
```

Footer.hbs ကို index.hbs , about.hbs and help.hbs file မ်ားတြင္လည္း ေအာက္ပါ အတိုင္း လိုက္ထည့္ ေပးရပါမည္။ ထိုမွသာ page တုိင္းတြင္ ပါဝင္ေနမည္ျဖစ္သည္။ example အေနျဖင့္ index.hbs တြင္ ေရးျပထားပါသည္။

```
<body>
      {{>header}}
      <h1>Handlebars</h1>
      {{>footer}}
  </body>
```

အထက္ပါ အတိုင္း program အားလံုးအား save ျပီး run ျကည့္ပါ သက္ဆိုရာ page မ်ားတြင္ name မ်ား footer.hbs ထဲတြင္ ပါဝင္ေသာ အခ်က္ အလက္မ်ား အားလံုးျပပါဝင္ေနပါမည္။

404 Page Not Found

404!

RootPage About Help

Created byWinHtut

OrganizationGreenHackersTeam

404 page ေဆာက္ျခင္း ရည္ရြယ္ခ်က္မွာ user မွ မရွိေသာ url တစ္ခုအား request လုပ္လာလွ်င္ မိမိ ေဖာ္ျပလိုေသာ စာသားမ်ားကို ျပန္လည္ ေဖာ္ျပရန္ျဖစ္သည္ ။

First Step အေနျဖင့္ 404 page not found အား ေရးသားရန္ views folder ေအာက္တြင္ 404.hbs ဆိုသည့္ file ကို တည္ေဆာက္ပါ ထို file ထဲတြင္ ေအာက္ပါ အတိုင္းေရးသားပါ။

```
<!DOCTYPE html>
<html>
  <head>
```

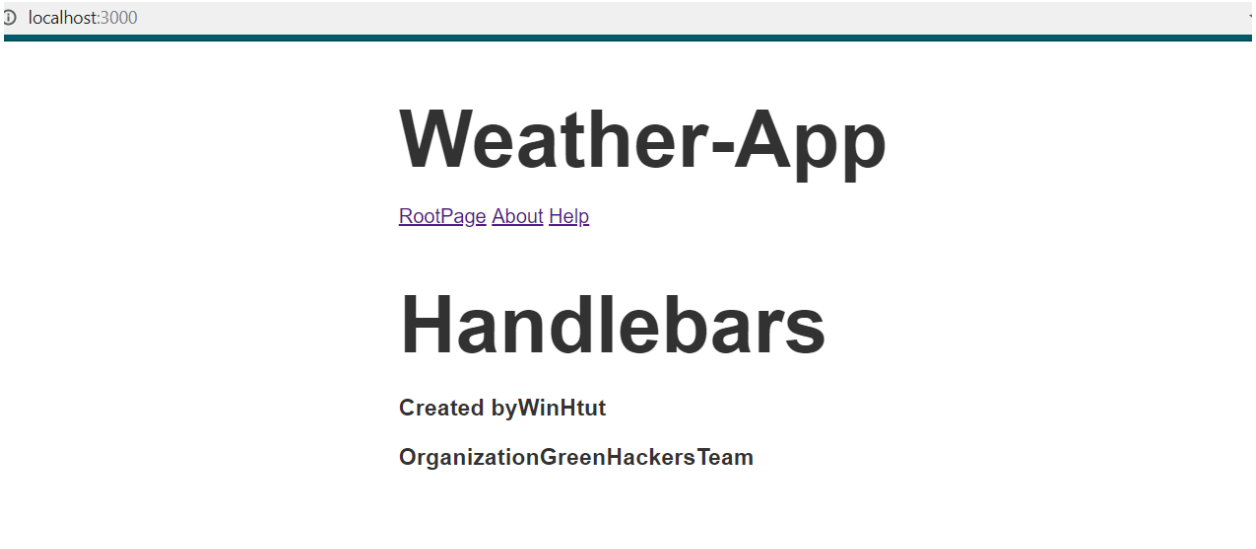
```
    <link rel="stylesheet" href="./css/styles.css">

  </head>
  <body>
     {{>header}}
    {{>footer}}
  </body>
</html>
```

ထို့ေနာက္ src folder ထဲမွ app.js ထဲတြင္ ေအာက္ပါ code မ်ားကို ထပ္မံ ေဂ်းသားေပးပါ။

```
app.get('*',(req,res)=>{
  res.render('404',{
    title:'404!',
    name:'WinHtut',
    org:'GreenHackersTeam',

  })
```

app.get ေနာက္မွ ပထမ argument ေနရာတြင္ * တစ္လုံးထည့္ထားျခင္းမွာ user မွ မရွိေသာ url မ်ားကို request လုပ္လာလွ်င္ သိရန္ျဖစ္သည္ ။

Styling The Applicatoin



မိမိတို့ page အား အထက္ပါ အတိုင္း ပံုစံ ခိန္းလိုပါက styles.css file ထဲတြင္ မူရင္း ရွိေသာ code အားလံုးကို ဖ်က္ျပီး ေအာက္တြင္ပါေသာ code မ်ားအား အစားထိုးနိုင္ပါသည္။

```
body{
  color:#333333;
```

```
    font-family: Arial;
    max-width: 650px;
    margin: 0 auto;
    padding: 0px
}
h1{
    font-size: 64px;
    margin-bottom: 16px;
}
```

Header ျပီးေနာက္ footer အတြက္ ထပ္ထည့္ဖုန္ partials folder ေအာက္တြင္ footer.hbs ဆိုသည့္ file တစ္ခုကို ထပ္ေဆာက္ပါမည္ ။ ထို file ထဲတြင္ မိမိ ေရးသားလိုေသာ စာသား မ်ားကို ထည့္ပါမည္ ။ example အေနျဖင့္ name ကို ထည့္ပါမည္။

Accessing API From Browser(Weather App)

The Qurey String

Cleint ဘက္မွ ထည့္ေပးလိုက္ေသာ query string မ်ားကို server ဘက္မွ ဖမ္းျပီး ျပန္ပို့ သင့္ေသာ respond မ်ားကို ျပန္ပို့မွာျဖစ္ပါတယ္။

First Step အေနျဖင့္ app.js ထဲတြင္ app.get တစ္ခုကို respon အတြက္ ေရးေပးရပါမည္။

```
app.get('/products',(req,res)=>{
    res.send({
        products:[]
    })
})
```

အရင္ သင္ခန္းစာမ်ားတြင္ res.render ကိုသံုးခဲ့ေသာ္လည္း ယခု သင္ခန္းစာတြင္ res.send function ကိုသာ သံုးပါမည္။ ထိုသို့ ေရးျပီးေနာက္ program အား save ျပီး run ျကည့္ပါ ။ url တြင္ http://localhost:3000/products ယခု အတိုင္းသြားျကည့္ပါ prduct ဆိုျပီး array တစ္ခု ေပၚလာ ရပါမည္။ ထို့သို့ ေပၚမလာပါက အရင္ program ေတြ ေရးထားေသာ app.get(‘*’) ကို ဖ်က္ျပစ္ရပါမည္။ သို့မွသာ product array ေပၚလာမည္ျဖစ္သည္။ ထို့သို့ မဖ်က္ျပစ္ခဲ့ပါက app.get(‘/*’) ကို ေအာက္ဆံုးမွာ သြားေရာက္ ေရးေပးရမွာ ျဖစ္ပါတယ္။ ဆက္လက္ျပီး url တြင္ မိမိတို့ search လုပ္ခ်င္ေသာ query string မ်ားကို ထပ္ထည့္ပါမည္။ထို့သို့ ထပ္ထည့္ရာတြင္ ပထမဦးဆံုး အေနျဖင့္ products ေနာက္တြင္ ? question mark ပါရပါမည္။example အေနျဖင့္ ecommerce site တစ္ခုမွ rating 5 ခု ရထားေသာ အေကာင္းဆံုး game တစ္ခုအား ရွာေဖြမည္ဆိုပါစို့ example ကို ေအာက္တြင္ ျပထားပါသည္။

http://localhost:3000/products?search=games&rating=5

ထိုကဲ့သို့ client ဘက္မွ ထည့္ေပးလိုက္ေသာ query string မ်ားကို server ဘက္မွ ျပန္လည္ရယူရန္ req.query ကို သံုးရပါမည္။ client ဘက္မွ ထည့္ေပးလိုက္သည့္ မ်ားကို console တြင္ ျပန္လည္ျကည့္ ရႈနိုင္ရန္ ေအာက္ပါ အတိုင္းေရးသားနုိင္ပါသည္။

```
app.get('/products',(req,res)=>{
    console.log(req.query)
    res.send({
            products:[]
    })
})
```

ယခု အတိုင္း save ျပီး http://localhost:3000/products?search=games&rating=5 url တြင္ run ျကည့္ပါ web page တြင္ products array ကိုသာ ေတြ့ရမည္ျဖစ္ျပီး console တြင္ ျကည့္ပါက user ထည့္ေပးလိုက္ေသာ အခ်က္ အလက္မ်ားကိုျမင္ရပါမည္။

```
{ search: 'games', rating: '5' }
```

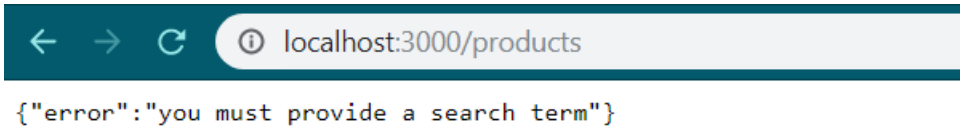
ထို့ေနာက္ req.query.rating ဟု program ထဲတြင္ ျပင္ျကည့္ပါက console တြင္ output အေနျဖင့္ 5 ကို ျမင္ေတြ့ရပါမည္။

```
console.log(req.query.rating)
```

Second Step အေနျဖင့္ Client ဘက္မွ url ကို products ဟု ေတာင္းလာလွ်င္ error message တစ္ခုအား ျပေပးရန္ if statement နွင့္ res.send မ်ားကို အသံုးျပဳပါမည္ program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္

```
app.get('/products',(req,res)=>{
    if(!req.query.rating){
        res.send({
            error:'you must provide a rating term'
        })
    }
    console.log(req.query.rating)
    res.send({
            products:[]
    })
})
```

Req.query.rating ကိုရွာတာ မဟုတ္လွ်င္ error messange ကို ေဖာျပေပးဂွန္ ေရးသားထားျခင္းျဖစ္ သည္။



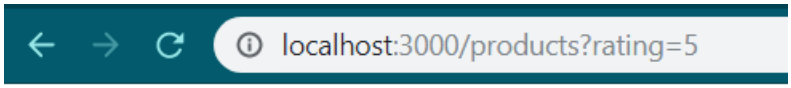
Console တြင် ျကည့္ပါက ေအာက္ပါ အတိုင္း error မ်ား တက္ေနသည္ကို ေတြ႕ရပါမည္။

```
Error [ERR_HTTP_HEADERS_SENT]: Cannot set headers after they are sent to the client
    at ServerResponse.setHeader (_http_outgoing.js:470:11)
    at ServerResponse.header (C:\Users\User\Documents\Nodejs\node_modules\express\lib\response.js:771:10)
    at ServerResponse.send (C:\Users\User\Documents\Nodejs\node_modules\express\lib\response.js:170:12)
    at ServerResponse.json (C:\Users\User\Documents\Nodejs\node_modules\express\lib\response.js:267:15)
    at ServerResponse.send (C:\Users\User\Documents\Nodejs\node_modules\express\lib\response.js:158:21)
    at app.get (C:\Users\User\Documents\Nodejs\node-test\src\app.js:58:9)
    at Layer.handle [as handle_request] (C:\Users\User\Documents\Nodejs\node_modules\express\lib\router\layer.js:95:5)
    at next (C:\Users\User\Documents\Nodejs\node_modules\express\lib\router\route.js:137:13)
    at Route.dispatch (C:\Users\User\Documents\Nodejs\node_modules\express\lib\router\route.js:112:3)
    at Layer.handle [as handle_request] (C:\Users\User\Documents\Nodejs\node_modules\express\lib\router\layer.js:95:5)
^CTerminate batch job (Y/N)? y
```

ထိုကဲ့သို႕ error မ်ား တက္ျခင္းမွာ http resquet သည္ တစ္ျကိမ္ resquet လုပ္ပါက္ တစ္ျကိမ္သာ respond လုပ္ရသည္ ယခု program app.get('/product') ထဲတြင္ တစ္ျကိမ္သာ resquet လုပ္ေသာ္လည္း ွစ္ျကိမ္ respond လုပ္ထားေသာေျကာင့္ျဖစ္သည္။ထို error ကို ေျဖရွင္းရန္ res.send ေရွ႕တြင္ return ကို ခံေပးလိုက္ျခင္းျဖင့္ ေျပလည္သြားမည္ျဖစ္ပါသည္။

```
app.get('/products',(req,res)=>{
    if(!req.query.serach){
        return res.send({
            error:'you must provide a search term'
        })
    }
```

Program ထဲတြင္ return ကို ထပ္ျဖည့္ျပီး save လုပ္ပါ url ကို localhost:3000/products ကို သြားျကည့္ပါ အျကိမ္ျကိမ္ refresh လုပ္ျကည့္ပါ ။ console တြင္ အရင္ ကဲ့သို႕ error မတက္ေတာ့ပဲ run ေနသည္ကို ေတြ႕ရပါမည္။ ထို႕ေနာက္ url တြင္ ေအာက္ပါ url အတိုင္း ဆက္ run ျကည့္ပါ products ဆိုသည့္ output ကို ျမင္ရပါမည္။



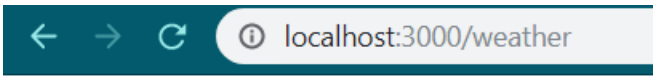

```
{"products":[]}
```

Third Step အေနျဖင့္ weather route တစ္ခုကို app.js ထဲတြင္ ထပ္ေဆာက္ပါမည္။

```
app.get('/weather',(req,res)=>{
    if(!req.query.address){
        return res.send({
            error:'You must provide an address'
        })
    }
    res.send({
        forecast:'Snowing',
        location:'Pyin Oo Lwin',
```

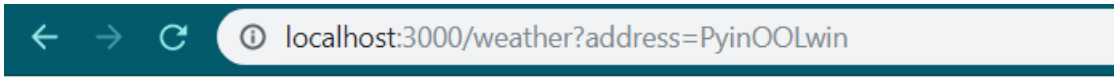
```
        address: req.query.address
    })
})
```

အထက္ပါ အတိုင္း ေရးျပီး program အား run ျကည့္ပါ




```
{"error":"You must provide an address"}
```

weather တစ္ခုတည္းသာ url တြင္ရိုက္မည္ဆိုလွ်င္ you must provide an address ဆိုသည့္ စာသားကို ေဖာ္ျပန္ if ေနာက္တြင္ req.query.address လို႕ ေရးထားျပီး ထိုေရွ႕တြင္ not ! ကို ေ႐ွးထားပါသည္။

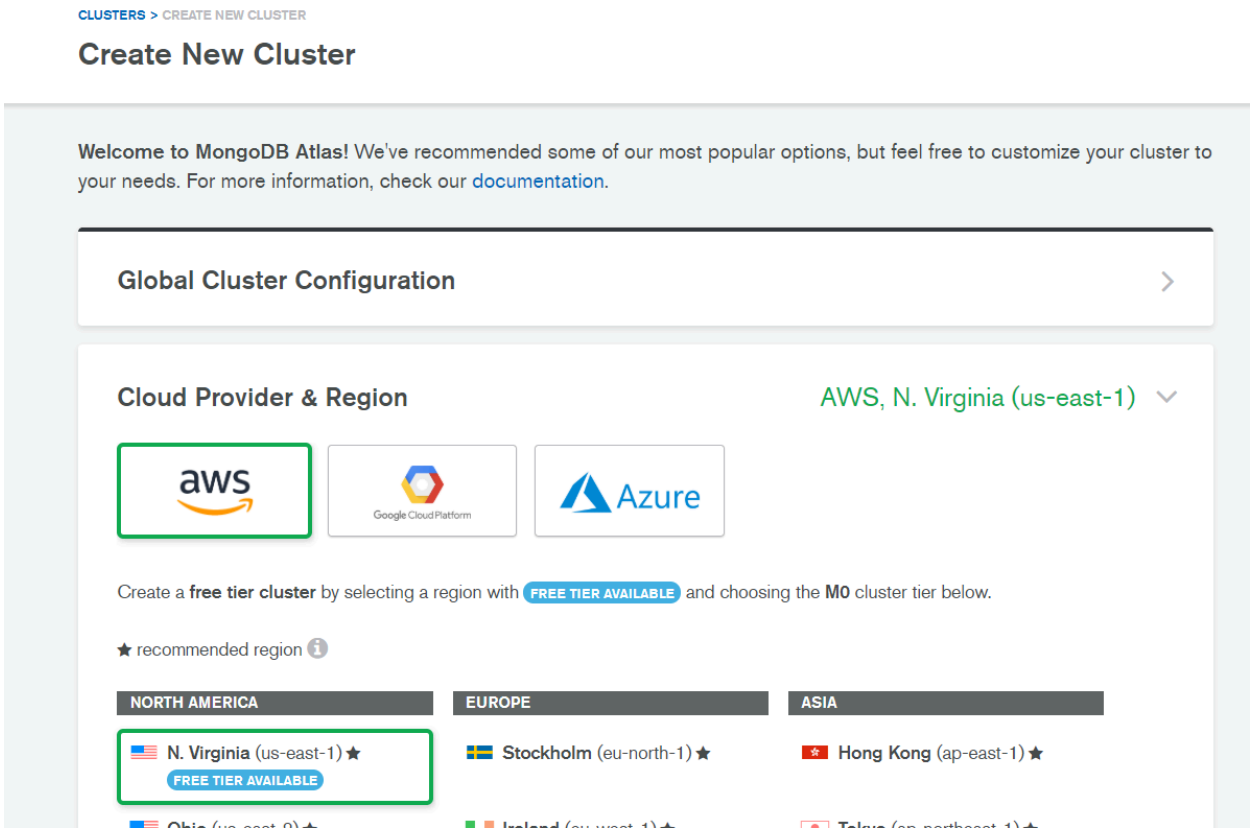



```
{"forecast":"Snowing","location":"Pyin Oo Lwin","address":"PyinOOLwin"}
```

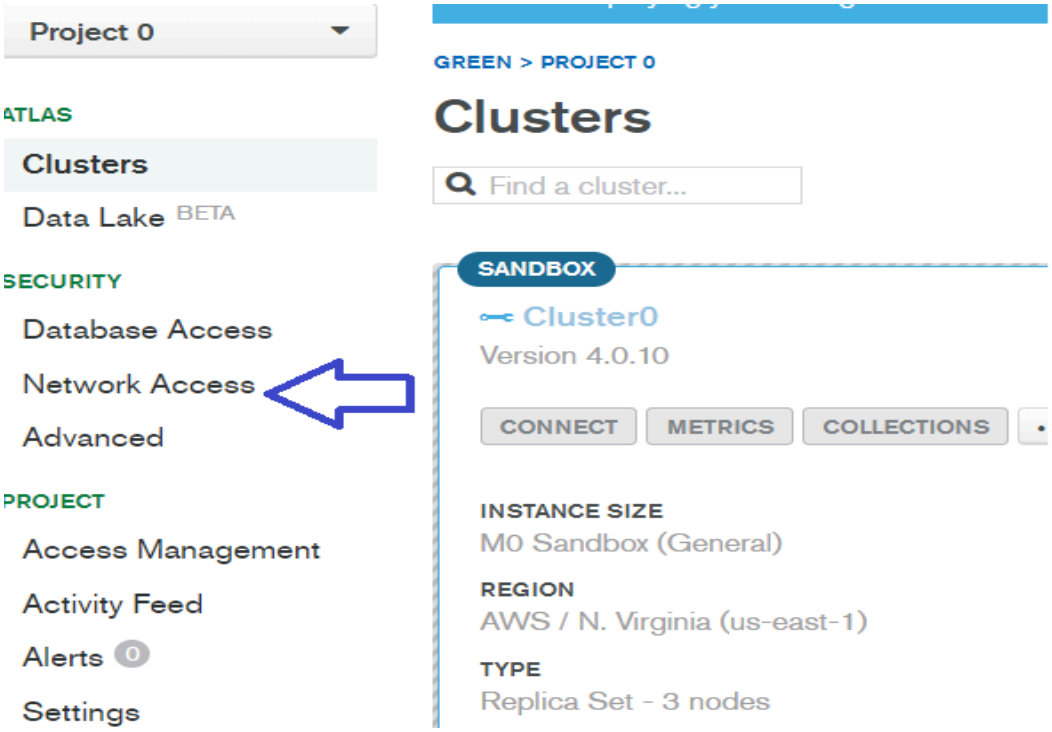
url တြင္ end point ? ေနာက္မွာ address=PyinOOLwin လို႕ ရိုက္ထည့္လိုက္လွ်င္ ထိုရိုက္ထည့္လိုက္ေသာ ျမို႕နံမည္ကို ျပန္ရန္ res.send ထဲတြင္ address: req.query.address ကို ေရးထားေပးျခင္းျဖစ္ပါသည္။ web page တြင္ ျကည့္ပါ address PyinOOLwin ကိုျမင္ရပါမည္။

Connecting MongoDB Atlas With Nodejs

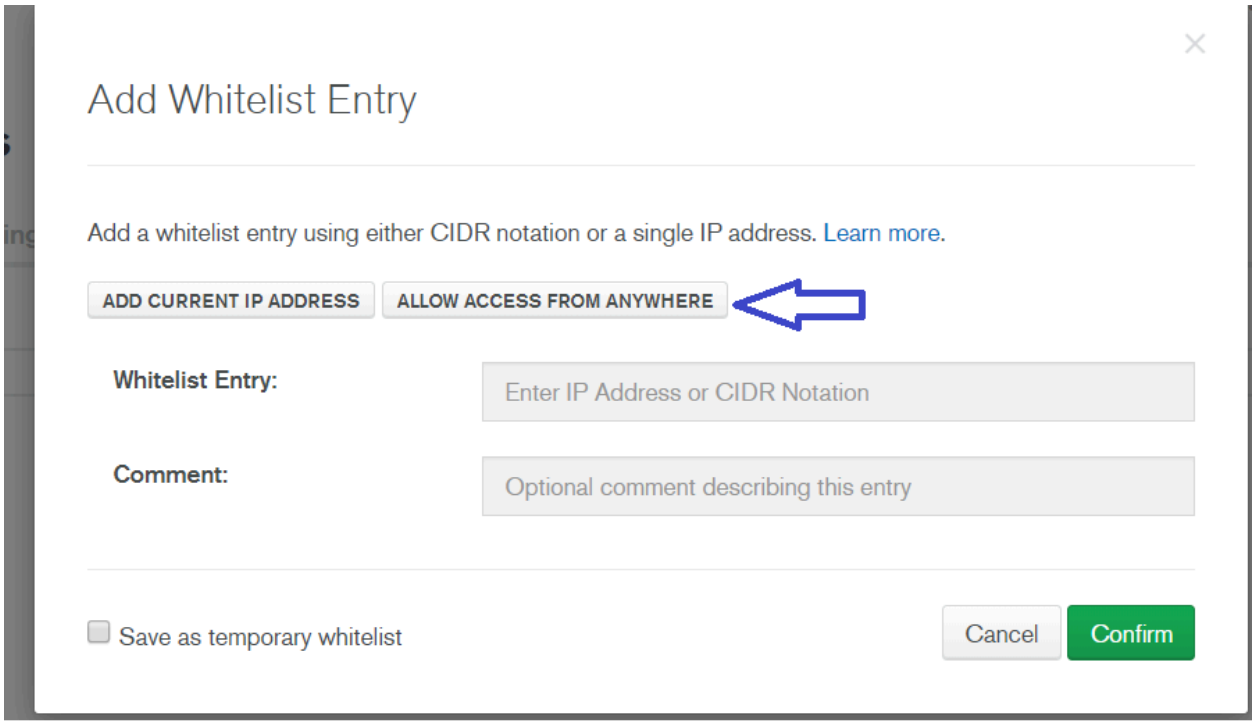
First Step အေနျဖင့္ MongoDB Atlas ျဖင့္ ခ်ိတ္ဆက္ရန္ https://www.mongodb.com/cloud/atlas သို႕ သြားျပီး account တစ္ခု ေဖာက္ရန္ လိုအပ္ပါသည္။ account ေဖာက္ျပီးသည့္ေ့ာ္ တစ္ျပိဳင္နက္ ေအာက္ပါ page သို႕ေရာက္သြားမည္ ျဖစ္ျပီး မည္သည့္ setting ကိုမွ ျပင္ဆင္စရာလိုအပ္ပါဘူး ထို႕ေနာက္ ေအာက္ဘက္တြင္ Cluster0 ဆိုသည့္ ေန႕ရာတြင္ မိမိ ျကိုက္သည့္ နံမည္ကို ထည့္ေ့ာ္ပါသည္ ။ ထို႕ေနာက္ ေအာက္ဆံုးတြင္ Create Cluster ကို နွိပ္ပါ ။



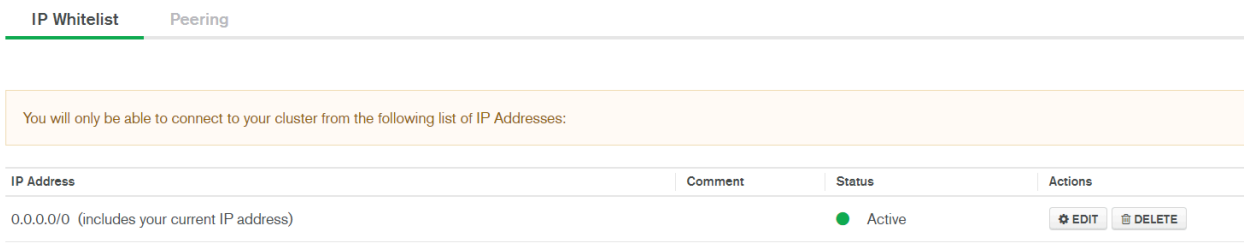
Create Cluster ကိုနှိပ်ပြီးသည့်နောက် တစ်ပြိုင်နက် ဘေးကပါ စာမ်ကွာသို့ ရောက်သွားမည်ဖြစ်ပြီး Network Access ထဲသို့ ဝင်ပါမည်။



MongoDB သည် မိမိ ထည့်ေပးလုိက်မည့် ip address ကို မွတ်ထားမည်ျဖစ်ျပီး ေနာက်တစ်ခ်ိန်တြင်လည်း ထို ip address မွ ျပန်ဝင် သာလွင် database ကို access လုပ်ျခင်း ရမည်ျဖစ်သည်။ ထို့ေနာက် ADD IP ADDRESS ဆိုသည်ကိုနှိပ်ပါ ျပီးလွင် ေအာက်ပါ အတိုင်း page တစ်ခု ဖ်လာပါမည်။ ထိုေနရာတြင် setting နှစ်ခုရှိပါသည် ပထမ တစ်ခုသည် ADD CURRENT IP ADDRESS ျဖစ်ျပီး ဒုတိယ တစ်ခုသည် ALLOW ACCESS FROM ANYWHERE ျဖစ်သည် ။ပထမ တစ်ခုသည် မိမိ ယခု လက်ရှိ IP ADDRESS ကိုမွတ်ထားမည်ျဖစ်ျပီး တစ်ျခား ေသာ IP ADDRESS မ်ားမွ ဝင်ေရာက် လက်ခံမည် မဟုတ်ပါ။ ယခု သင်ခန်းစာတြင် ALLOW ACCESS FROM ANYWHERE ကို အသံုးျပ်မည်ျဖစ်ပါသည်။

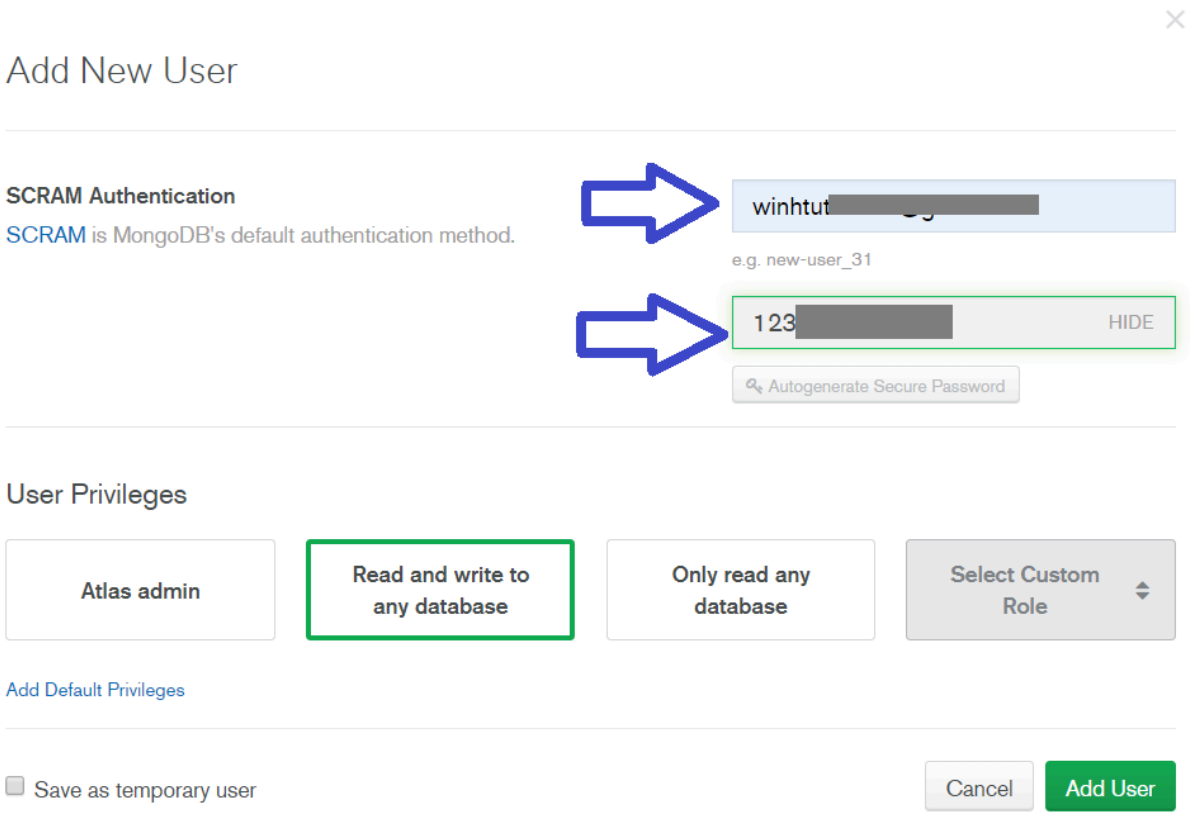


ထို့ေနာက် Whitelist Entry တြင် 0.0.0.0/0 ဆိုသည့် address တစ်ခု ေပ့လာပါမည် ျပီးလွင် Confirm ကို နှိပ်ပါ။ ျပီးလွင် အခ်ိန် ခန ေစာင့်ေပးျပီး ေအာက်ပါ အတိုင်း ေပ့လာလွင် network access setting ျပီးပါျပီ။



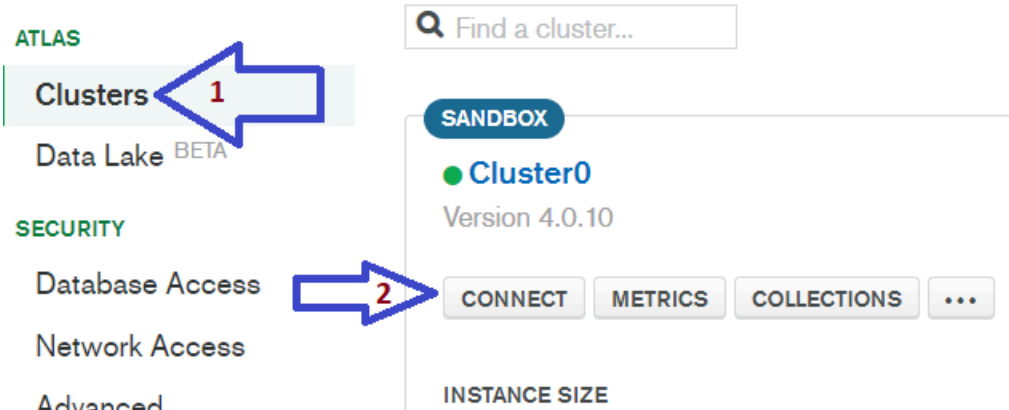
ထို့ေနာက် Database Access ထဲသို့ ဝင်ပါမည်။ +ADD NEW USER ထဲသို့သြားပါမည်။ username and password ကို မိမိ အဆင်ေျပတဲ့ အတိုင်း ေပးနုိင်ပါသည်။

သတိျပဳရမည့္အခ်က္မွာ name ကို special character မ်ားထည့္ျပီးေပးလို့ မရသလို password ကို character အျကီး တစ္လုံး ေလာက္ ထည့္ေပးရပါမည္။ ထို password ကို ေသခ်ာ မွတ္ထားရမည္ ျဖစ္သည္။ database ကို Node မွ လွမ္းခ်ိတ္ေသာ အခါတြင္ ထို password ကိုျပန္လည္ အသံုးျပဳရမည္ျဖစ္ပါသည္။



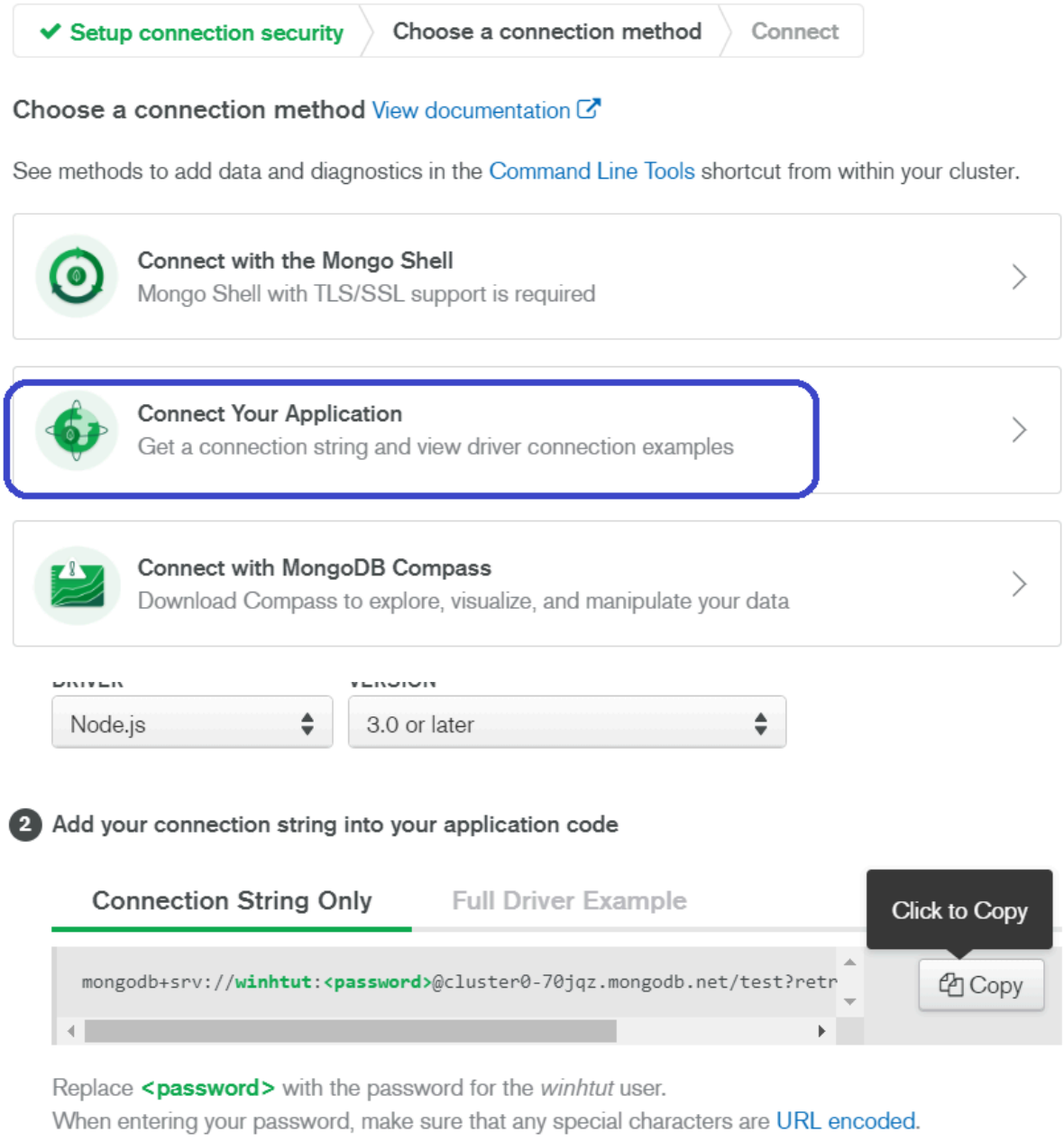
ျပီးလွ်င္ Add User ကို နွိပ္ပါ။ ေအာက္ပါ အတိုင္း ေပၚလာလွ်င္ ေအာင္ျမင္ပါျပီ။

ထို့ေနာက္ Clusters ထဲသို့ ဝင္ပါ ျပီးလွ်င္ ေအာက္ပါပံု အတိုင္း CONNECT ထဲသို့ ဝင္ပါ။



Clusters ထဲသို့

ဝင္ျပီးေနာက္ connect ထဲသို့ ဆက္ဝင္ပါ။ connect ထဲမွ program ထဲမွ access လွမ္းလုပ္ဂုန္ link ယူရမည္ျဖစ္ သည္။ ေအာက္ပါပံုထဲ တြင္ျပထားပါသည္။ Connect Your Applicaton ထဲ သို့ ဆက္ဝင္ပါ



ထို့ေနာက္ copy ကို နွိပ္ပါ။ ျပီးလွ်င္ node program တစ္ခုျဖင့္ပါ file name ကို app.js ဆို့ျပီး ေပးထားပါသည္ ။ ပထမဆံုး အေနျဖင့္ mongodb အား install ျပဳလုပ္ရန္ npm i mongodb ကို သံုးရပါမည္ ။ mondodb ကို install လုပ္ျပီးေနာက္ ေအာက္ပါ code မ်ားကိုေရးသားပါ။

```
const MongoClient = require('mongodb').MongoClient;
```

```
// replace the uri string with your connection string.
const uri
="mongodb+srv://winhtut:123wh@cluster0-70jqz.mongodb.net/test?retryWrites=true&w
=majority"
MongoClient.connect(uri,{useNewUrlParser:true}, function(err, client) {
  const collection = client.db("GH").collection("web");
  console.log("connected");
  var insert={name:'winthtu',email:'loveyou@gmail.com'};

  collection.insertOne(insert,function(err,res){
     console.log("data inserted");
  })
   client.close();
});
```

မိမိတို့ copy ကူးလားေသာ link ကို uri ေနာက္က “ ” ထဲတြင္ ထည့္ေပးရပါမည္။ ထို copy လုပ္လာေသာ စာေျကာင္းထဲမွ 123wh ေနရာတြင္ မိမိတို့ ေပးခဲ့ေသာ password ကို ျပန္ထည့္ေပးရပါမည္ ။ ထို့ေနာက္ ယခု program အား run ျကည့္ပါ terminal တြင္

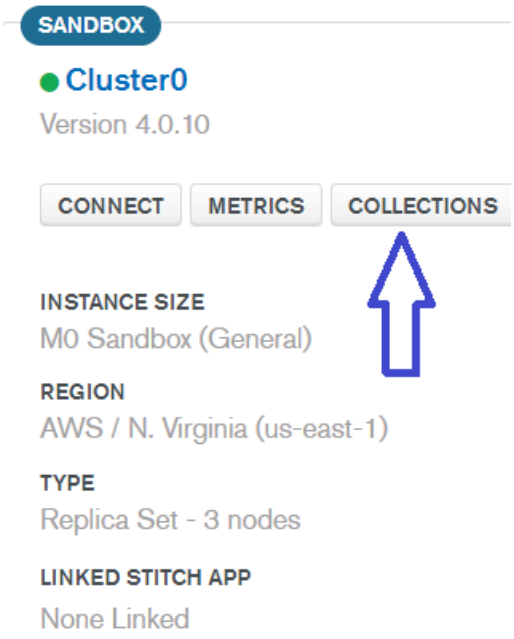
Connected

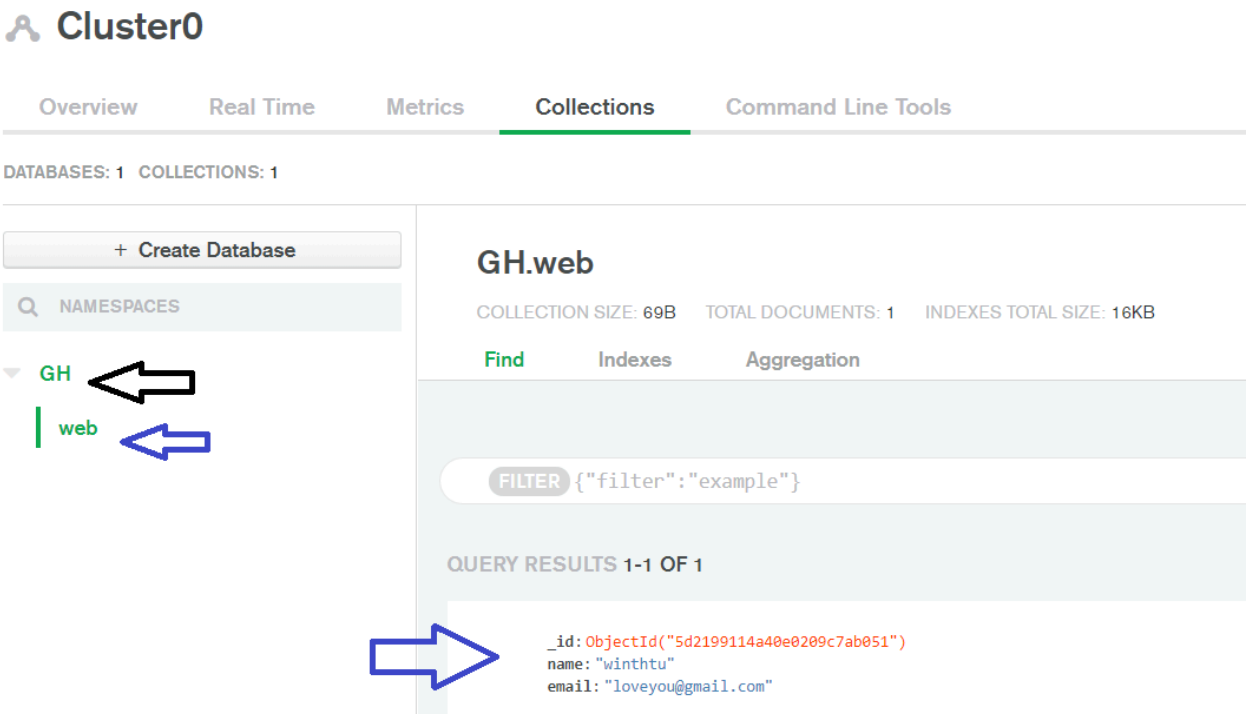
Data inserted ဆိုသည့္ စာသားမ်ားကို ျမင္ရလွ်င္ step အားလံုး ေအာင္ျမင္ပါသည္။ ထိုသို့ မေအာင္ျမင္ပါ step by step အားလံုးကို ေသခ်ာ ျပန္စစ္ေဆးရမည္ျဖစ္သည္။ password ထည့္ေသာ ေနရာတြင္ အေကာင့္ ေဖာက္ေသာ အခါက password နွင့္ database access လုပ္ေသာ ေနရာက password ျဖင့္ မွားတတ္ပါသည္ ။ ကြ်န္ေတာ္တို့ program တြင္ အသံုးျပဳမည့္ password မွာ database access လုပ္ေသာ အခ်ိန္က ေပးထားေသာ password ျဖစ္ပါသည္။

Second Step program ကို တစ္ေျကာင္း၌င္းစီ ရွင္းလင္းပါမည္ ။ mongodb အား သံုးမွာ ျဖစ္ေသာ ေျကာင့္ program line 1 တြင္ require(‘mongodb’).MongoClient ကို ေျကျငာထား ပါသည္။ line 4 တြင္ uri ဆိုသည့္ variable တစ္ခုကို တည္ေဆာက္ျပီး ထို variable ထဲတြင္ မိမိတို့ copy ကူးလာေသာ link ကို ထည့္ထားပါသည္ ။ ထို link ထဲမွ password ေနရာတြင္ မိမိတို့ database access setting ထဲတြင္ ေပးခဲ့ေသာ password ကို ထည့္ေပးရပါမည္ ။ line 5 တြင္ MongoClient ထဲမွ connect ဆိုသည့္ function ကို ေခၚသံုးထားျပီး ထို function ထဲတြင္ argument သံုးခု ပါဝင္ပါသည္ ။ ပထမ တစ္ခုသည္ uri ျဖစ္ျပီးဒုတိယ တစ္ခုသည္ useNewUrlParser ျဖစ္ပါသည္ ထို useNewUrlParser ကို true ေပးခဲ့ရန္ လိုအပ္ပါသည္။ တတိယ တစ္ခုသည္ function တစ္ခု ျဖစ္ျပီး ထို function သည္ mongodb ကို ခ်ိတ္ဆက္ရာတြင္ error တစ္စံုတစ္ရာ တက္ခဲ့လွ်င္ error message ကို terminal တြင္ ျပန္လည္ ေဖာ္ျပေပးမည္ ျဖစ္ျပီး error မတက္ပါက instruction အတိုင္း ဆက္လက္ လုပ္ေဆာင္သြားမွာ ျဖစ္ပါတယ္။ line 6 မွ client.db() function ထဲမွ GH သည္ database name ျဖစ္ျပီး collection function ေနာက္မွ web သည္ sub name ျဖစ္သည္ ။ program

ဆိုလိုရင်းမှာ Cluter ထဲတြင် GH ဆိုသည့် name ဖြင့် database တစ်ခု တည်ဆောက်မည်ဖြစ်ပြီး GH ဆောက်ထဲတြင် web ဆိုသည့် collection တစ်ခုကို တည်ဆောက်လိုက်ခြင်းဖြစ်သည်။ ထို web ဆိုသည့် collection ထဲတြင် client ဘက်မှ ထည့်ပေးလိုက်သော data များကို သိမ်းဆည်း ထားမည် ဖြစ်သည် ။ line 7 သည် connected ဖြစ်စင် စာသား ပေဖော်ပြပေးဂွန် ရေးသားထားခြင်း ဖြစ်သည်။line 8 တြင် insert ဆိုသည့် variable တစ်ခုကို တည်ဆောက် လိုက်ပြီး ထို variable ထဲတြင် မိမိ ထည့်လိုသော data များကို ထည့်ထားနိုင်ပါသည်။ line 10 တြင် insertOne ဆိုသည့် function ကိုသုံးပြီး မိမိတို့ ထည့်ခဲ့သော data များ ထည့်ထားသည့် insert ဆိုသည့် variable ကို argument တစ်ခု အနေဖြင့် ထည့်ပေးလိုက်ပါသည် ။ ထို insertOne ဆိုသည့် function ထဲတြင် ဒုတိယ argument အနေဖြင့် function တစ်ခု ပါဝင်ပြီး ထို function သည် error ကို check လုပ်ပြီး error တက်ခဲ့ပါက error message ကို ပေဖော်ပြပေးမည်ဖြစ်သည်။ line 11 တြင် database ထဲသို့ data များ ထည့်ပြီးပါက ပေဖော်ပြ ပေးဂွန် စာပေကြောင်း ထည့်ပေးထားခြင်းဖြစ်သည်။ line 13 မှ client.close() သည် ယခု ခိတ်ဆက်ထားသော database db connection ကို ပိတ်ရန်ဖြစ်ပါသည် ။

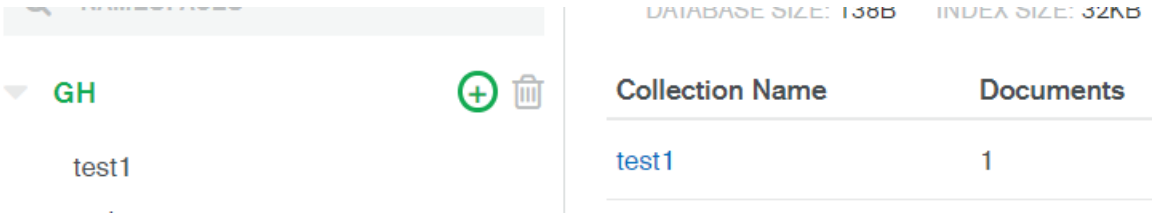
Third Step အနေဖြင့် Cluter ထဲမှ database ထဲတြင် data များ တစ်ကယ် ရောက်လာ သလား ဆိုတာကို သြားရောက် စစ်ဆေးပါမည် ထိုသို့ စစ်ဆေးရန် Cluster ထဲမှ collection ကိုသြားပါ

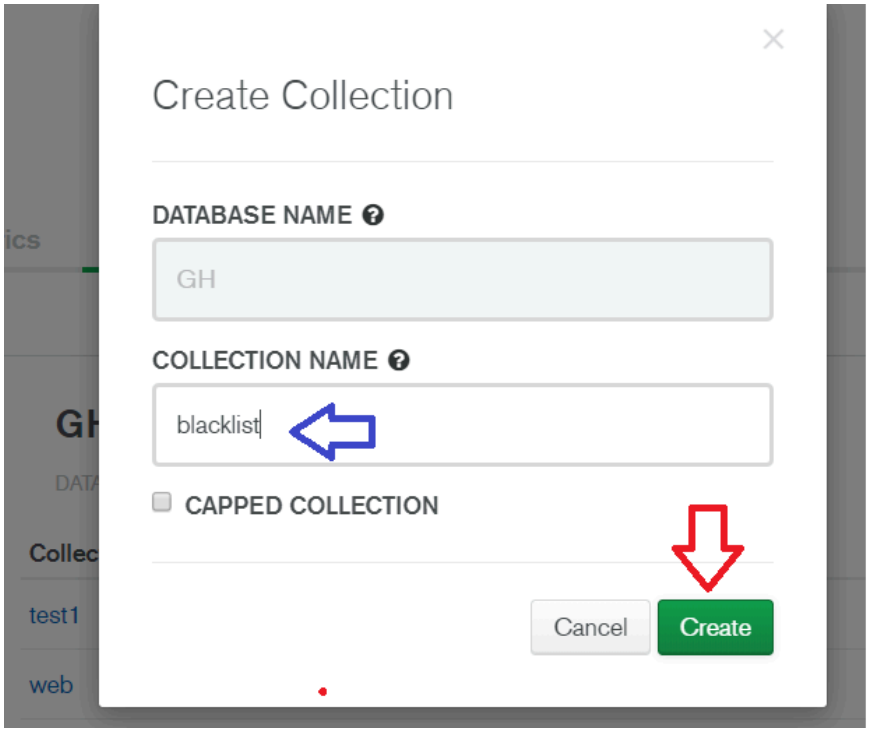


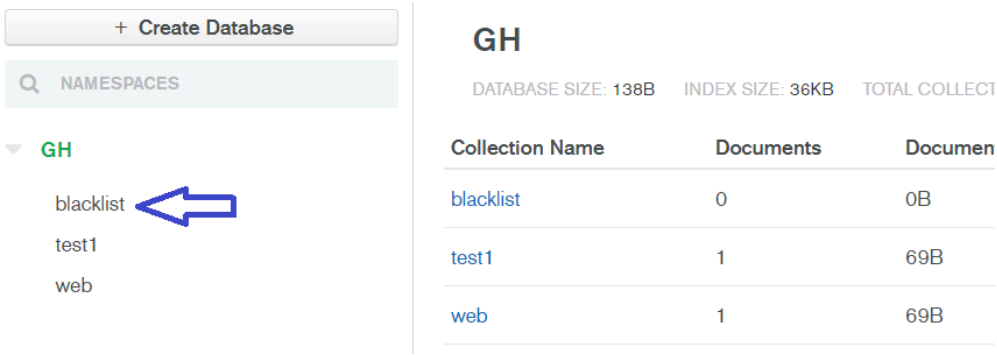


အထက်ပါ ပံုထဲမွ အတုိင်း step by step သြားျပီး မိမိတို့ထည့္လုိက္ေသာ data မ်ားကို ျမင္ရပါက mongodb atlas ထဲသို့ data insertion ျပီလုပ္ျခင္း ေအာင္ျမင္ပါသည္ ။

Fourth Step အေနျဖင့္ GH ဆိုသည့္ Database ထဲတြင္ blacklist ဆိုသည့္ collection တစ္ခု တည္ေဆာက္ပါ မည္။ထို့ သို့ တည္ေဆာက္ပုံနွ GH ေဘးမွ အေပါင္း ေလးကို နွိပ္ပါ ထို ေနာက္ form တစ္ခု ေပၚလာပါမည္ ထို form ထဲတြင္ မိမိ ထည့္လိုေသာ collection name ကို ထည့္ုိင္ပါသည္ ။ယခု program တြင္ blacklist ဆိုသည့္ collection တစ္ခုကို တည္ေဆာက္ လုိ့ရွိသည္ ။







ယခုကဲ့သို့ တည်ေဆာက်ျပီးေသာ blacklist ဆိုသည့် collection ထဲသို့ data မ်ား ထည့်ဂွန် program collection name ေနရာ မွာ blacklist ကို ခိန်လ်ြကုံသာ ျဖစ္သည် ။ program မွာ ေအာက္ပါ အတိုင်းျဖစ္သည် ။

```
const collection = client.db("GH").collection("blacklist");
```

ထို့ေနာက် GH db ထဲမွ blacklist collection အား စစ်ေဆးျကည့္ပါ data မ်ားေရာက်ေနပါမည်။

CRUD with mongo DB Atlas

First Step အေနျဖင့် crud ဆိုသည့် folder တစ္ခု ျပီလုပ္ပါ ထို folder ထဲတြင် terminal >> npm init ကိုေရးပါ ။ ထို folder ထဲတြင် package.json ဆိုသည့် file တစ္ခု ေပၚလာပါမည်။ ထို crud folder ထဲတြင္ပါ server.js ဆိုသည့် file တစ္ခု ကို တည်ေဆာက္ပါ ။ထို့ေနာက် terminal တြင်

npm i –s express mongoose body-parser express-handlebars

ေအာင္ျမင္ေျကာင္း ေအာက္ပါ စာသားမ်ားကို ျမင္ရပါမည္

+ body-parser@1.19.0

+ express-handlebars@3.1.0

+ mongoose@5.6.4

+ express@4.17.1

added 99 packages from 98 contributors and audited 230 packages in 9.629s

found 0 vulnerabilities

Crud folder ထဲတြင္ models ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါ ထို folder ထဲတြင္ db.js ဆိုသည့္ file တစ္ခု တည္ေဆာက္ပါ ထို file ထဲတြင္ ေအာက္ပါ code မ်ားကို ေရးသားပါ။

```
const mongoose=require('mongoose')
mongoose.connect('mongodb+srv://winhtut:123wh@cluster0-70jqz.mongodb.net/test?re
tryWrites=true&w=majority',{useNewUrlParser:true},(err)=>{
    if(!err){
        console.log('MongooDB Connection Succeeded')
    }else{
        console.log('error in db connection :'+err)
    }
})
```

ထို႔ေနာက္ models ထဲက db.js file ကို server.js ကေန သိေစရန္ server.js ထဲတြင္ ေအာက္ပါ အတိုင္းေရးသားေပးပါ ။

```
require('./models/db');
```

ထို႔ေနာက္ models folder ထဲတြင္ employee.model.js ဆိုသည့္ file တစ္ခုကို တည္ေဆာက္ပါ ထို file ထဲတြင္ ေအာက္ပါ code မ်ားကို ေရးသားပါ ။ mongoose db ကို သံုးမွာ ျဖစ္သည့္ အတြက္ mongoose ကို ေျကျငာေပးရန္ လိုအပ္ပါသည္။ ထို႔ေနာက္ form တစ္ခုမွ information မ်ားကို ရယူရန္ mongoose.Schema ကိုေခၚသံုးပါမည္။

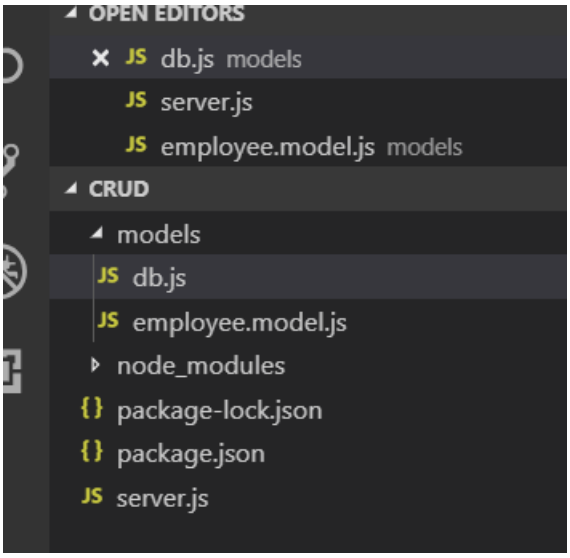
```
const mongoose = require('mongoose');

var employeeSchema = new mongoose.Schema({
    fullName: {
        type: String,
        required: 'This field is required.'
    },
    email: {
```

```
        type: String
    },
    mobile: {
        type: String
    },
    city: {
        type: String
    }
});
mongoose.model('Employee', employeeSchema);
```

model ကို အသံုးျပဳျပီး employeeSchema object ကို ေျကျငာထားေပးရန္လည္း လိုအပ္ပါသည္ ။ first parameter သည္ ထို schema name ျဖစ္ျပီး ဒုတိယ parameter သည္ object ျဖစ္သည္။ ထို နည္းအားျဖင့္ mongodb ထဲသို့ data မ်ား insert လုပ္ရန္ျဖစ္ပါသည္။ ယခု တည္ေဆာက္ခဲ့ေသာ employee.model file ကို db.js မွ သိေစရန္ ေျကျငာေပးရန္ လိုအပ္ပါသည္။ ထို့ေျကာင့္ db.js ထဲတြင္ ေအာက္ပါ အတိုင္းေရးသားရပါမည္ ။

```
const mongoose=require('mongoose')
mongoose.connect('mongodb+srv://winhtut:123wh@cluster0-70jqz.mongodb.net/test?retryWrites=true&w=majority',{useNewUrlParser:true},(err)=>{
    if(!err){
        console.log('MongooDB Connection Succeeded')
    }else{
        console.log('error in db connection :'+err)
    }
})
require('./employee.model');
```

file directory မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။



ထို့ေနာက္ terminal တြင္ node server.js ကို run ျကည့္ပါ terminal တြင္ ေအာက္ပါ စာသားမ်ား ျမင္ရပါက ေအာင္ျမင္ပါသည္ ။

C:\Users\MAT\Documents\nodeprogram\crud>node server.js

Express server started at port : 3000

MongooDB Connection Succeeded

မိမိ နွစ္သက္ရာ browser တြင္ ေအာက္ပါ အတိုင္း url ကို ေရးျကည့္ပါ

http://localhost:3000/ cannot get ကို ရရွိလာပါက မွန္ပါသည္။

cannot Get/ ကို ရ ရွိလာျခင္းမွာ route လုပ္မည့္ file ကို မေရးသား ရ ေသးေသာေျကာင့္ ျဖစ္သည္ ။

ထို႕ေနာက္ crud folder ထဲတြင္ controllers ဆိုသည့္ folder တစ္ခုကို တည္ေဆာက္ျပီး ထို folder ထဲတြင္ employeeController.js ဆိုသည့္ file တစ္ခုကို တည္ေဆာက္ပါမည္။ ထို file ထဲတြင္ route လုပ္မည္ program ကို ေရးသားပါမည္။

```
const express=require('express')
var router=express.Router()

router.get('/',(req,res)=>{
    res.json('sample text')
})

module.exports=router;
```

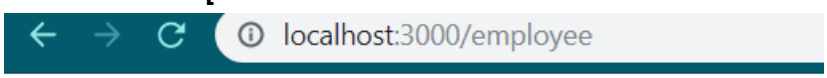
ေအာက္ဆံုး၂ုိးမွ ေဂ်ားသားထားေသာ module.exports သည္ router ကို export လုပ္ေပးထားျခင္း ျဖစ္သည္ ။ ထို႕ေနာက္ server.js ထဲတြင္ route လုပ္ရန္ ေအာက္ပါ code မ်ားကို ေရးသားရပါမည္ ။

```
const employeeController=require('./controllers/employeeController')
```

employeeController file ကို သံေစရန္ ေျကျငာ ေပးျခင္းျဖစ္သည္ ။

```
app.use('/employee',employeeController)
```

employeeController ကို route အေနျဖင့္ အသံုးျပဳျခင္းျဖစ္သည္။ terminal တြင္ nodemon ကို စတင္ အသံုးျပဳျပီး nodemon server.js ကို run ျကည့္ပါ ထို႕ေနာက္ browser url တြင္ localhost:3000/employee ကို သြားျကည့္ပါ sample text ဆိုသည့္ စာသားကို ျမင္ရလွ်င္ route file ခိတ္ျခင္ ားေအာင္ျမင္ပါသည္ ။



"sample text"

ထို႕ေနာက္ express-handlebars ကို စတင္ အသံုးျပဳရန္ server.js ထဲတြင္ ေအာ္ကပါ အတိုင္း path and express-handlebars ကို ေျကျငာေပးရန္ လိုအပ္ပါသည္ ။

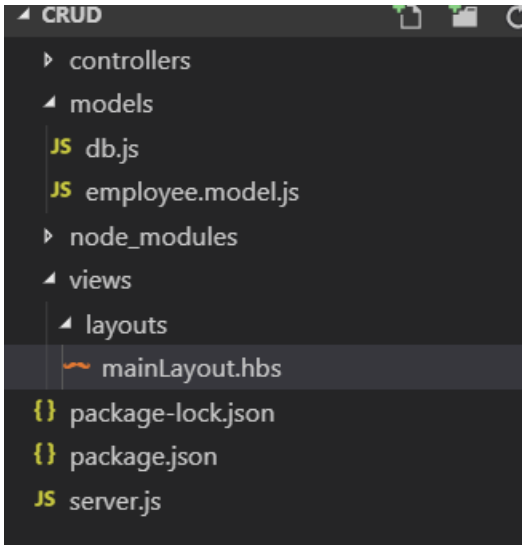
```
// 3 . for express handlebar and body parser
```

```
const path = require('path');
const exphbs = require('express-handlebars');
```

ထို႕ေနာက္ mainLayout တစ္ခုအား ဖန္တီးပါမည္ ။ mainLayout အား ဖန္တီးရန္ crud folder ေအာက္ပါ views ဆိုသည့္ folder တစ္ခုကို တည္ေဆာက္ပါမည္ ထို folder ေအာက္တြင္ layouts ဆိုသည့္ folder တစ္ခုကို တည္ေဆာက္ပါမည္ ။ ထို layouts ဆိုသည့္ folder ေအာက္တြင္ mainLayout.hbs ဆိုသည့္ hbs foder တစ္ခုကို တည္ေဆာက္ပါမည္ ။

```
// 4 . for path
app.set('views', path.join(__dirname, '/views/'));
app.engine('hbs', exphbs({ extname: 'hbs', defaultLayout: 'mainLayout', layoutsDir:
__dirname + '/views/layouts/' }));
// starting veiw engine
app.set('view engine', 'hbs');
```

အထက္ပါ code မ်ားသည္ directory name မ်ားကို အတိအက် သိေစရန္ ေရးသားထားျခင္းျဖစ္သည္ ။

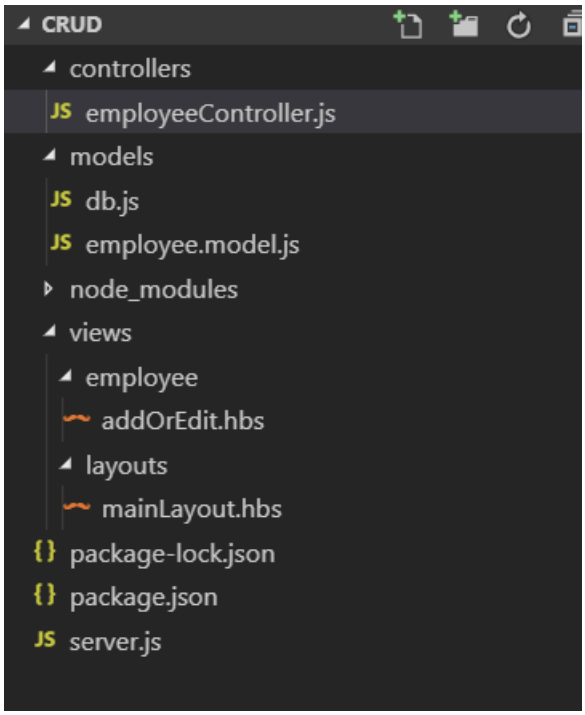


ထို႕ေနာက္ mainLayout.hbs ထဲတြင္ ေအာက္ပါ html code မ်ားကို ေရးသားပါမည္။

```
<!DOCTYPE html>

<html>

<head>
    <title>Node.js express mongDB CRUD</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFO
JwJ8ERdknLPMO"
        crossorigin="anonymous">
```

```
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.cs
s">
</head>

<body class="bg-info">
    <div class="row">
        <div class="col-md-6 offset-md-3" style="background-color: #fff;margin-top:
25px;padding:20px;">
            {{{body}}}
        </div>
    </div>

</body>

</html>
```

Views folder ေအာက္တြင္ employee folder တစ္ခုကို တည္ေဆာက္မည္ျဖစ္ျပီး ထို folder ေအာက္တြင္ addOrEdit.hbs ဆိုသည့္ folder တည္ခုကို တည္ေဆာက္ပါမည္ ။ ထို addOrEdit.hbs folder ထဲတြင္ ေအာက္ပါ code မ်ားကို ေရးသားပါမည္ ။




```
<h3>{{viewTitle}}</h3>
```

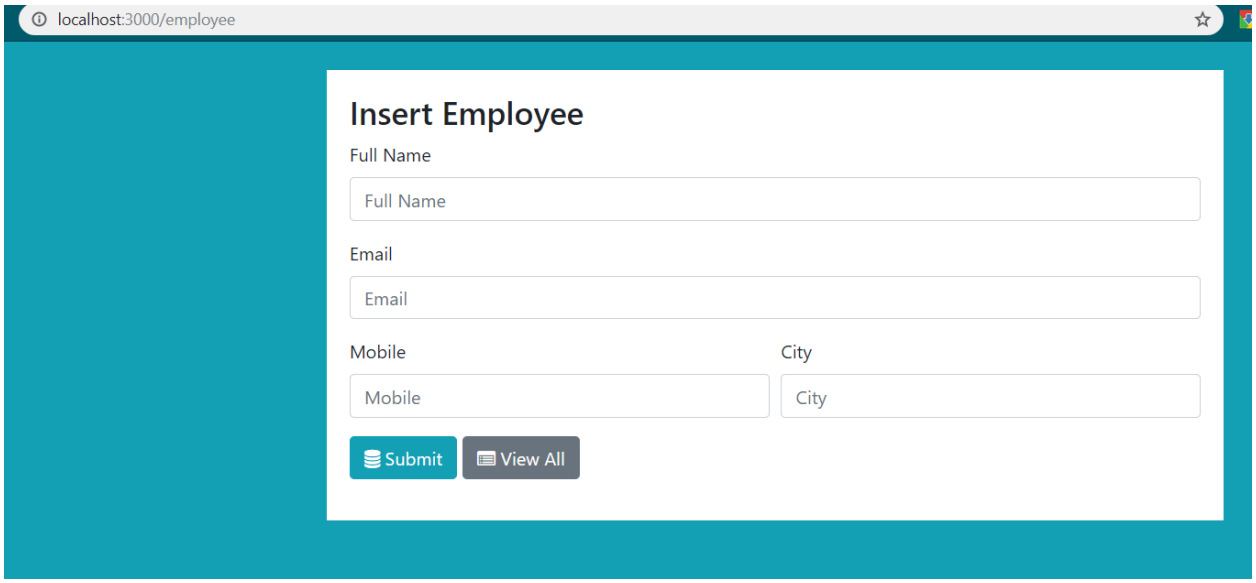
```
<form action="/employee" method="POST" autocomplete="off">
    <input type="hidden" name="_id" value="{{employee._id}}">
    <div class="form-group">
        <label>Full Name</label>
        <input type="text" class="form-control" name="fullName" placeholder="Full Name"
value="{{employee.fullName}}">
        <div class="text-danger">
            {{employee.fullNameError}}</div>
    </div>
    <div class="form-group">
        <label>Email</label>
        <input type="text" class="form-control" name="email" placeholder="Email"
value="{{employee.email}}">
        <div class="text-danger">
            {{employee.emailError}}</div>
    </div>
    <div class="form-row">
        <div class="form-group col-md-6">
            <label>Mobile</label>
            <input type="text" class="form-control" name="mobile" placeholder="Mobile"
value="{{employee.mobile}}">
        </div>
        <div class="form-group col-md-6">
            <label>City</label>
            <input type="text" class="form-control" name="city" placeholder="City"
value="{{employee.city}}">
        </div>
    </div>
    <div class="form-group">
        <button type="submit" class="btn btn-info"><i class="fa fa-database"></i>
Submit</button>
        <a class="btn btn-secondary" href="/employee/list"><i class="fa fa-list-alt"></i>
View All</a>
    </div>
</form>
```

ထို့ေနာက္ employeeController.js ထဲတြင္ code မ်ားကို ေအာက္ပါ အတိုင္း ျဖည့္စြက္ ေရးသားပါမည္

```
const express=require('express')
var router=express.Router()
```

```
router.get('/',(req,res)=>{
    res.render("employee/addOrEdit",{
        viewTitle:"Insert Employee"
    })
})

module.exports=router;
```

ထို႕ေနာက္ file အားလံုးကို save ပါ ျပီးလွ်င္ localhost:3000/employee ကို ျပန္ run ျကည့္ပါ ေအာက္ပါ အတိုင္းျမင္ရလွ်င္ အားလံုး ေအာင္ျမင္ပါသည္။



Second Step employee page url အား encode ျပဳလုပ္ရန္ ေအာက္ပါ code မ်ားကို server.js ထဲတြင္ ထပ္မံ ေရးသားပါမည္ ။

```
const bodyparser = require('body-parser');
```

body-parser သည္ url ကို encode လုပ္ရာတြင္ အသံုးျပဳပါသည္ ။

```
var app = express();
app.use(bodyparser.urlencoded({
    extended: true
}));
```

အထက္ပါ code မ်ားသည္ encode ျပဳလုပ္ရန္ ေရးသားထားျခင္းျဖစ္သည္ ။ ထို႕ေနာက္ json format ျပန္ေျပာင္းရန္ ေအာက္ပါအတိုင္းေရးေပးရပါမည္ ။

```
app.use(bodyparser.json());
```

ယခု အဆင့္ထဲ server.js ထဲမွ program အျပည့္အစံုသည္ ေအာက္ပါ အတိုင္းျဖစ္သည္။

```
require('./models/db');
const express = require('express');
```

```
// 3 . for express handlebar and body parser
const path = require('path');
const exphbs = require('express-handlebars');
// 5 . to encode url
const bodyparser = require('body-parser');

// 2 . for route file
const employeeController=require('./controllers/employeeController')

//5. to encode url
var app = express();
app.use(bodyparser.urlencoded({
    extended: true
}));
app.use(bodyparser.json());
///

var app = express();
// 4 . for path
app.set('views', path.join(__dirname, '/views/'));
app.engine('hbs', exphbs({ extname: 'hbs', defaultLayout: 'mainLayout', layoutsDir:
__dirname + '/views/layouts/' }));
// starting veiw engine
app.set('view engine', 'hbs');

app.listen(3000, () => {
    console.log('Express server started at port : 3000');
});
// 2. using route file
app.use('/employee',employeeController)
```

mongodb ထဲ သို့ data မ်ား စတင္ ထည့္ရန္ ေအာက္ပါ လိုအပ္ခ်က္မ်ားကိုေျကျငာေပးရပါမည္ ။

```
//2. for insert data to mongodb
const mongoose=require('mongoose')
const Employee=mongoose.model('Employee')
```

ထို႕ေနာက္ data မ်ားထည့္ရန္ function တစ္ခုကို စတင္ တည္ေဆာက္ပါမည္။ ထို function သည္ employee body ထဲမွ စာသားမ်ားကို ရယူေပးမည္ ျဖစ္ပါသည္ ။ ထိုသို့ ရယူ႐န္

Employee() ဆိုသည့် ကနဦးေျကျငားထားထည့် function အားေခၚသံုးရပါမည္ ။ ထို သို႕ ေခၚသံုးျပီးမွ data မ်ားကို ယူမည္ example

```
var employee = new Employee();
   employee.fullName = req.body.fullName;
```

ထို႕သို႕ ရယူျပီး employee.save အား အသံုးျပဳျပီး mongodb ထဲသို႕ data မ်ား ထည့ပါမည္ ။ထိုသို႕ ထည့ျခင္း ေအာင္ျမင္ပါက list ဆိုသည့် page တစ္ခုအားသြားရန္ res.redirect('emplyee/list') ဆိုသည့် code ကိုေရးရပါမည္ ။ error ျဖစ္ပါက error message ျပန္ျပေပးဂ္ရန္လည္း save function ထဲတြင္ err,doc ဆိုသည့် parameter နွစ္ခုကို ျပန္ျပ ေပးရပါမည္။ doc သည္ document ကိုဆိုလို ျခင္းျဖစ္သည္ ။ ပထမ ဦးဆံုး အေနျဖင့ ေအာက္ function ကို ေဂ်းရမည္

```
function insertRecord(req, res) {
   var employee = new Employee();
   employee.fullName = req.body.fullName;
   employee.email = req.body.email;
   employee.mobile = req.body.mobile;
   employee.city = req.body.city;
   employee.save((err, doc) => {
      if (!err)
         res.redirect('employee/list');
      else {
            console.log('Error during record insertion : ' + err);
      }
   });
}
```

ထို function အား ေအာက္ပါ အတို္းး ျပန္ေခၚပါမည္ ။

```
router.post('/',(req,res)=>{
   insertRecord(req,res);
})
```

Data မ်ား ထည့ျပီးလွ်င္ ျပန္ျပဂ့န္ list route ကိုလည္း ေအာက္ပါအတို္းး ေရးထားရပါမည္

```
router.get('/list',(req,res)=>{
   res.json('from list')
})
```

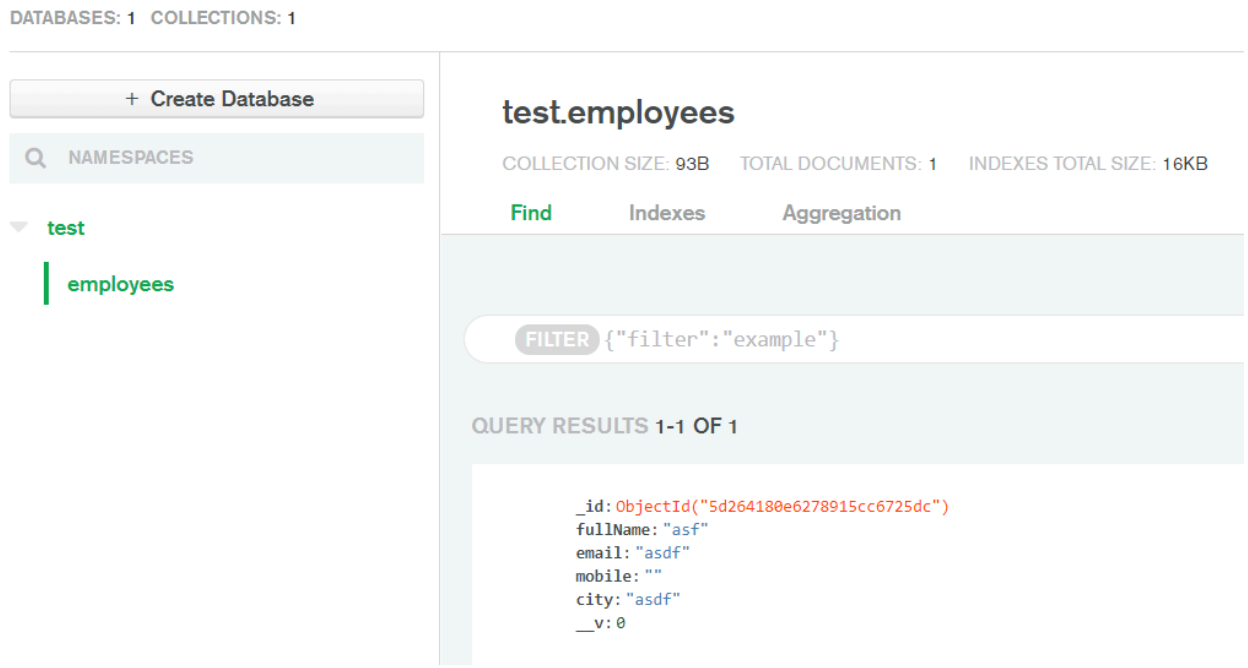
အထက္ပါ အဆင့္မ်ား အားလံုးျပီးသြားလွ်င္ employeeControllers.js ထဲရွိ code မ်ားမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const express=require('express')
var router=express.Router()
```

```
//2. for insert data to mongodb
const mongoose=require('mongoose')
const Employee=mongoose.model('Employee')

router.get('/',(req,res)=>{
    res.render("employee/addOrEdit",{
        viewTitle:"Insert Employee"
    })
})

router.post('/',(req,res)=>{
    insertRecord(req,res);
})

function insertRecord(req, res) {
    var employee = new Employee();
    employee.fullName = req.body.fullName;
    employee.email = req.body.email;
    employee.mobile = req.body.mobile;
    employee.city = req.body.city;
    employee.save((err, doc) => {
        if (!err)
            res.redirect('employee/list');
        else {
                console.log('Error during record insertion : ' + err);
        }
    });
}

router.get('/list',(req,res)=>{
    res.json('from list')
})

module.exports=router;
```

ယခု အဆင့်ထဲ အားလုံးပြီးပါက program အားလုံးကို save ပြီး run ပြကြည့်ပါ ထို့နောက် localhost:3000/employee page တြင်လည်း data မ်ားထည့်ပြီး submit လုပ်ပြကြည့်ပါ ပြီးလျှင် from list ဆိုသည့် စာသားမ်ားကို ပြမည်ပါမည်။ fullName ကို ဖတ်မရဘူးဆိုသည့် error တက်နိုင်ပါသည် ထို error ကို typing မ်ားကို သေခ်ာ ပြန်လည် စစ်ဆေးရပါမည်။ error မတက်ပါက mongodb ထဲက collection ထဲတြင် test ဆိုသည့် database name အောက်တြင် employees ဆိုသည့် collection တစ်ခုအား

ျမင္ေတြ့ရပါမည္ ထို collection အား ျကည့္ ျကည့္ပါက မိမိ ထည့္ေပးလိုက္ေသာ data မ်ားကို ျမင္ေတြ့ရပါမည္ ။



Third Step employee form ထဲတြင္ data မ်ား မျဖည့္ပါက alarm ျပေပးရန္ code မ်ားကို ေဂ်းသားပါမည္။ထို့နည္းတူ email မ်ားကို အမွားမ်ား ထည့္ပါကလည္း email မ်ားကို စစ္ေဆးဂ်န္ အတြက္ employee.model.js ထဲတြင္ ေအာက္ပါ အတို့ုိင္းေရးသားေပးရပါမည္ ။ employee.model.js ထဲမွ program အျပည့္အစံုမွာ ေအာက္ပါ အတုိင္းျဖစ္သည္ ။

```
const mongoose = require('mongoose');

var employeeSchema = new mongoose.Schema({
    fullName: {
        type: String,
        required: 'This field is required.'
    },
    email: {
        type: String
    },
    mobile: {
        type: String
    },
    city: {
        type: String
    }
```

```
});

employeeSchema.path('email').validate((val) => {
   emailRegex =
/^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\
.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
   return emailRegex.test(val);
}, 'Invalid e-mail.');
mongoose.model('Employee', employeeSchema);
```

ထို႕ေနာက္ employeeController.js ထဲတြင္ name ကို ပထမဦးစြာ စစ္ေဆးပါမည္ ထိုသို႕ စစ္ေဆးရန္ handleValidationError ဆိုသည့္ function တစ္ခုအား တည္ေဆာက္ပါမည္ ။ ထို function ထဲတြင္ field in err.errors ဆိုသည့္ err object ကို သံုးပါမည္ ဆိုလိုရင္းမွာ field ထဲတြင္ မည္သည့္ data မ်ိဳး မျဖည့္သြင္းပါက error alarm ေပးဂ်န္ျဖစ္သည္။ ထိုနည္းတူပင္ email ကိုလည္း email address တစ္ခုတြင္ပါဝင္ရမည္ အခ်က္ အလက္မ်ား မပါဝင္ပါက စစ္ေဆးေပးျပီး alarm ျပေပးဂ်န္ ေဂ်းသား ရပါမည္။ handleValidationError function မွ code မ်ားမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
function handleValidationError(err, body) {
   for (field in err.errors) {
      switch (err.errors[field].path) {
         case 'fullName':
            body['fullNameError'] = err.errors[field].message;
            break;
         case 'email':
            body['emailError'] = err.errors[field].message;
            break;
         default:
            break;
      }
   }
}
```

ထို function အား ျပန္ေခၚသံုး၍သည့္ program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။ insertRecord function ထဲမွ error တက္သည့္ေနရာတြင္ ျပန္ေခၚသံုးမည္ျဖစ္သည္။

```
function insertRecord(req, res) {
   var employee = new Employee();
   employee.fullName = req.body.fullName;
   employee.email = req.body.email;
   employee.mobile = req.body.mobile;
```

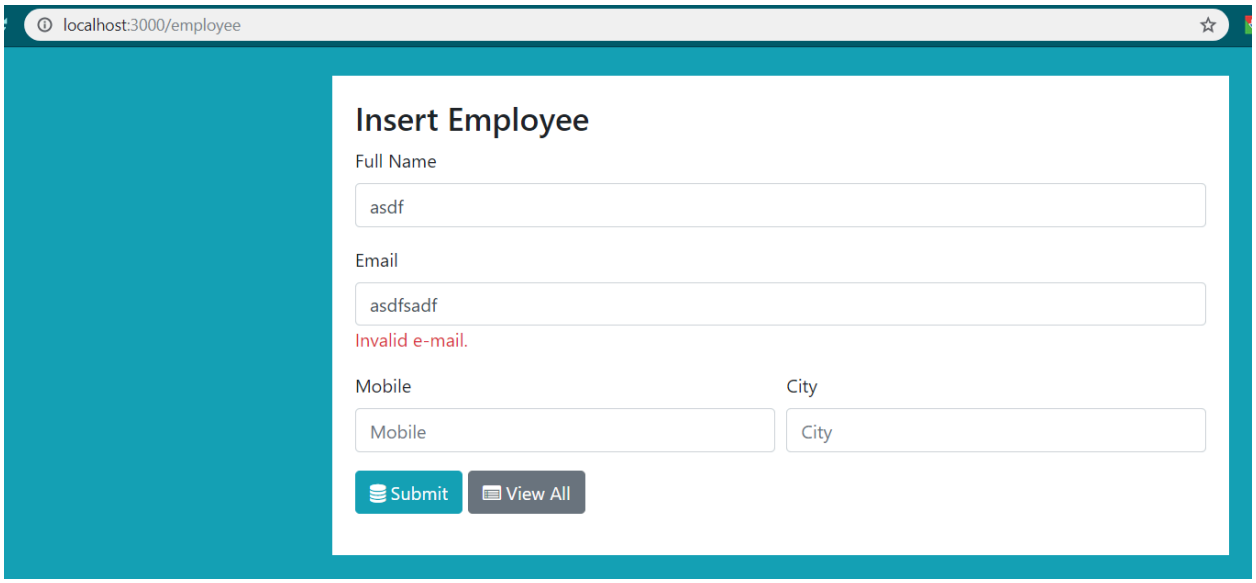
```
    employee.city = req.body.city;
    employee.save((err, doc) => {
        if (!err)
            res.redirect('employee/list');
        else {
            if (err.name == 'ValidationError') {
                handleValidationError(err, req.body);
                res.render("employee/addOrEdit", {
                    viewTitle: "Insert Employee",
                    employee: req.body
                });
            }
            else
                console.log('Error during record insertion : ' + err);
        }
    });
}
```

ယခု အဆင့်ထပ်ပြီးပါက employeeController.js ထဲတြင်ရှိေသာ program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const express=require('express')
var router=express.Router()

//2. for insert data to mongodb
const mongoose=require('mongoose')
const Employee=mongoose.model('Employee')

router.get('/',(req,res)=>{
    res.render("employee/addOrEdit",{
        viewTitle:"Insert Employee"
    })
})

router.post('/',(req,res)=>{
    insertRecord(req,res);
})

function insertRecord(req, res) {
    var employee = new Employee();
    employee.fullName = req.body.fullName;
```

```
    employee.email = req.body.email;
    employee.mobile = req.body.mobile;
    employee.city = req.body.city;
    employee.save((err, doc) => {
        if (!err)
            res.redirect('employee/list');
        else {
            if (err.name == 'ValidationError') {
                handleValidationError(err, req.body);
                res.render("employee/addOrEdit", {
                    viewTitle: "Insert Employee",
                    employee: req.body
                });
            }
            else
                console.log('Error during record insertion : ' + err);
        }
    });
}
function handleValidationError(err, body) {
    for (field in err.errors) {
        switch (err.errors[field].path) {
            case 'fullName':
                body['fullNameError'] = err.errors[field].message;
                break;
            case 'email':
                body['emailError'] = err.errors[field].message;
                break;
            default:
                break;
        }
    }
}

router.get('/list',(req,res)=>{
    res.json('from list')
})

module.exports=router;
```

ထို့ေနာက္ program အားလံုးအား save ျပီး form တြင္ အလြတ္မ်ား သို့မဟုတ္ email format မျပည့္စံုမွ် မ်ားကို ျဖည့္ျကည့္ပါ ေအာက္ပါ အတုိင္း error မ်ားကို ျမင္ေတြ့ရပါမည္။



ထို့ေနာက္ employee folder ေအာက္တြင္ list.hbs ဆိုသည့္ file တစ္ခု တည္ေဆာက္ပါမည္။ ထို file ထဲတြင္ အခ်က္ အလက္မ်ားကို ျပန္ျပ ရန္ ေအာက္ပါ code မ်ားကို ေရးသားရပါမည္

```
<h3><a class="btn btn-secondary" href="/employee"><i class="fa fa-plus"></i> Create New</a> Employee List</h3>
<table class="table table-striped">
  <thead>
    <tr>
      <th>Full Name</th>
      <th>Email</th>
      <th>Mobile</th>
      <th>City</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    {{#each list}}
    <tr>
      <td>{{this.fullName}}</td>
      <td>{{this.email}}</td>
      <td>{{this.mobile}}</td>
      <td>{{this.city}}</td>
```

```
            <td>
                <a href="/employee/{{this._id}}"><i class="fa fa-pencil fa-lg"
aria-hidden="true"></i></a>
                <a href="/employee/delete/{{this._id}}" onclick="return confirm('Are you sure
to delete this record ?');"><i class="fa fa-trash fa-lg" aria-hidden="true"></i></a>
            </td>
        </tr>
        {{/each}}
    </tbody>
</table>
```

ထို့ employeeController.js ထဲတြင် router.get('/list') ထဲတြင် ေအာက္ပါ code မ်ားအား သြားေရာက္ ေဂၚသားပါမည္ ။ ထို function ထဲတြင္ find function ကိုသံုးပါမည္ find function ကိုသံုးျခင္းအားျဖင့္ database ထဲတြင္ ရွိေသာ အခ်က္ အလက်ားအားလံုးကို ျပေပးနုိင္ပါသည္ ။ ထိုသို့ ျပေပးဂ့ုန္ docs ဆိုသည့္ parameter တစ္ခု ကို ဒုတိယ အေနျဖင့္ ထည့္ေပးရပါမည္။ ထိုသို့ ျပန္ေပးဂ့ုန္ list : docs ကို ေရးလိုက္ရံုျဖင့္ အခ်က္ အလက်ား အားလံုးကို ရပါသည္ ။ထည့္ျဖည့္ ရမည့္ program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္

```
router.get('/list', (req, res) => {
    Employee.find((err, docs) => {
        if (!err) {
            res.render("employee/list", {
                list: docs
            });
        }
        else {
            console.log('Error in retrieving employee list :' + err);
        }
    });
});
```

list ကို list.hbs ထဲတြင် ျပန္သံုးမည္ ျဖစ္ပါသည္ ။

ထို့ေနာက္ database ထဲသို့ ေက်ာ္ရိုလာေသာ data မ်ားအား update ျပဳလုပ္ရန္ findByid function ကိုသံုးပါမည္ ထို function ပထမ parameter တြင္ req.params.id ကိုသံုးမည္ျဖစ္ျပီး ဒုတိယ အေနျဖင့္ callback function ကိုသံုးပါမည္ ။ error မျဖစ္ပါက addOrEdit ေနရာသို့ ျပန္ပို့ ေပးပါမည္ ။

Program အျပည့္အစံုမွာ ေအာက္ပါ အတုိင္းျဖစ္သည္ ။

```
router.get('/:id', (req, res) => {
    Employee.findById(req.params.id, (err, doc) => {
        if (!err) {
            res.render("employee/addOrEdit", {
```

```
            viewTitle: "Update Employee",
            employee: doc
        });
    }
  });
});
```

ထို့ေနာက္ data မ်ားအား remove ျပဳလုပ္ရန္ findByidAndRemove ဆိုသည့္ function ကိုသံုးပါမည္ ။

```
router.get('/delete/:id', (req, res) => {
  Employee.findByIdAndRemove(req.params.id, (err, doc) => {
    if (!err) {
      res.redirect('/employee/list');
    }
    else { console.log('Error in employee delete :' + err); }
  });
});
```

ထို့ေနာက္ data မ်ားအား update ျပဳလုပ္ဂုန္ router.post ေနရာတြင္ program အနည္းငယ္ ပုိထည့္ပါမည္ ။ updateRecord ဆိုသည့္ function တစ္ခုကို တည္ေဆာက္ပါမည္ ။

```
//final
router.post('/', (req, res) => {
  if (req.body._id == '')
    insertRecord(req, res);
    else
    updateRecord(req, res);
});
```

updateRecord function ထဲတြင္ findOneAndUpdate ဆိုသည့္ function ကို ေခၚသံုးရပါမည္

```
function updateRecord(req, res) {
  Employee.findOneAndUpdate({ _id: req.body._id }, req.body, { new: true }, (err, doc)
=> {
    if (!err) { res.redirect('employee/list'); }
    else {
      if (err.name == 'ValidationError') {
        handleValidationError(err, req.body);
```

```
            res.render("employee/addOrEdit", {
                viewTitle: 'Update Employee',
                employee: req.body
            });
        }
        else
            console.log('Error during record update : ' + err);
    }
  });
}
```

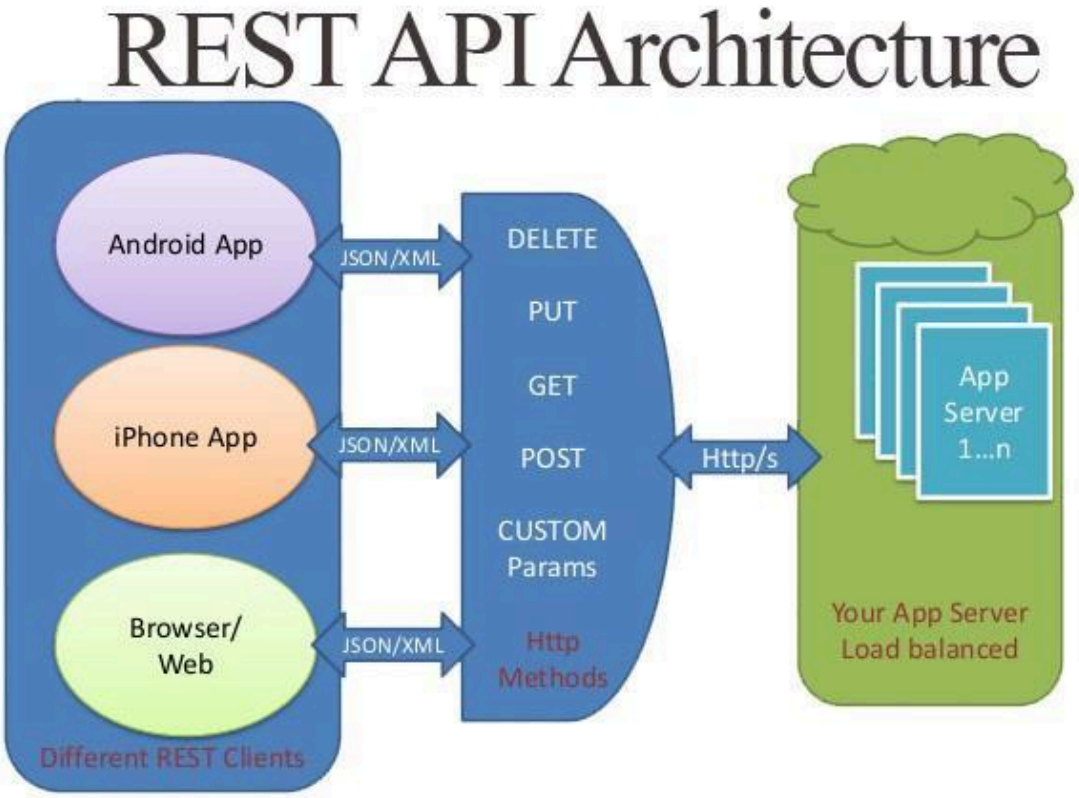
Program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္

```
const express = require('express');
var router = express.Router();
const mongoose = require('mongoose');
const Employee = mongoose.model('Employee');

router.get('/', (req, res) => {
    res.render("employee/addOrEdit", {
        viewTitle: "Insert Employee"
    });
});

router.post('/', (req, res) => {
    if (req.body._id == '')
        insertRecord(req, res);
        else
        updateRecord(req, res);
});


function insertRecord(req, res) {
    var employee = new Employee();
    employee.fullName = req.body.fullName;
    employee.email = req.body.email;
    employee.mobile = req.body.mobile;
    employee.city = req.body.city;
    employee.save((err, doc) => {
        if (!err)
            res.redirect('employee/list');
```

```
        else {
            if (err.name == 'ValidationError') {
                handleValidationError(err, req.body);
                res.render("employee/addOrEdit", {
                    viewTitle: "Insert Employee",
                    employee: req.body
                });
            }
            else
                console.log('Error during record insertion : ' + err);
        }
    });
}

function updateRecord(req, res) {
    Employee.findOneAndUpdate({ _id: req.body._id }, req.body, { new: true }, (err, doc)
=> {
        if (!err) { res.redirect('employee/list'); }
        else {
            if (err.name == 'ValidationError') {
                handleValidationError(err, req.body);
                res.render("employee/addOrEdit", {
                    viewTitle: 'Update Employee',
                    employee: req.body
                });
            }
            else
                console.log('Error during record update : ' + err);
        }
    });
}


router.get('/list', (req, res) => {
    Employee.find((err, docs) => {
        if (!err) {
            res.render("employee/list", {
                list: docs
            });
        }
        else {
```

```
            console.log('Error in retrieving employee list :' + err);
        }
    });
});


function handleValidationError(err, body) {
    for (field in err.errors) {
        switch (err.errors[field].path) {
            case 'fullName':
                body['fullNameError'] = err.errors[field].message;
                break;
            case 'email':
                body['emailError'] = err.errors[field].message;
                break;
            default:
                break;
        }
    }
}

router.get('/:id', (req, res) => {
    Employee.findById(req.params.id, (err, doc) => {
        if (!err) {
            res.render("employee/addOrEdit", {
                viewTitle: "Update Employee",
                employee: doc
            });
        }
    });
});

router.get('/delete/:id', (req, res) => {
    Employee.findByIdAndRemove(req.params.id, (err, doc) => {
        if (!err) {
            res.redirect('/employee/list');
        }
        else { console.log('Error in employee delete :' + err); }
    });
});
```

```
module.exports = router;
```

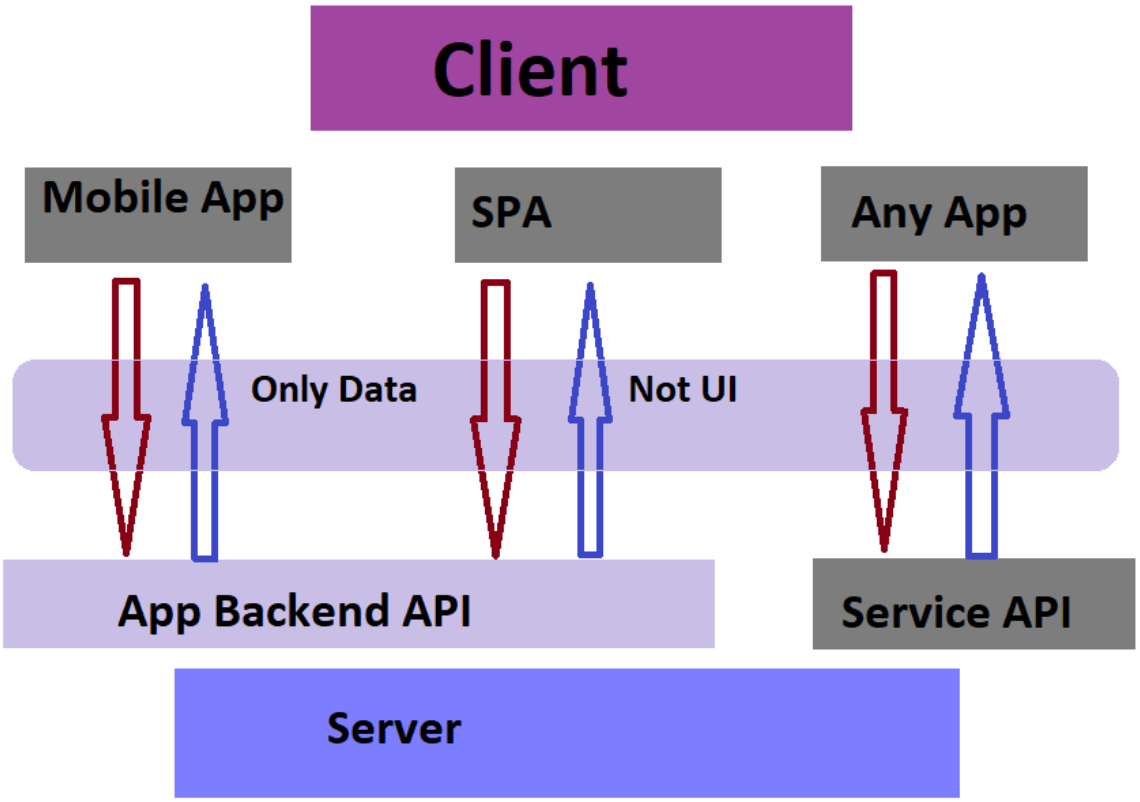


REST API

REST အရှည်မှာ Representational State Transfer ျဖစ္ျပီး User Interface အစား data မ်ား Transfer လုပ္ေပးျခင္းကို ဆိုလိုျခင္း ျဖစ္သည္။ Data ဟာ object ျဖစ္ေနလွ်င္ ထို object ရဲ႕ state အား ပို႔ေပးျခင္းျဖစ္သည္ ။HTML စသည့္ user interfaces မ်ား အစား client or fronted ဆီသို႔ data မ်ား Transfer လုပ္ျခင္းျဖစ္ပါတယ္။

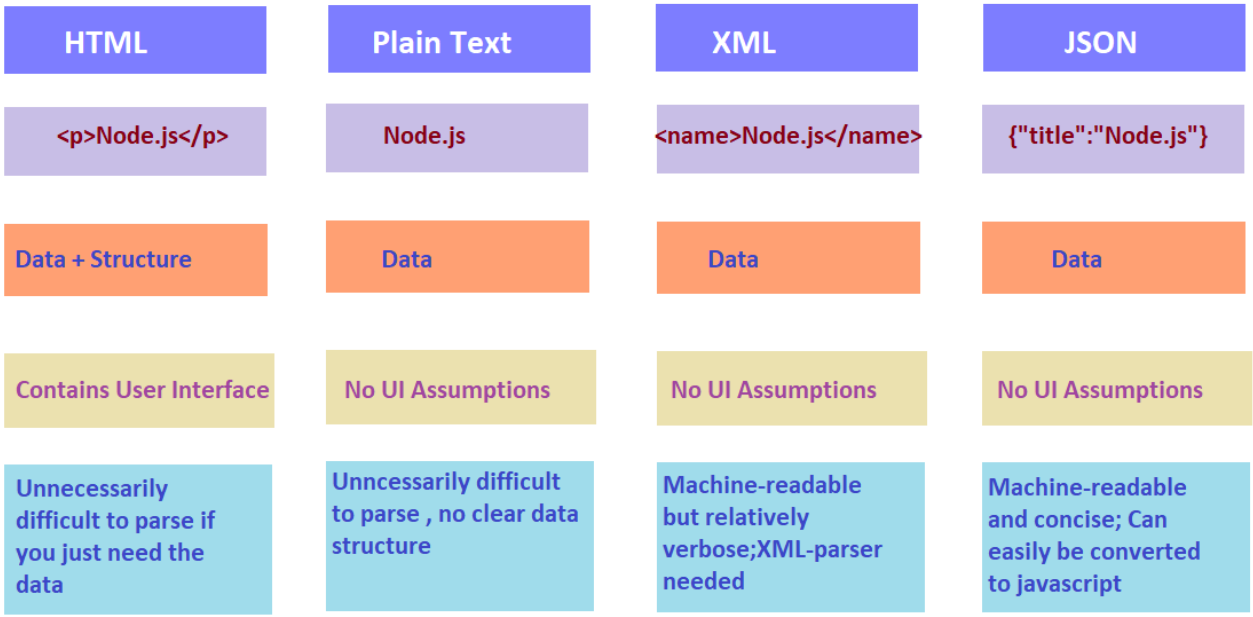
API သည္ application programming interface ကုိဆိုလိုျခင္းျဖစ္ျပီး ရွင္းလင္းစြာ ေျပာရလွ်င္ components မ်ား တစ္ခုနွင့္တစ္ခုျကား ဆက္သြယ္ရမည့္ နည္းလမ္း မ်ားကို ဆိုလိုျခင္းျဖစ္ပါသည္။ components ဆိုရာဝယ္ web app မ်ား mobile app မ်ား ကိုဆိုလိုျခင္း ျဖစ္ပါတယ္။ API ေတြဟာ **web-based system**

အတြက္လည္း ျဖစ္နိုင္သလို operating system, database system, computer hardware, or software library ေတြ အတြက္ လည္းျဖစ္နုိင္ပါတယ္။



Exchange Data Formats

Data ေတြကို ဘယ္လို Format မိုးနဲ့ exchange လုပ္ျကသလဲ ။ လူသိမ်ားတဲ့ formats ေတြကေတာ့ HTML , Plain Text , XML , JSON တို့ျဖစ္ျကပါတယ္ ။ HTML သည္ data exchange လုပ္ရာတြင္ data လည္းပါဝင္သလို သူရဲ့ html structure လည္းပါဝင္ပါတယ္။ ထိုေျကာင့္ HTML ျဖင့္ data exchange လုပ္ရာတြင္ UI လည္းပါဝင္ေနပါတယ္ ။ Plain text သည္ HTML ကဲ့သို့ structure မပါဝင္ပါဘူး data ခ်ည္းပဲ exchange ျပဳလုပ္နိုင္ပါတယ္ သူ့မွာ design element မ်ား မပါဝင္ပါဘူး။ သို့ေသာ္ plain text သည္ လူသားမ်ား ဖတ္ရွုနိုင္ လြယ္ကူေသာ္လည္း computer အတြက္ေတာ့ မလြယ္ကူပါဘူး။ XML သည္ HTML ျဖင့္ ဆင္တူပါေသာ္လည္း XML data format တြင္မူ UI မပါဝင္ပါ ။ ေနာက္ဆံုး တစ္ခုကေတာ့ JSON (JavaScript Object Notation) ပါ သူကေတာ့ data ခ်ည္းပဲ ပါဝင္ပါတယ္ မည္သည့္ UI မွ် မပါဝင္ပါ Machine ကေန လည္း အလြယ္ကူဆံုး ဖတ္နိုင္တဲ့ data format ျဖစ္ပါတယ္ ။ JavaScript ကိုလည္း အလြယ္တကူ ေျပာင္းလဲ နိုင္တဲ့ အတြက္ nodejs နဲ့ အတူ အလူလူရာမွာ အရမ္းကို အားသာပါတယ္။



Communication Between Server and Client (Theory Part)

Server and Client ျကားမွာ data transfer ျပဳလုပ္ဖုိ့နွ္ HTTP methods ကုိ အသံုးျပဳပါတယ္။ HTPP methods သည္ TCP/IP protocols ေပၚမွာ အေျခခံထားတာ ျဖစ္ျပီး HTTP ဆုိတာ Hypertext Transport Protocol ကုိ ဆုိလုိျခင္း ျဖစ္ပါတယ္ rest မွာဆုိရင္ေတာ့ Verbs လုိ့ ေခၚဆုိပါတယ္။ ဘာေျကာင့္လဲဆုိေတာ့ သူတို့ရ့ဲ actions ေတြျကာင့္ပါ ။ example အေနျဖင့္ CRUD (create , read , update , delete) စတဲ့ operation ေတြ ျပဳလုပ္ဖုိ့ GET – POST – PUT – DELETE တုိ့ ျဖစ္ပါတယ္။ အျခားေသာ methods ေတြရွိေသးေသာ္လည္း အထက္ပါ methods ေလးခုကုိသာ အထူးသျဖင့္ အသံုးျပဳျကပါတယ္။

GET method ကုိ server ကေန data မ်ား ျပန္ရယူနုိင္ရန္ အသံုးျပဳျကပါတယ္ ။ PUT and POST ကုိေတာ့ server ေပၚသုိ့ data မ်ား create and update လုပ္ဖုိ့ သံုးပါတယ္ ။ POST ကုိေတာ့ resourecs မ်ား create လုပ္ဖုိ့ သံုးျကသလုိ ရွိျပီးသား resources ေတြကုိ ထပ္ျပီး append လုပ္ရန္လည္း သံုးပါတယ္။ PUT ကုိေတာ့ create လုပ္ရန္ သုိ့မဟုတ္ update or over write လုပ္ရန္လည္း သံုးပါတယ္။ resoures or data မ်ား delete လုပ္ရန္ အတြက္ေတာ့ DELETE method ကုိ အသံုးျပဳပါတယ္။

Creating Our Own API

Folder တစ္ခု တည္ေဆာက္ပါ ထုိ folder ထဲတြင္ npm init လုပ္ပါ ထုိ့ေနာက္ express and nodemon ကုိ install လုပ္ပါ ။ ထုိ့ေနာက္ app.js ဆုိသည့္ js file တစ္ခု တည္ေဆာက္ပါ ။ ထုိ့ေနာက္ body-parser ကုိလည္း install လုပ္ထားပါ။ app.js ထဲတြင္ server တစ္ခု စတင္ရန္ ေအာက္ပါ code မ်ားကုိ ေရးပါ။

```
const express=require('express');
```

```
const app=express();

app.listen(8080,(err)=>{
    console.log('Server is on 8080');
})
```

ထို့ေနာက္ route လုပ္ရန္ routes ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါ ။ ထို folder ထဲတြင္ feed.js ဆိုသည့္ js file တစ္ခု တစ္ေဆာက္ပါ ။ express ကို သံုးမွာ ျဖစ္သည့္ အတြက္ express ကို ေျကျငာေပးပါ ။ Router() function ကိုသံုးမွာ ျဖစ္သည့္အတြက္ express.Router() ကိုလည္း ေျကျငာေပးရပါမည္။

```
const express=require('express')
const router = express.Router();
module.exports=router;
```

router ကိုလည္း export လုပ္ ေပးထားရပါမည္။

ထို့ေနာက္ controllers ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါမည္ ။ ထို folder ထဲတြင္ feed.js ဆိုသည့္ file တစ္ခု ထပ္ေဆာက္ပါမည္။ ယခု ေဆာက္ေသာ file သည္ controllers ထဲမွ file ျဖစ္သည္ကို သတိျပဳပါ။ controller ထဲမွ feed.js ဆိုသည့္ file ထဲတြင္ ေအာက္ပါ အတိုင္းေရပါမည္ ။

```
exports.getPosts=(req,res,next)=>{
    res.status(200).json({
        posts:[{ title:'first post' , content:'This is the first post!'}]
    })
};
```

getPosts ဆိုသည့္ function တစ္ခုကို တည္ေဆာက္ လိုက္ပါသည္ ထို function ထဲတြင္ req,res,next စသည့္ arguments သံုးခု ထည့္ေပးထားျပီး req လုပ္ပါက json data မ်ားကို respon လုပ္ ေပးပါမည္။ status(200) သည္ respon လုပ္ျခင္းကို ေအာင္ျမင္ေျကာင္း indicate လုပ္ျပျခင္း ျဖစ္သည္။ ယခု ေရးသားလိုက္ေသာ feed.js ကို routes folder ထဲမွ feed.js ထဲမွာ ေခၚသံုးရန္ routes ထဲမွ feed.js ထဲတြင္ ေအာက္ပါ အတိုင္းေရးေပးရပါမည္။

```
const feedController=require('../controllers/feed')
```

feedController ဆိုသည့္ object တစ္ခုကို တည္ေဆာက္လုိက္ျပီး ထို object ထဲမွ getPosts ဆိုသည့္ function ကို ျပန္ေခၚသံုးရန္ ေအာက္ပါ အတိုင္းေရးေပးရပါမည္။

```
router.get('/post',feedController.getPosts)
```

ထို့ေျကာင့္ routes ထဲမွ feed.js code အားလံုးမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္။

```
const express=require('express')

const feedController=require('../controllers/feed')
```

```
const router = express.Router();
router.get('/post',feedController.getPosts)
module.exports=router;
```

ထို႕ေနာက္ routes ထဲမွ feed.js အား app.js ဆိုသည့္ file မွ ျပန္ေခၚသံုးပါမည္။ ထို႕သို႕ ျပန္ေခၚသံုးဂ့ုန္

```
const feedRoutes=require('./routes/feed')
```

feedRoutes ဆိုသည့္ object တစ္ခု ကို တည္ေဆာက္လုိက္ပါသည္။ ထို႕ေနာက္

```
app.use('/feed',feedRoutes)
```

အထက္ပါ အတိုင္း ဆက္ေရးရမည္။ app.js ထဲရွိ code အားလံုးမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const express=require('express');
const app=express();
const feedRoutes=require('./routes/feed')

app.use('/feed',feedRoutes)

app.listen(8080,(err)=>{
   console.log('Server is on 8080');
})
```

url တြင္ http://localhost:8080/feed/post အတိုင္းသြားျကည့္ပါ ။ json format ျဖင့္ data မ်ားကို ျမင္ေတြ႕ရပါမည္ ။

{"posts":[{"title":"first post","content":"This is the first post!"}]}

ထို႕ေနာက္ controller ထဲက feed.js ထဲတြင္ function တစ္ခု တည္ေဆာက္ပါမည္ ထို function သည္ request လုပ္လာေသာ data မ်ားကို res လုပ္ေပးရန္ျဖစ္သည္ ။

```
exports.createPost=(req,res,next)=>{
   res.status(201).json({
      message:'Post created successfully',
      post:{id: new Date().toISOString(),title:title , content : content}
   })
};
```

Date သည္ date ေဖာျပ ေပးဂ့ုန္ ျဖစ္ျပီး toISOString သည္ string အျဖစ္ ေျပာင္းေပးဂ့ုန္ျဖစ္သည္ ။ title: titleသည္ body မွ ရလာေသာ title အား ျပန္လည္ ေဖာ္ျပေပးဂ့ုန္ ျဖစ္သည္ ။ content သည္ လည္း ထို္နည္း အတုိင္းပင္ျဖစ္သည္။ ထပ္ထည့္ ေသာ program အျပည့္အစံုမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
exports.createPost=(req,res,next)=>{
```

```
    const title=req.body.title;
    const content=req.body.content;
    res.status(201).json({
        message:'Post created successfully',
        post:{id: new Date().toISOString(),title:title , content : content}
    })
};
```

Status(201) သည် resources မ်ား update လုပ္ျခင္းကို ေအာင္ျမင္ေၾကာင္း indicate လုပ္ပုိ့ေပးသည္ ။ ထုိ့ေနာက္ app.js ထဲတြင္ body-parser ကို အသံုးျပဳရန္ လုိအပ္ပါသည္။ ထုိ့ေၾကာင့္ ေအာက္ပါ code နွစ္ေၾကာင္းအား ထပ္ထည့္ေပးပုိ့ရန္ လုိအပ္ပါသည္။bodyParser.json() သည္ json format ျဖင့္ body parse လုပ္ရန္ျဖစ္သည္။

```
const bodyParser=require('body-parser')
app.use(bodyParser.json())
```

routes folder ထဲမွ feed.js ထဲတြင္ creatPost function အတြက္ ေရးေပးရန္ လုိအပ္ပါေသးသည္ ။

```
router.post('/post',feedController.createPost)
```

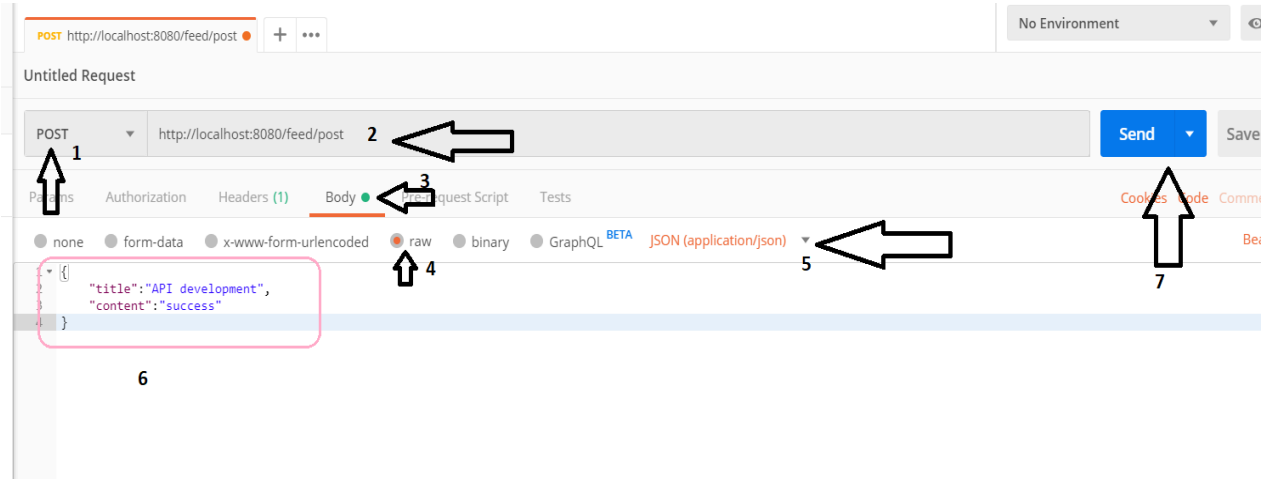
အထက္ပါ code line ထပ္ျဖည့္ေပးရပါမည္။

```
router.get('/posts',feedController.getPosts)
router.post('/post',feedController.createPost)
```

program အားလံုး အား save ျပီးရင္ run ထားပါ ။ယခု ေရးသားလိုက္ေသာ app အား စမ္းသပ္ပုိ့ရန္ postman ကို download ဆြဲရန္ လုိအပ္ပါသည္ ။

https://www.getpostman.com/

အထက္ပါ link အတိုင္းသြားျပီး download ဆြဲကာ install လုပ္ထားေပးပုိ့ရန္ လုိအပ္ပါသည္ ။

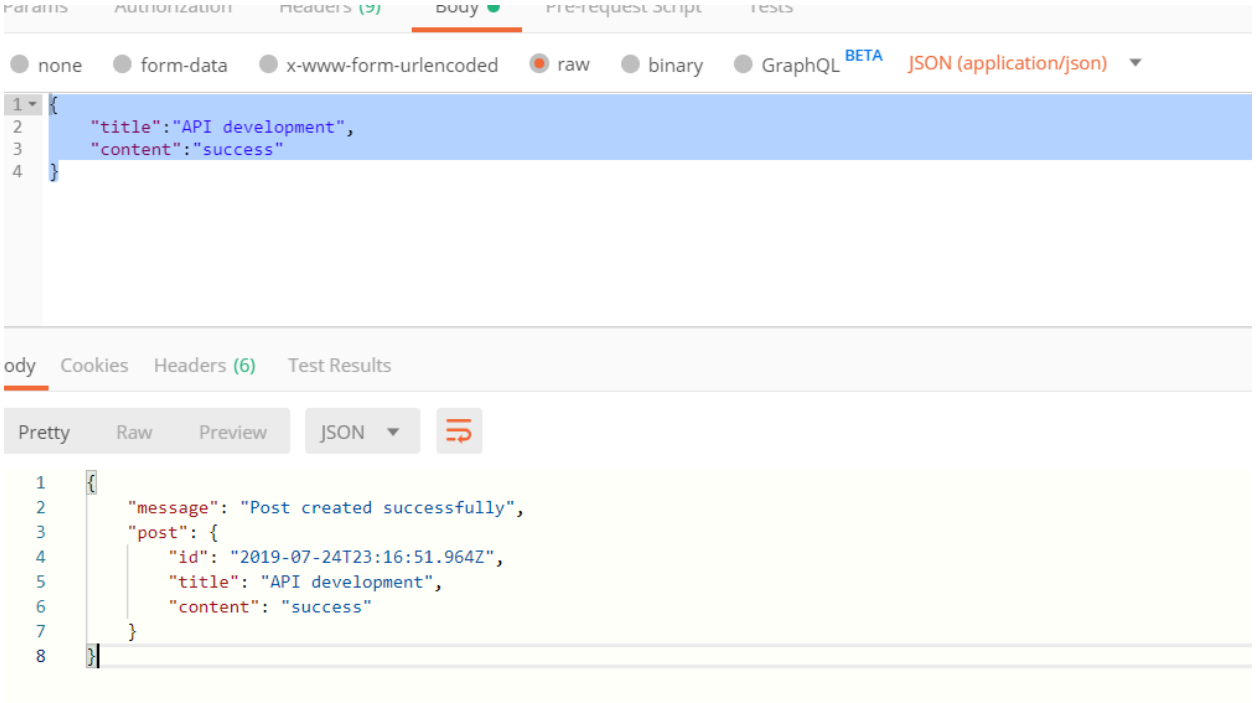


ပထမဆံုး အေနျဖင့္ method ေနရာတြင္ POST ကို ေရြးခ်ယ္ပါ ထို့ေနာက္ မိမိတို့ url ကို ထည့္ပါ ။ ျပီးလွ်င္ body request လုပ္ဟာ ျဖစ္သည့္ အတြက္ Body ကိုေရြးခ်ယ္ပါ

ျပီးလွ်င္ raw ကို ေဖြးပါ ထို႕ေနာက္ JSON(application/json) ကို ေရြးပါ။ ထို႕ေနာက္ ေအာက္ပါ json စာသားမ်ားကို ေရးပါ ျပီးလွ်င္ send ကို နွိပ္ပါ ။

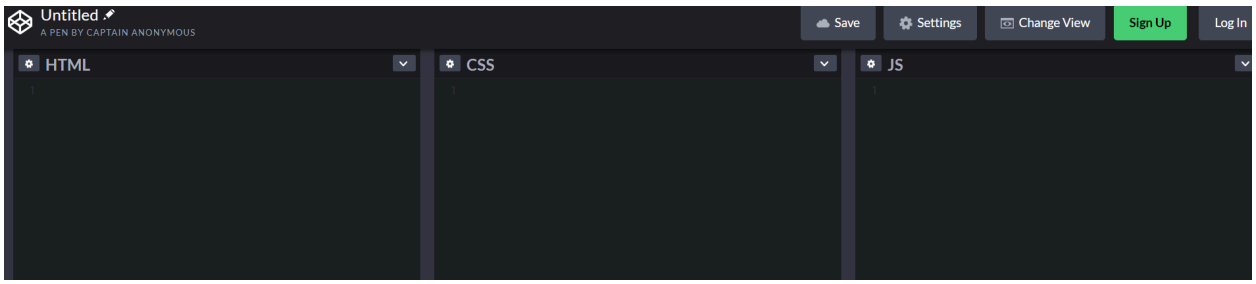
```
{
        "title": "API development",
        "content": "success"
}
```

step အားလံုး ေအာင္ျမင္ပါက ေအာက္ပါ အတိုင္း server မွ respon လုပ္သည္ကို ျမင္ရပါမည္ ။



REST APIs,Clients & CORS Errors

ပထမဆံုး အေနျဖင့ codepen.io ကို သြားပါမယ္ ထို site သို႕ ေရာက္သြားလွ်င္ starting coding ကိုေရြးခ်ယ္ပါ ။ တစ္ခ်က္ သတိျပဳရန္မွာ စာေရးသူ အေနျဖင့ codepen ကို Mozilla firefox browser အသံုးျပဳထားပါတယ္။ chrome လည္း အသံုးျပဳနုိင္ပါသည္။



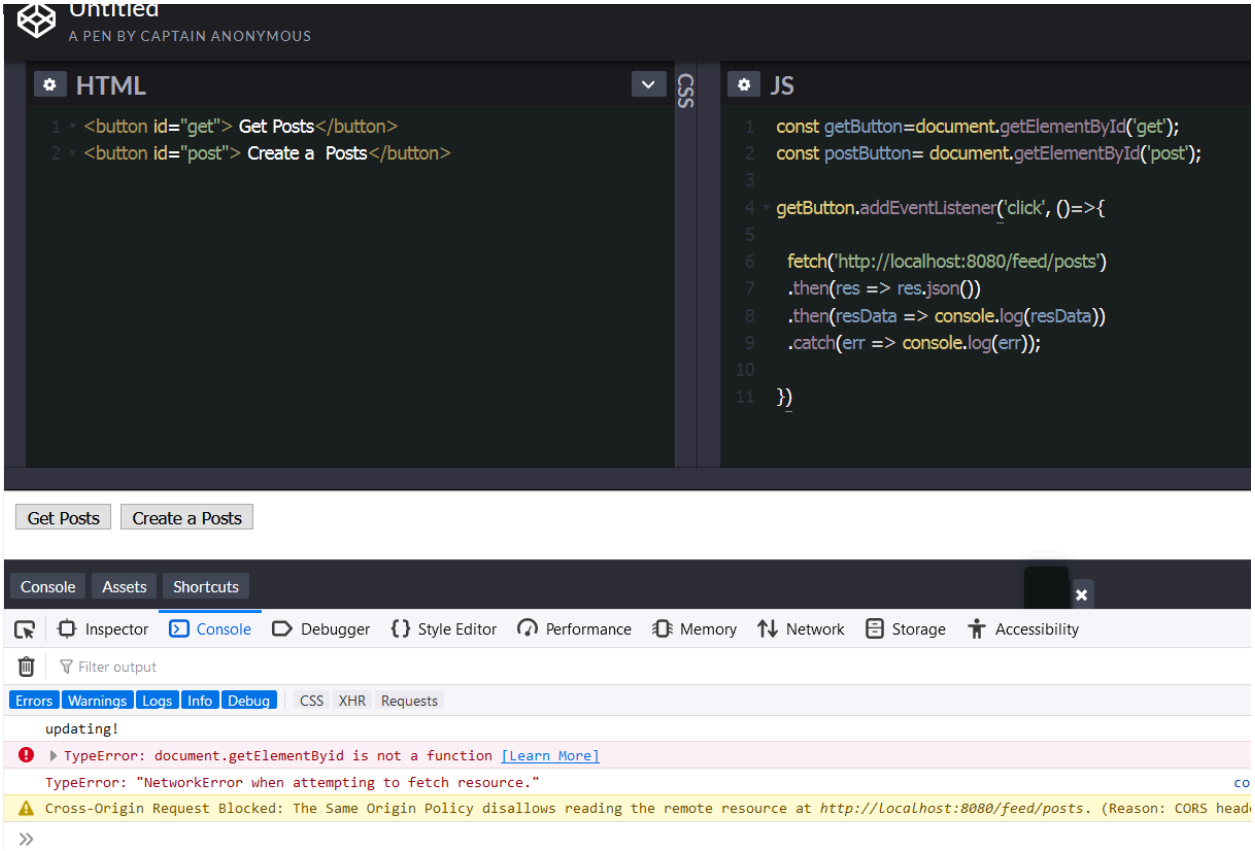
HTML နှင့် JS သာ သုံးမှာ ဖြစ်သည့် အတြက် CSS ကို ဖ်ာထွားလို့ ရပါတယ် ။ HTML page တြင် button နှစ်ခု တည်ဆောက်ပါမယ်။ codepen တြင် code မ်ားအားေရးပြီး save လုပ်စရာ မလုိပါဘူး automatic checking ပြ်ုလုပ်ေပးပါတယ် ။

```
<button id="get"> Get Posts</button>

<button id="post"> Create a  Posts</button>
```

ထိုနည်းတူ javascript အတြက်လည်း program ေရးသားပါမယ်။

```
const getButton=document.getElementById('get');

const postButton= document.getElementById('post');

getButton.addEventListener('click', ()=>{

  fetch('http://localhost:8080/feed/posts')

  .then(res => res.json())

  .then(resData => console.log(resData))

  .catch(err => console.log(err));

})
```

JavaScript ကေန မွ တစ်ဆင့် network requests ပြ်ုလုပ်ပြီတဲ့ အခါမီးေတြမှာ fetch method ကိုသုံးပါတယ် ။ ယခု သင်ခန်းစာမွာေတာ့ fetch method ထဲမွာ မိမိ စမ်းလိုေသာ localhost address ကို အတိအက် ထည့်ေပးရပါမယ်။ code မ်ားအားလံုး ေရးပြီးေသာ အခါ မိမိတို့ browser မွ developers tool ကို ဖွင့ပါ ပြီးလ်ွင် get button ကို တစ်ခ်က်နှိပ်ပြကည့ပါ ေအာကပါ အတိုင်း error တက်သည်ကို ြမင်ရပါမည် ။



အဝါေရာင္ စာတန္းျဖင့္ အထက္ပါ error ကို CORS (Cross – Orgin Resource Sharing) error လို႕ေခၚပါတယ္။ modern web application မ်ားနွင့္ single page web application ေတြမွာ ေတြ႕ရတတ္ပါတယ္ ။

Adding setHeader

အထက္ပါ error မ်ားကို ေျဖရွင္းရန္ Origin ဆိုသည့္ request မ်ားကို စီစစ္ျပီးလက္ခံရန္ ဘယ္ methods မ်ားကို accept လုပ္မည္ျဖစ္ေၾကာင္း နွင့္ Authorization မ်ားကို server side တြင္ ထည့္ေပး ရပါမည္ ။ ထို႕ေၾကာင့္ app.js ထဲတြင္ ေအာက္ပါအတိုင္း ထည့္ေပးပါမည္ ။

```
res.setHeader('Access-Control-Allow-Origin','*')
```

ဘယ္ ေနရာကေန မဆို access လုပ္ခြင့္ရန္ * ကို ထည့္ေပးထားရပါမည္ ။ သို႕မဟုတ္ မိမိ accept လုပ္လုိေသာ address ကိုလည္း သီးသန္႕ ထည့္ေပးနိုင္ပါသည္။

```
res.setHeader('Access-Control-Allow-Methods','GET,POST,PUT,PATCH,DELETE');
```

မိမိ လက္ခံလိုေသာ methods မ်ားကိုလည္း လိုသလို ထည့္နုိင္သည္။

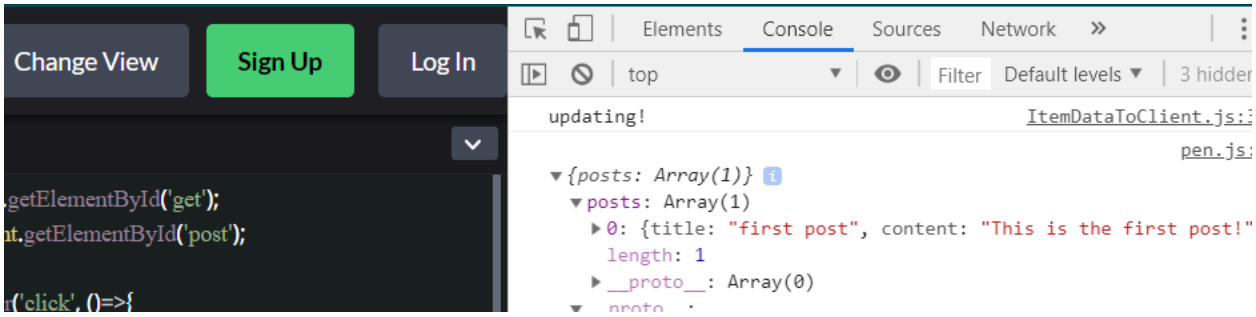
```
res.setHeader('Access-Control-Allow-Headers','Content-Type,Authorization')
```

Authorized ျဖစ္ရန္အတြက္လည္း ထည့္ေပးနုိင္ပါသည္ ။

အထက္ပါ အတိုင္း code မ်ား ထည့္ျပီးေသာ္လည္း error တက္ေနပါက cors ကို သံုးေပးရပါမည္ ။ npm i cors ကို install လုပ္ပါ ထို႔ေနာက္ const cors= require('cors') ကို ေရးပါ။ ျပီးလွ်င္ app.use(cors()) ကို ေရးေပးရပါမည္။၏

```
const cors=require('cors')
app.use(cors());
```

အထက္ပါ အတိုင္း app.js ထဲတြင္ထပ္ထည့္ျပီးပါက server ကို ျပန္run ပါ ထို႔ေနာက္ codepen ကို ျပန္သြားျပီး getpost button ကို ျပန္ run ျကည့္ပါ ။ ျပီးလွ်င္ developer tools ကို ဖြင့္ျကည့္ပါက error မတက္ေတာ့ ေျကာင္းကို ျမင္ေတြ႔နိုင္ပါသည္ ။



Posts ဆိုသည့္ စာသားကို ျမင္ရမည္ျဖစ္ျပီး သူ႔ေဘးမွ ျမွားေလးကို နွိပ္ျကည့္ပါက ပံုပါ အတိုင္း resources မ်ားကို ျမင္ရပါမည္ ။

အထက္ပါ program source code မ်ားကို ေအာက္ပါ link တြင္ download ဆြဲနိုင္ပါသည္ ။

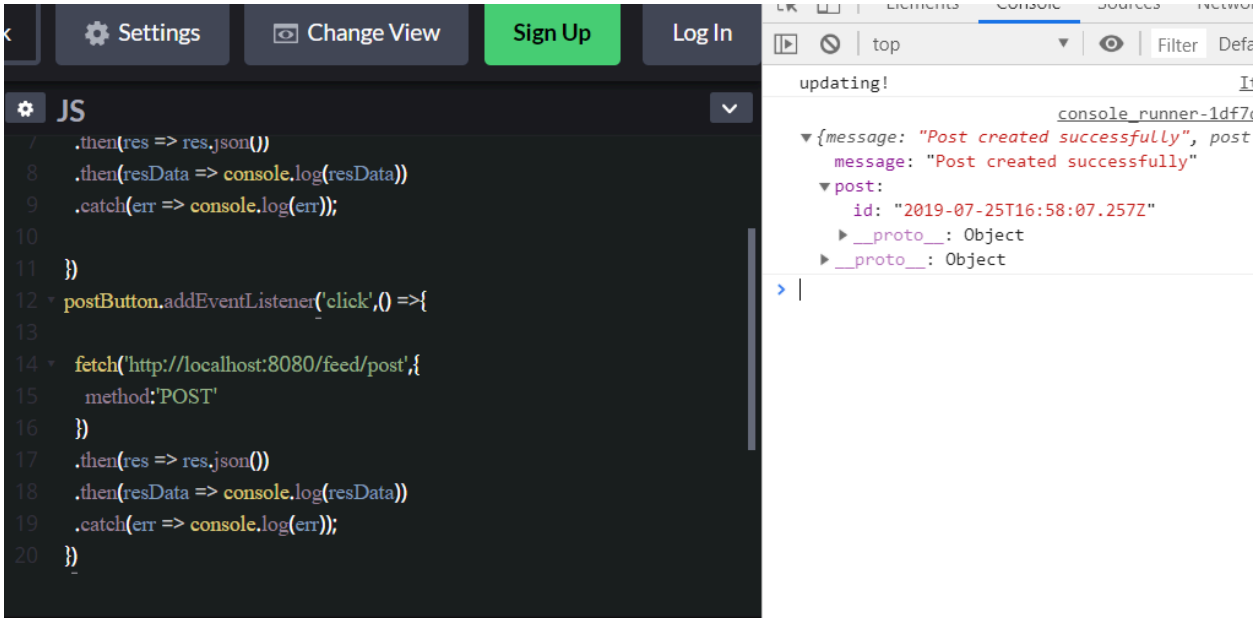
https://my.pcloud.com/publink/show?code=XZa6Jv7ZmgmYQ89nm9BwvYt3dY62m57et8Vy

Sending POST request

POST request ျပဳလုပ္ရန္ codepen ထဲမွ javascript ထဲတြင္ ေအာက္ပါ code မ်ားကို ဆက္ေရးပါ ။

```
postButton.addEventListener('click',() =>{
  fetch('http://localhost:8080/feed/post',{
    method:'POST'
  })
  .then(res => res.json())
  .then(resData => console.log(resData))
  .catch(err => console.log(err));
})
```

အထက္ပါ အတိုင္းေရးျပီး Creat a Post button ကို နွိပ္ျကည့္ပါ ေအာက္ပါ ပံုအတုိင္း post created succefully ဆိုတဲ့ စာေျကာင္းကို ျမင္ရပါမည္။



Getတြင် method မထည့္ေပးလိုက္ေသာ္လည္း post တြင္မူ method ထည့္ေပးလုိက္သည္ကို သတိျပဳပါ။ ထို႔ေနာက္ javascript ထဲတြင္ post လုပ္ရန္ body တစ္ခု တည္ေဆာက္ပါမည္ ။

```
method:'POST',
  body:{
    title:'A codeopen post',
    content:'Created by Win Htut'
  }
```

အထက္ပါ code မ်ားအား javascript ထဲတြင္ ထပ္ေဂ်ာျပီး Creat a post button ကို နွိပ္ျကည့္ပါ မိမိတို႕ ထည့္ေပးလိုက္ေသာ စာသားမ်ား ျပန္လည္ ေဖာ္ျပ ေပးျခင္း မရွိေသးသည္ကို ျမင္ရပါမည္။ထို႕သို႕ ျမင္ရ ဂုန္ controllers ထဲမွ feed.js ထဲတြင္ ျပင္ေပးဂုန္ လိုအပ္ပါေသးသည္။

```
const title=req.body.title;
const content=req.body.content;
console.log(title,content)
```

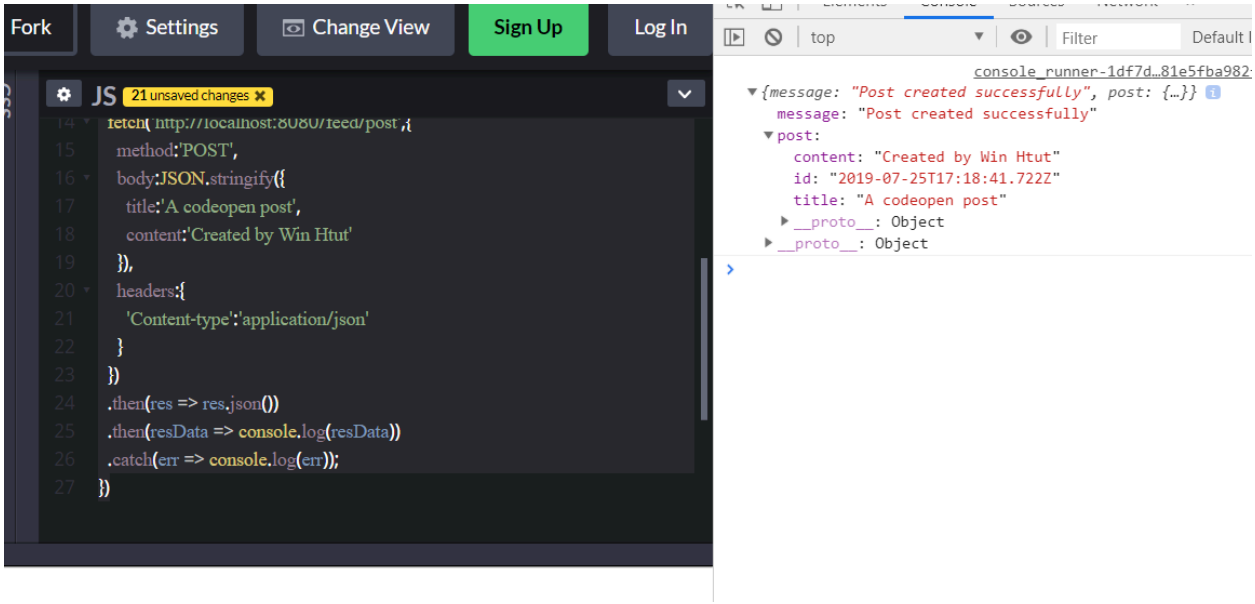
console.log(title,content) ဆိုသည့္ code တစ္ေျကာင္းကို ထပ္ထည့္ပါ resource မ်ား respon ဘာေျကာင့္ မျပန္လဲ ဆိုတာကို debug လုပ္ရန္ ျဖစ္သည္ ။ အထက္ပါ အတိုင္း ထည့္ေဂ်ာျပီး program ကို save ျပီး run ပါ ထို႔ေနာက္ codepen တြင္ creat a post button ကို နွိပ္ျကည့္ပါ ။ ျပီးလွ်င္ sever side console တြင္ျကည့္ပါ undefined ဆိုသည့္ စာသားမ်ားကို ျမင္ရပါမည္ ။

အထက္ပါ error မ်ားကို ေျဖရွင္းဂုန္ body ကို JSON ပံုစံမ်ိဳးျဖင့္ ေရးသားေပးရမည္ျဖစ္ျပီး headers ကိုလည္း Content-type မွာ application/json

ဆိုျပီး ထည့္ေပးဂ့ပါမည္ ။ codepen javascript ထဲတြင္ ထပ္ျဖည့္မည့္ program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္။

```
postButton.addEventListener('click',() =>{
  fetch('http://localhost:8080/feed/post',{
    method:'POST',
    body:JSON.stringify({
      title:'A codeopen post',
      content:'Created by Win Htut'
    }),
    headers:{
      'Content-type':'application/json'
    }
  })
  .then(res => res.json())
  .then(resData => console.log(resData))
  .catch(err => console.log(err));
})
```

အထက္ပါ အတိုင္း ျဖည့္ေရးျပီး Creat a post button အား နွိပ္ျကည့္ပါ ထို့ေနာက္ developer tools ကို ဖြင့္ပါ Console တြင္ ေအာက္ပါ အတုိင္း မိမိတို့ ထည့္ေပးလုိက္ေသာ စာသားမ်ားကို ျမင္ရပါမည္။



Developing RESTFUL APIs

ယခု အခန္းတြင္ေတာ့ RESTFUL APIs တစ္ခုကုိ တည္ေဆာက္သြားပါမည္ ။ ပထမဆံုး အေနျဖင့္ မိမိ နွစ္သက္ရာ folder တစ္ခုကို ဖြင့္ျပီး app.js ဆိုသည့္ file တစ္ခု တည္ေဆာက္ပါ ။ထို႔ေနာက္ express and nodemon ကို install ျပဳလုပ္ထားပါ ။ ထို႔ေနာက္ routes နွစ္ခု ေရးပါမည္ ပထမ တစ္ခုသည္ root route ျဖစ္ျပီး ဒုတိယ တစ္ခုသည္ /api/courses ျဖစ္သည္ ။ program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const express=require('express')
const app=express();

app.get('/',(req,res)=>{
   res.send('Hello World');
})
app.get('/api/courses',(req,res)=>{
   res.send([1,2,3]);
})

const port=process.env.PORT || 3000;
app.listen(port,()=>{
   console.log(`A journey is on port ${port}... `)
});
```

ယခု သင္ခန္းစာတြင္ process.env.PORT || 3000 ကို သံုးထားပါသည္ ။ process object ထဲမွ environment property ထဲမွ PORT ကို သံုးထားျခင္းျဖစ္ပါသည္ ။ထိုသို႕ သံုးထာျခင္းျဖင့္ port number ကို မိမိ လိုသလို ေျပာင္းလဲ အသံုးျပဳနုိင္ပါသည္ ။

terminal တြင် set PORT=5000 ဟုေရးပါက port number 5000 ကို ခိန္ုးေပးပါမည္။ ေအာက္ပါ အတိုင္းစမ္း ျကည့္ရုိင္ပါသည္ ။ mac တြင္ေတာ့ set ေနရာမွာ export ကို သံုးပါသည္ ။

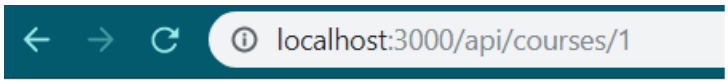
```
C:\Users\MAT\Documents\nodeprogram\RESTFUL_APIs>set PORT=5000

C:\Users\MAT\Documents\nodeprogram\RESTFUL_APIs>nodemon app.js
[nodemon] 1.19.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
A journey is on port 5000...
```

အဆင့္အားလံုးျပီးပါက browser တြင္ မိမိ ေဂၚသားထားေသာ routes မ်ားကို စမ္းျကည့္ပါ ။

အထက္ပါ အဆင့္မ်ား ေအာင္ျမင္ပါက ေနာက္ထပ္ route တစ္ခု ထပ္ တည္ေဆာက္ပါမည္ ။

```
app.get('/api/courses/:id',(req,res)=>{
    res.send(req.params.id);
})
```

ယခု route တြင္ :id သည္ parameter ကို ဆိုလိုျခင္းျဖစ္သည္ ။ client ဘက္မွ /api/courses/ေနာက္မွာ ထည့္ေပးလုိက္ေသာ parameter ကို : (colon) ေနာက္က name တစ္ခုထဲသို႕ ထည့္ေပးလိုက္မည္ ျဖစ္ပါသည္ ။ client ဘက္ မွ ထည့္ေပးလိုက္ေသာ parameter ကို ျပန္ဖတ္ရန္ req.params.(client ထည့္ေသာ parameter) ကို သံုးပါသည္ ။ id ေနရာတြင္ မိမိ ျကိုက္ခ်ိန္သက္ ေသာ နံမည္ကို သံုးနုိင္ပါသည္ ။ တစ္ခုျခင္းစီ ဖတ္ယူေသာ အခ်ိန္မ်ားတြင္မူ : ေနာက္မွ ထည့္ေပးလိုက္ေသာ name အတိုင္း params ေနာက္တြင္လည္း ထည့္ေပးရပါမည္ ။ အထက္ပါ program အား app.js ထဲတြင္ ထည့္ေဂၚျပီး ေအာက္ပါ url အတိုင္း စမ္းသပ္ျကည့္ရုိင္ပါသည္ ။

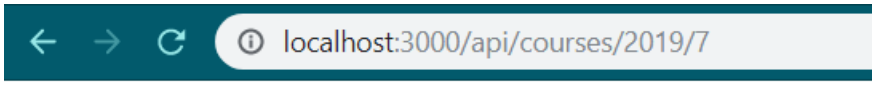


1

Params ဆိုတဲ့ Object ကိုသံုးျပီး req မ်ားကို ဖတ္ပါမည္ ။ program ကို ေအာက္ပါ အတိုင္း အနည္းငယ္ ျပင္ေဂၚပါ ။

```
app.get('/api/courses/:year/:month',(req,res)=>{
    res.send(req.params);
})
```

ထို႕ေနာက္ url တြင္ ေအာက္ပါ ပံု အတိုင္း စမ္းျကည့္ရုိင္ပါသည္ ။



{"year":"2019","month":"7"}

အထက္ပါ အဆင့္မ်ား ေအာင္ျမင္ပါ program ထဲတြင္ course ဆိုသည့္ array တစ္ခု တည္ေဆာက္ပါ ။ ထို course ဆိုသည့္ array ထဲတြင္ id and name ကို ေပးထားပါမည္ ။

```
const courses=[
   {id:1,name:'course1'},
   {id:2,name:'course2'},
   {id:3,name:'course3'},
];
```

ျပီးလွ်င္ မိမိ တို႕ ေနာက္ဆံုး ေရးခဲ့ေသာ route တြင္ year and month မ်ားကို ျဖတ္ျပီး ေအာက္ အတိုင္ း ေဂၚပါမည္ ။

```
app.get('/api/courses/:id',(req,res)=>{
   const course=courses.find(c => c.id === parseInt(req.params.id));
   if(!course)res.status(404).send('The course with the given id was not found');
   res.send(course);
})
```

Const course = courses.find (c => c.id === praseInt (req.params.id)) ဆိုသည့္ code line အား ရွင္းလင္းပါမည္။ (req.params.id) သည္ client ဘက္မွ ထည့္ေပးလိုက္ေသာ parameter မ်ား အား ဖတ္ရန္ ျဖစ္သည္ ။ praseInt သည္ user ထည့္ေပးလိုက္ေသာ string စာေျကာင္းအား integer အေနျဖင့္ေျပာင္းလဲ ေပးရန္ျဖစ္သည္ ။ courses.find သည္ courses array အား find ဆိုသည့္ method ကို သံုး၍ျပီး ရွားျခင္း ျဖစ္ပါသည္ ။ c => c.id သည္ id ကို ရွာျခင္း ျဖစ္သည္ ။ c ေနရာတြင္ မိမိ ျကိုက္ရာစက္ေသာ variable name မ်ားကို သံုးနိုင္သည္ ။ ထို code line ရဲ့ အျပည့္အစံု ဆိုလိုရင္းမွာ client ဘက္မွ ထည့္ေပးလိုက္ေသာ id ကို ယူမယ္ integer ပံုစံေျပာင္းမယ္ ထို႕ေနာက္ courses ဆိုတဲ့ array ထဲက id နဲ့ တိုက္စစ္မယ္ တူရင္ res.send(course) ဆိုျပီး ျပန္ေပးမယ္ မတူရင္ status 404 နဲ့ error ျပန္ျပ ေပးမွာ ျဖစ္ပါတယ္ ။ ေအာက္ေဖာ္ျပပါ ပံု အတိုင္း စမ္းသပ္ၾကည့္ပါတယ္ ။

{"id":1,"name":"course1"}

မရွိတဲ့ id number ဥပမာ 10 ဆိုျပီး ထည့္ျကည့္ပါက error log ကို ျပန္ျပ ေပးမွာ ျဖစ္ပါတယ္ ။ အထက္ပါ အဆင့္အားလံုး ျပီးသည့္ အခ်ိန္တြင္ app.js ထဲတြင္ ရွိေသာ program အားလံုးမွာ ေအာက္ပါ အတိုင္းျဖစ္ပါသည္ ။

```
const express=require('express')
const app=express();

const courses=[
   {id:1,name:'course1'},
   {id:2,name:'course2'},
   {id:3,name:'course3'},
];
app.get('/',(req,res)=>{
   res.send('Hello World');
})
app.get('/api/courses',(req,res)=>{
   res.send([1,2,3]);
})

app.get('/api/courses/:id',(req,res)=>{
   const course=courses.find(c => c.id === parseInt(req.params.id));
   if(!course)res.status(404).send('The course with the given id was not found');
   res.send(course);
})

const port=process.env.PORT || 3000;
app.listen(port,()=>{
   console.log(`A journey is on port ${port}... `)
});
```

POST

ယခု သင္ခန္းစာတြင္ post လုပ္ရန္ အတြက္ route တစ္ခု ေရးပါမည္ ။

```
app.post('/api/course',(req,res)=>{
   const course={
```

```
    id: courses.length+1,
    name: req.body.name
  };
  courses.push(course);
  res.send(course);
})
```

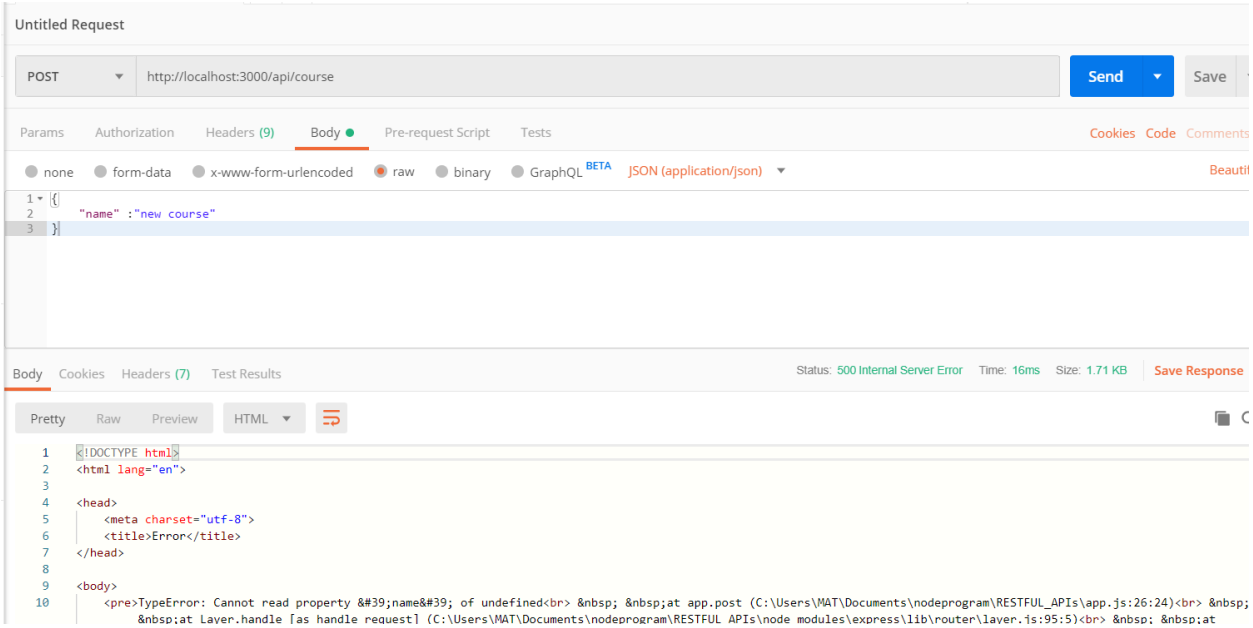
ထို route ထဲတြင် course ဆိုသည့် object တစ်ခု တည်ေဆာက်ပါမည့် ထို object သည့် id number ကို ထပ်ိုးရန်နှင့် client ဘက်မှ server သို့ data မ်ား ထပ့ျပီး create လုပ်လွှင့် ထို client ဘက်မှ post လိုက့ေသာ data ကို ရယူဂုန် req.body.name ကို ေရးေပးရျခင်း ျဖစ်သည့် ။ id ကို ထပ်ိုးေပးရန် အတြက် courses ဆိုတဲ့ array ရဲ့ length ကို +1 ဟိုေရးက 1 ထပ့ ေပါင်းေပးလိုက်ျခင်း ျဖင့် id number ၁ တိုးသြားမည့် ျဖစ်သည့် ။ ထိုဂ့ေနာက် တြင် push ဆိုသည့် method ကို အသံုးျပ်ျပီး course ဆိုသည့် new data object ကို courses ဆိုသည့် array ထဲသို့ ထပ္ထည့် ေပးဂုန် courses.push(course) ဆိုသည့် code line ကို အသံုးျပ်ရျခင်းျဖစ်ပါသည့် ။ ထို့ေနာက်တြင် client ဘက်သို့ course ကို ျပန်လည့် ျပသ ေပးဂုန် အတြက် res.send(course) ဆိုသည့် code line ကို အသံုးျပ်ရျခင်းျဖစ်သည့် ။ အထက်ပါ အဆင့်မ်ားျပီးပါက postman ကို ဖြင့်ပါ ထို့ေနာက် url တြင် မိမိတို့ ယခု ေရးသားလိုက်သည့် route ကို အတိအက် ထည့်ေပးပါ ထို့ေနာက် POST ကို ခိန်းပါ ျပီးလွှင့် body and JSON application ကို ခိန်းပါ ။ ျပီးေနာက်

{

“name” : “new course “

}

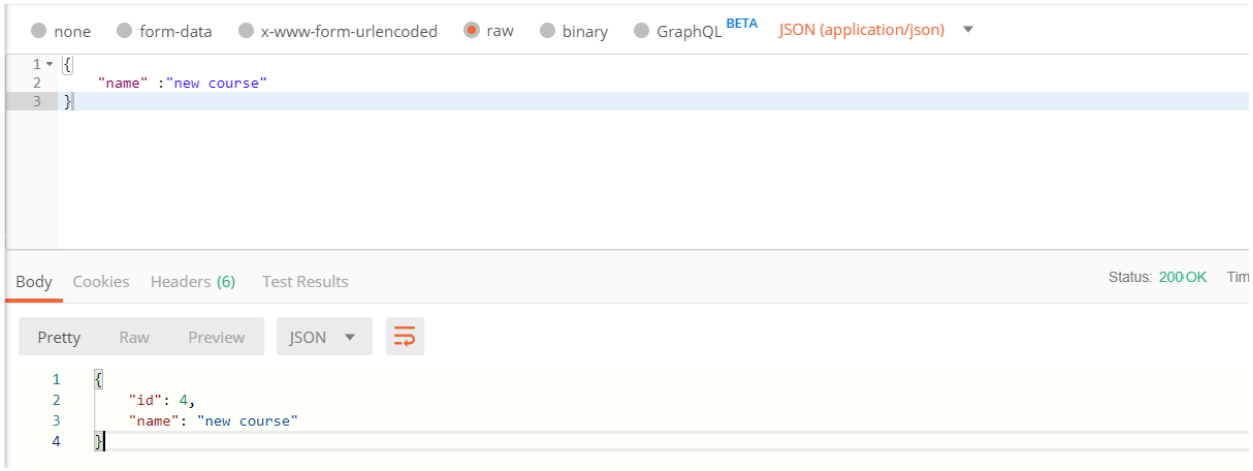
အထက်ပါ အတိုင်းေရးျပီး send button ကို နှိပ်ပါ ။ ျပီးလွှင့် error တက်သည့်လို့ ျမင်ရပါမည့် ။ အဘယ့် ေျကာင့်ဆို့ေသာ့ json ကို သံုးထားေသာ်လည့်း server ဘက်တြင် json ကို အသံုးျပ်မည့် ျဖစ့် ေျကာင်း မေျကျငာ ေပးထားေသာေျကာင့်ျဖစ်သည့် ။



ထို error ကို ေျဖရွင္းရန္ app.js ဆိုသည့္ server ဘက္တြင္ ေအာက္ပါ code line ကို ျဖည့္ေပးရပါမည္ ။

```
app.use(express.json())
```

အထက္ပါ အတိုင္း ျဖည့္ေရးျပီးေနာက္ program အား save ျပီး ျပန္ run ပါ ျပီးလွ်င္ postman မွ တစ္ဆင့္ post ကို သံုးျကည့္ပါ ။ error မ်ား မတက္ေတာ့ပဲ server ဘက္မွ course ကို respon ျပန္လာ သည္ကို ျမင္ရပါမည္ ။ id မွာလည္း တစ္ခု တိုးသြားသည္ကို ျမင္ရပါမည္ ။



Courses အားလံုးကို ျပန္လည္ ျကည့္လိုပါက ယခင္က ေအာက္ပါ အတိုင္းေရးထားေသာ route တြင္ code အနည္းငယ္ျပင္ေဂ်းပါမည္ ။

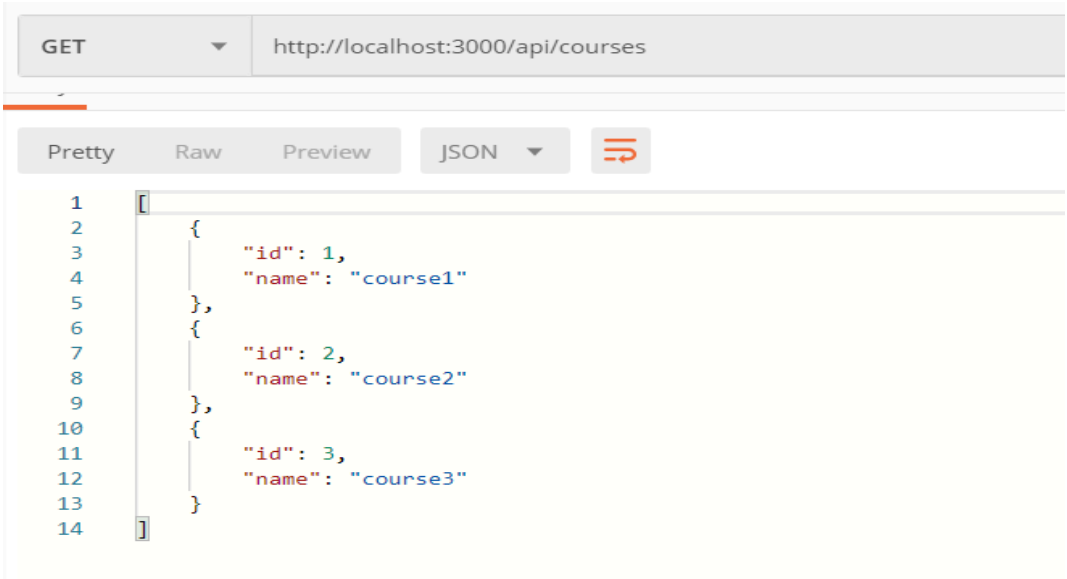
```
app.get('/api/courses',(req,res)=>{
```

```
    res.send([1,2,3]);
})
```

1 2 3 ေနရာတြင္ courses ဆိုသည့္ array ကိုေအာက္ ပါ အတိုင္းထည့္ေရး ေပးရပါမည္ ။

```
app.get('/api/courses',(req,res)=>{
    res.send(courses);
})
```

ထို႕ ေရးျပီးပါက postman url တြင္ /api/courses သို႕ သြားပါ ။ method ကို GET သို႕ ခ်ိန္းျပီး send ကို နွိပ္ပါ ေအာက္ပါ ပံုအတိုင္း courses မ်ား ျပန္ေပးလာသည္ကို ျမင္ရပါမည္ ။



Validation with Joi

Client ဘက္မွ normal user ေသာ္လည္း hackers ေသာ္လည္းေကာင္း ပံုစံ အမ်ိဳးမ်ိဳးနဲ႕ post လုပ္လာသမွ်ကို လက္ခံ ေပးလို႕ မရပါဘူး ထို ျပသနာ ကို ေျဖရွင္းရန္ အတြက္ post ျဖင့္ေရာက္လာေသာ data မ်ားကို validate (မွန္ကန္ေျကာင္း အတည္ျပဳေပးရပါမည္) ။ ထို႕ေျကာင့္ validate ဆိုသည့္ method ကို ေခၚသံုးရပါမည္ ။ ထို method ကို joi ျဖင့္ အသံုးျပဳပါမည္ ။ joi package ကို အသံုးျပဳဖို့ npm i joi ျပဳလုပ္ပါ ။ထို႕ေနာက္ ေအာက္ပါ တိုင္း ေျကျငာေပးပါ

```
const Joi=require('joi')
```

ထိုကဲ့သို႕ ေျကျငာျပီးပါက post method ျဖစ္သည့္ api/course ဆိုသည့္ route တြင္ ေအာက္ပါ code line တို႕ကို ျဖည့္စြက္ ေရးသားပါမည္ ။

```
const schema={
      name : Joi.string().min(3).required()
  };//schema
```

ပထမဆံုး အေနျဖင့္ schema ဆိုသည့္ object တစ္ခုကို တည္ေဆာက္ပါမည္ ထို schema သည္ shape ျဖစ္ပါသည္ ။ shape ဟု ဆိုလိုျခင္းမွာ မိမိလိုခ်င္ေသာ ပံုစံကို ဆိုလိုသည္ ။ ဥပမာ name သည္ string ျဖစ္ရမည္ ထို႔နည္းတူ name တြင္လည္း အနည္း ဆံုး စကားလံုး ၃ လံုး ပါရမည္ စသည့္ အခ်က္ မ်ား လိုအပ္ပါသည္ဆို၍သည့္ shape ျဖစ္သည္ ။ ထို အခ်က္မ်ား စစ္ေဆးရန္ validate method ကို သံုးပါမည္ ။

```
const result=Joi.validate(req.body,schema)
```

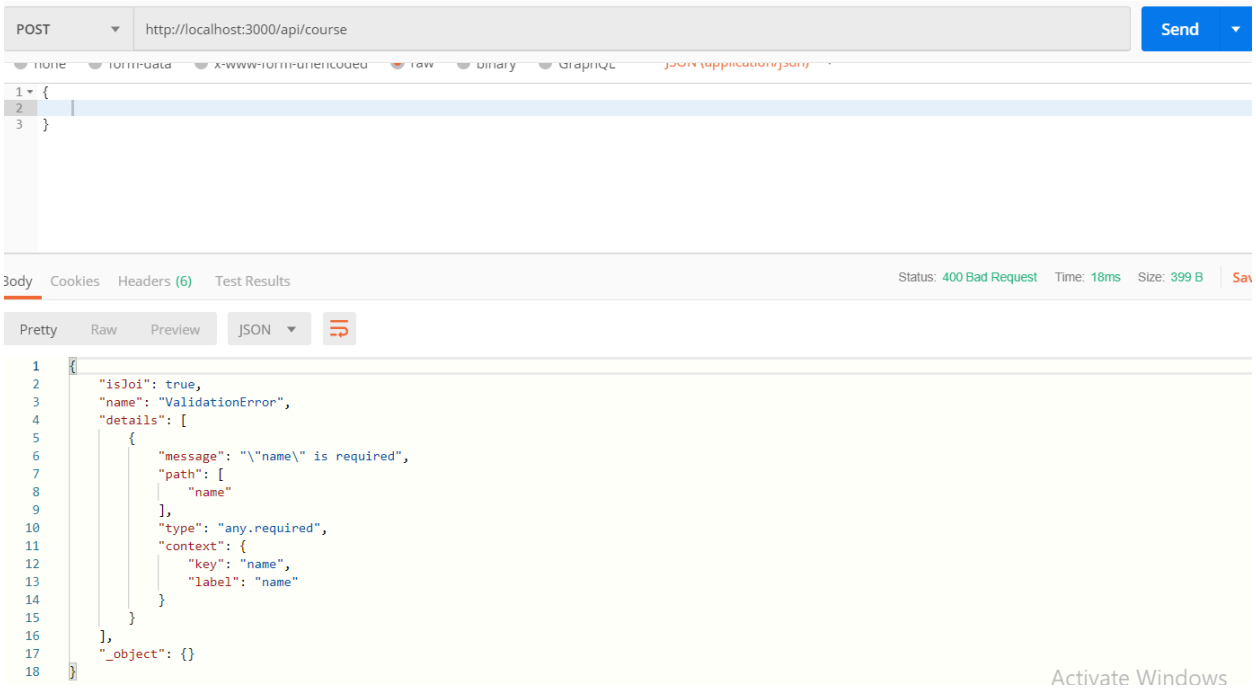
body မွ ရလာေသာ data သည္ shcema ထဲတြင္ သတ္မွတ္ထားေသာ shape အတိုင္း ျဖစ္သလား ဟု စစ္ေဆးရန္ အတြက္ validate method ေနာက္တြင္ parameter ႏွစ္ခု ထည့္ေပးထားျခင္းျဖစ္သည္ ။ validate method သည္ Joi ထဲမွ ျဖစ္ေသာေၾကာင့္ ေရွ႕တြင္ Joi ကိုေရးေပးထားျခင္းျဖစ္သည္ ။ ထို႔ေၾကာင့္ စစ္ေဆးျပီး ရလာေသာ ရလဒ္ကို result ထဲသို႔ ထည့္ပါမည္ ။ ထို႔သို႔ထည့္ျပီး result သည္ မိမိတို႔ လိုခ်င္ေသာ ပံုစံျဖင့္ မကိုက္ညီပ် error ျဖစ္ေနပါက error log ကို ျပန္ ျပ ေပးရန္ အတြက္ ေအာက္ပါ အတိုင္း status(400) ျဖင့္ ထပ္မံ ေရးပါမည္ ။

```
if(result.error){
      res.status(400).send(result.error);
      return;
   }
```

POST method တြင္ api/course url ေအာက္တြင္ ေရးထားေသာ code အားလံုးမွာ ေအာက္ပါ အတိုင္း ျဖစ္သည္ ။

```
app.post('/api/course',(req,res)=>{

   const schema={
      name : Joi.string().min(3).required()
   };//schema
   const result=Joi.validate(req.body,schema)
   if(result.error){
      res.status(400).send(result.error);
      return;
   }
   const course={
      id: courses.length+1,
      name: req.body.name
   };
   courses.push(course);
   res.send(course);
})
```

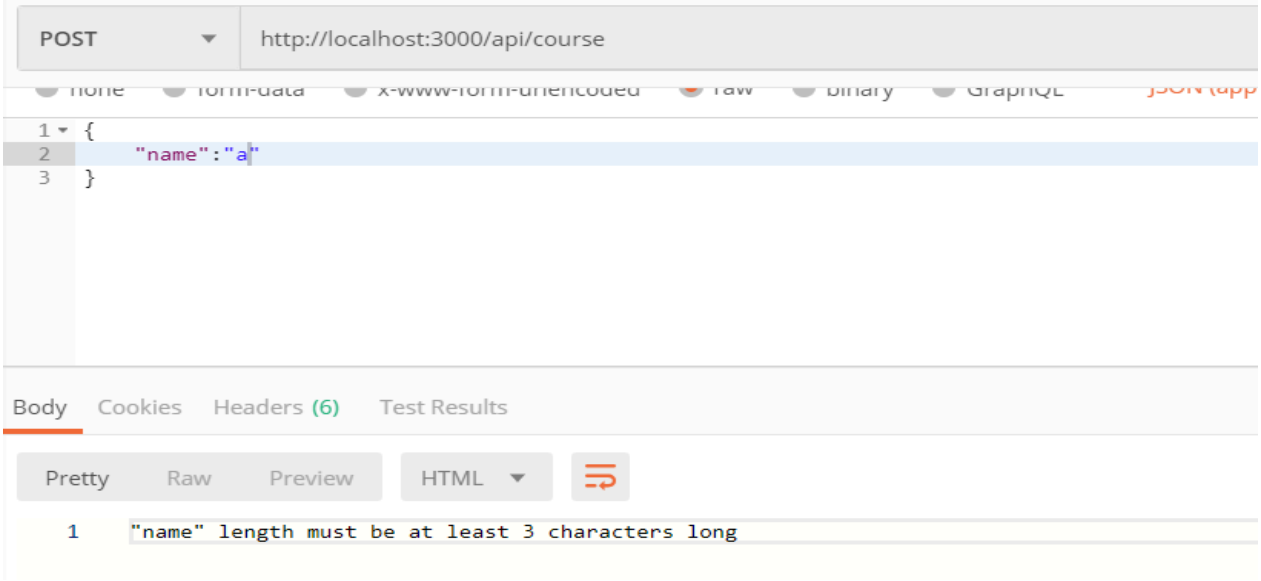
မိမိေရးထားေသာ code မ်ားအား စမ္းရန္ postman တြင္ POST method ကို သံုးျပီး body တြင္ ဘာမွ မေရးပဲ send ျကည့္ပါ ေအာက္ပါ အတိုင္း error log ကို ျမင္ရပါမည္ ။



joi သည္ input မ်ားကို စစ္ေဆးနိုင္ရံုသာမက သင့္တင့္ေသာ error message မ်ားကိုလည္း ျပန္ေပးနုိင္ပါတယ္ ။ ထို႕သို႕ error message မ်ားကို ျပန္ေပးနိုင္ရန္ details ဆိုသည့္ Object ထဲမွ message ကို ေအာက္ပါ အတိုင္းေခၚသံုးေပးရပါမည္ ။

```
res.status(400).send(result.error.details[0].message);
```

error message မ်ားကို ျပန္လည္စစ္ဆးရန္ body ကို မည္သည့္ စာသားမွ် မပါ ပဲ send လုပ္ျကည့္ပါ ရိုးရွင္းျပီးတိက်ေသာ error message ကိုသာ ျမင္ရပါမည္ ။ စကားလံုးကို 3 ထက္နည္းျပီး ပို႕ပါကလည္း သံုးလံုးထိ လိုအပ္ေျကာင္း တိက်ေသာ error message ကို ျပန္ျပ ေပးနိုင္ပါေသးသည္



PUT (data မ်ားကို update ျပဳလုပ္ခ်င္ရန္ put method ကို အသံုးျပဳပါမည္)

PUT method အတြက္ route ထဲတြင္ ေရးသားရမည့္ ပံုစံမွာ

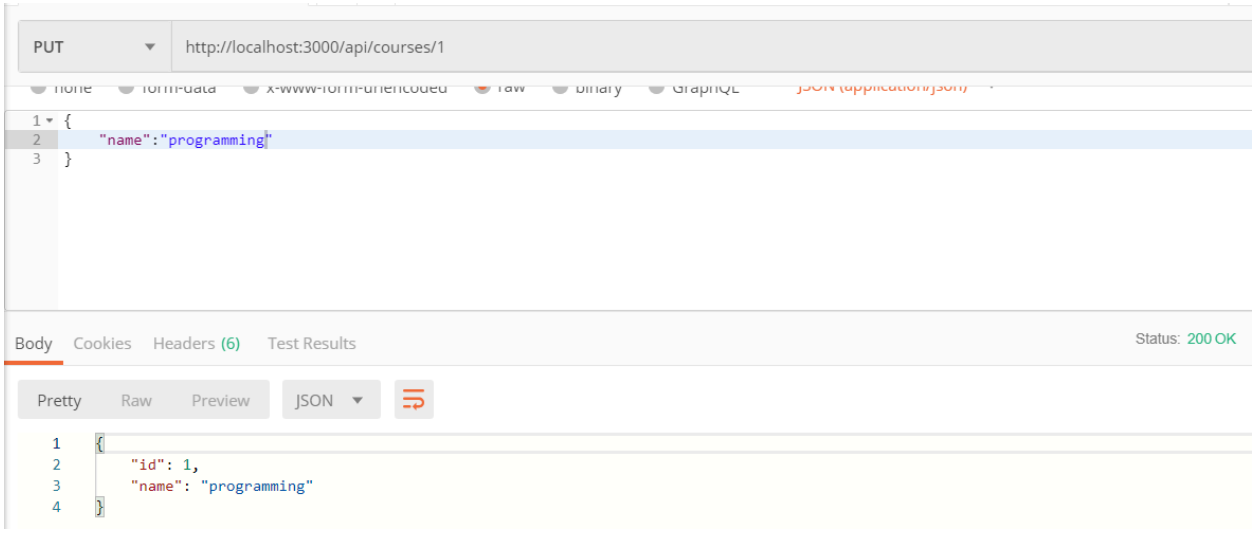
1. **Courses ေတြ အားလံုးကို ျကည့္ပါမယ္**
2. **တစ္ကယ္လို႕ courses ေတြ မရွိဘူးဆိုရင္ 404 resource not found ဆိုသည့္ status ကိုျပန္ ေပးပါမယ္**
3. **ထို႕ေနာက္ validate လည္းလုပ္ပါမည္ validation မေအာင္ျမင္ပါက 400 ဆိုသည့္ bad request status ကို ျပန္ေပးပါမည္ ။**
4. **အထက္ပါ အဆင့္မ်ား အားလံုးေအာင္ျမင္ပါက course ကို update လုပ္ပါမည္ ။**
5. **ထို႕ေနာက္ update လုပ္ထားေသာ courses မ်ားကို ျပန္ေပးပါမည္ ။**

အထူး သတိျပဳရန္မွာ courses ေတြအားလံုးျကည့္ျခင္း ၊ courses id မ်ားအားစစ္ျခင္း ၊ validation မ်ားသည္ အရင္ ေရးထားျပီးေသာ code မ်ားကို ျပန္လည္ အသံုးျပဳနုိင္ပါသည္ ။ ယခု program တြင္ update လုပ္ရန္ တစ္ေျကာင္းသာ ပိုေရးရပါမည္ ။ app.put ဆိုသည့္ route အသစ္အတြက္ code အားလံုးမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
app.put('/api/courses/:id',(req,res)=>{
   const course=courses.find(c => c.id === parseInt(req.params.id));
   if(!course)res.status(404).send('The course with the given id was not found'); //
finding course

   const schema={
      name : Joi.string().min(3).required()
   };//schema // validation course
   const result=Joi.validate(req.body,schema)
   if(result.error){
```

```
    res.status(400).send(result.error.details[0].message);
    return;
  }
      // just one code line to update
  course.name=req.body.name;// updating course
  res.send(course);

});
```

အထက္ပါ အတိုင္း program အား ျဖည့္စြက္ေရးသားျပီး postman တြင္ /api/courses/1 ကို ျပန္လည္ update ျပင္ဆင္ေရးသားရန္ ေအာက္ပါ ပံုအတိုင္းေရးသားျပီး send နွိပ္ပါ ။ method တြင္ put ကို အသံုးျပဳရန္ အေရးျကီးပါသည္ ။



Course1 ေနရာတြင္ programming ဟု ေျပာင္းလဲရွန္ အတြက္

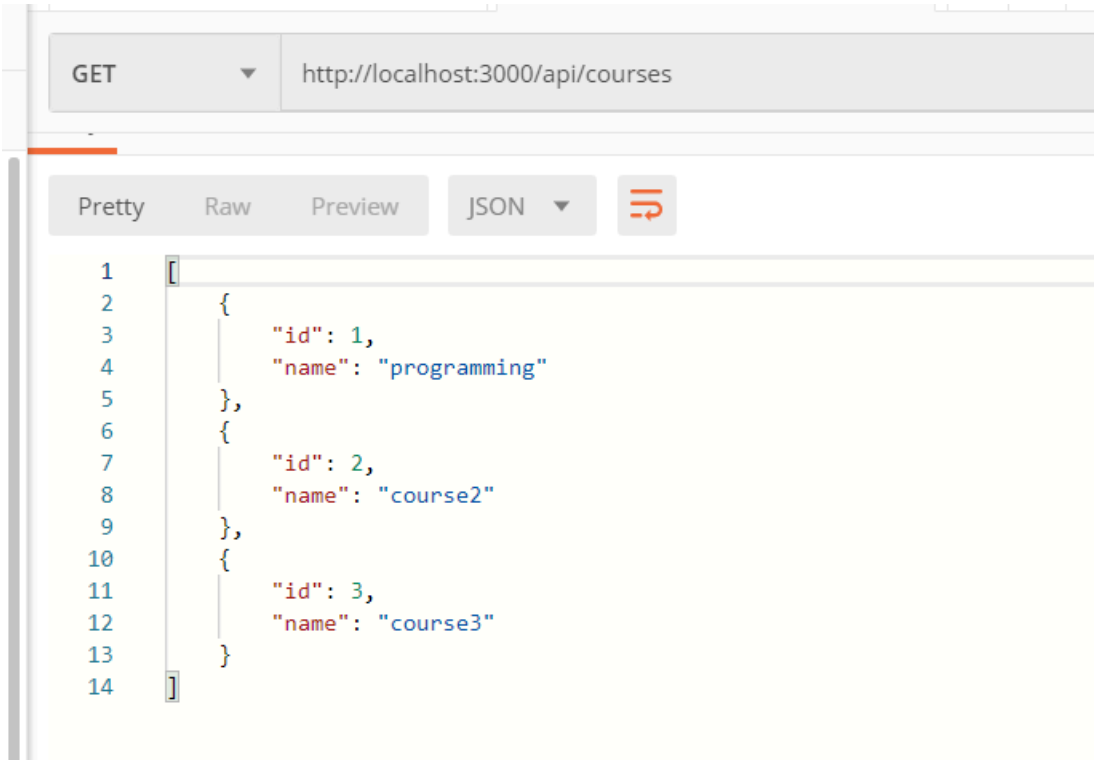
{

“name”:”programming”

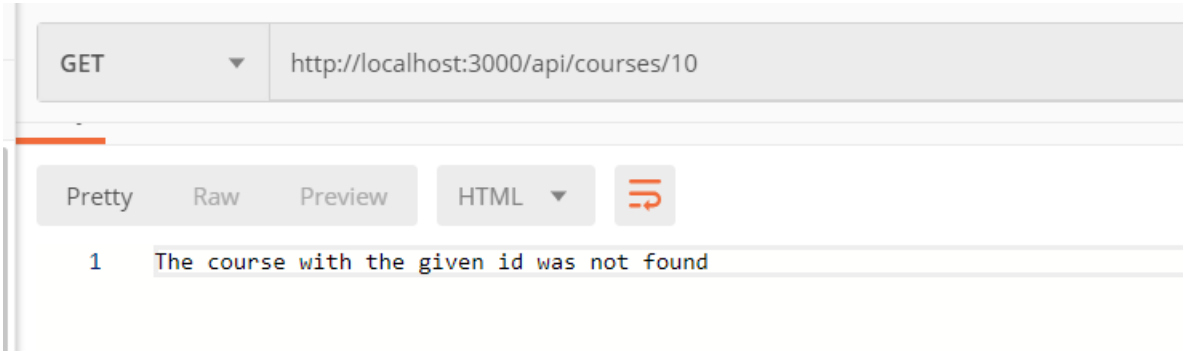
} ဟို body ထဲတြင္ ေရးသားျပီး put method ကို သံုးကာ send နွိပ္လိုက္ရင္ respon အေနျဖင့္ id 1 ေနရာ၌ programming အျဖစ္ ခိန္းသြားသည္ကို ျမင္ရပါမည္ ။

Checking resourses

Courses အားလံုး အား စစ္ေဆးရွန္ postman တြင္ new tab ကို ဖြင့္ျပီး GET method ကို ခိန္းပါ ထို့ေနာက္ http://localhost:3000/api/courses ယခု link အတိုင္း send နွိပ္ပါ ။ ေအာက္ပါ ပံုအတိုင္း resources အားလံုးကို ျမင္ရပါမည္ ။



ထို႕ေနာက္ url တြင္ မရွိေသာ course တစ္ခုကို request လုပ္ျကည့္ပါ ။ မရွိေျကာင္းကို ေအာက္ပါ ပံု အတုိင္း error message ျပန္လည္ ျပသ ေပးပါမည္ ။



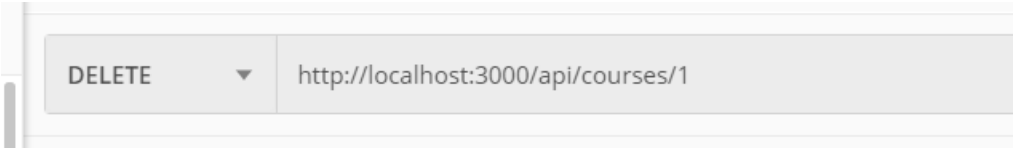
Delete method

Resources မ်ားကို delete ျပဳလုပ္ရန္ ပထမဆံုး client ဘက္မွ delete လုပ္လုိေသာ course id ကို ယူပါမည္ ။ ထို id ကို ယူျပီးေနာက္ ရွိလား မရွိလား စစ္ေဆးပါမည္ ။မရွိပါက error message ကို ျပ မည္ျဖစ္ျပီး ရွိပါက courses ထဲက resources ကို ဖ်က္ပစ္ပါမည္ ။ ထိုသို႕ ျပဳလုပ္ရန္ ပထမဆံုး app.delete ဆိုသည့္ route တစ္ခုကို ေရးပါမည္ ထို route ထဲတြင္ ေအာက္ပါ code မ်ားကို ေရးသားပါမည္

```
app.delete('/api/courses/:id',(req,res)=>{
  const course=courses.find(c => c.id === parseInt(req.params.id));
```

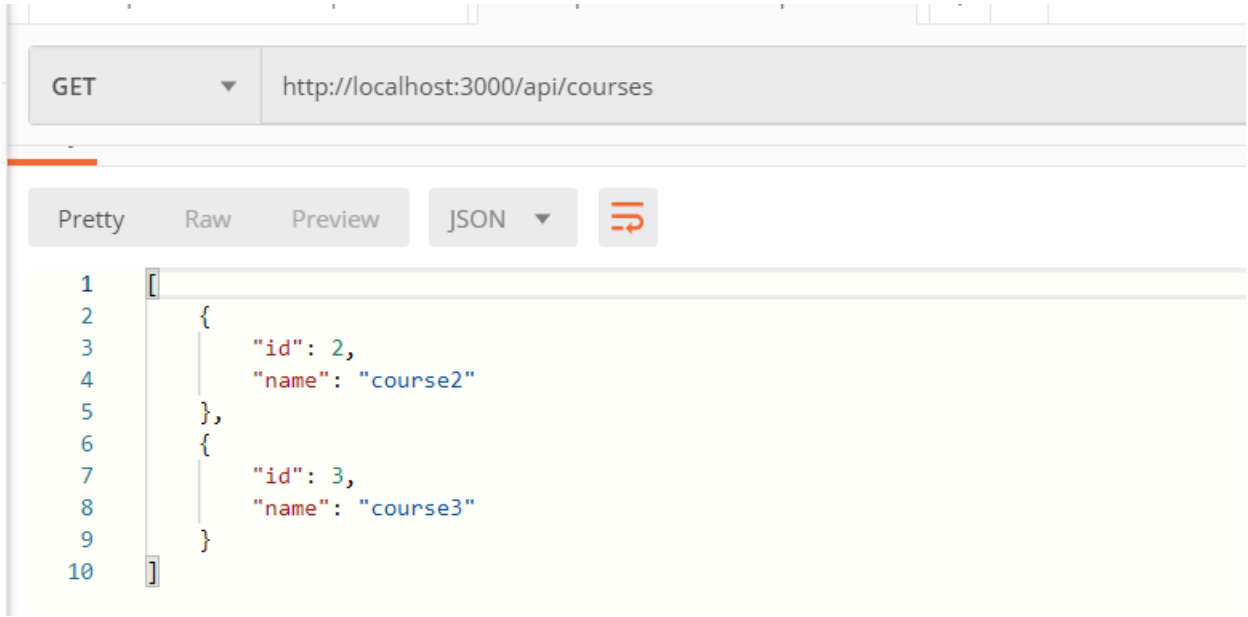
```
  if(!course)res.status(404).send('The course with the given id was not found');

  const index=courses.indexOf(course)
  courses.splice(index,1);

})
```

ဘယ္ေနရာက data ကို ဖ်က္ခ်ဖ္ခမည္ဆိုတာကို သိနိုင္ဖို့ indexOf ဆိုသည့္ method ကိုသံုးပါသည္ ထို method ကို သံုးျပီး ရလာေသာ data ကို index ထဲသို့ ထည့္ပါသည္။ ျပီးလွ်င္ splice ဆိုသည့္ method ကိုသံုးျပီး ဖ်က္ပါသည္ ထို method ကိုသံုးရာတြင္ ပထမ parameter သည္ ေနရာ ျဖစ္ျပီး ဒုတိယ parameter သည္ အေရအတြက္ျဖစ္ပါသည္။ courses array ထဲမွ resources မ်ားကို ဖ်က္မွာ ျဖစ္သည့္ အတြက္ ေရွ့ဆံုးတြင္ courses ဆိုသည့္ array ကို ထည့္ထားေပးျခင္းျဖစ္ပါသည္။



ဖ်က္ဂုန္ အတြက္ေရးသားျခင္းျဖစ္သည္ ။

Courses မ်ားအား ေအာက္ပါ အတိုင္း ျပန္လည္ စစ္ေဆးနိုင္သည္ ။



RESTFUL APIs with Mongodb (Lessons)

ပထမဆံုး အေနျဖင့္

STEP 1

1. Folder တစ်ခု တည်ေဆာက်ပါမည် ။
2. App.js ဆိုသည့် file တစ်ခု တည်ေဆာက်ပါမည် ။
3. Express and nodemon တို့ကို install လုပ်ထားပါမည် ။
4. ထို့ေနာက် server တစ်ခု တည်ေဆာက်ပါမည် ။
5. Express ကို သံုးမည်ျဖစ်ေျကာင်း ေျကျငာေပးထားပါမည်။
6. url ဆိုသည့် object တစ်ခု တည်ေဆာက်ပါမည် ။ ထို url တြင် မိမိ ခိတ်လို့ေသာ database link ကိုခိတ်ဆက်ပါမည် ။
7. ျပီးလ်ွင် mongodb ကို connect လုပ်ပါမည် ။
8. Connection success ျဖစ်ြင် Connected to database ... ဆိုသည့် စာေျကာင်းကို ေဖာ်ျပေပးမည် ျဖစ်ျပီး ။ connection success မျဖစ်ြင် error log ကို ျပေပးပါမည် ။
9. Express url မ်ားကို urlencoded လုပ်ပါမည် ။
10. Json() ကို သံုးမည်ျဖစ်ေျကာင်းလည်း ေျကျငာေပးထားပါမည် ။

အထက်ပါ အဆင့်မ်ား ေဆာင်ရွက် ျပီးသည့် ေနာက်တြင် program မွာ ေအာက်ပါ အတိုင်း ျဖစ်သည် ။

```
const express=require('express')
const app=express();
const
url='mongodb+srv://winhtut:.....@loginwinhtut-e6gu8.mongodb.net/test?retryWrites=true&w=majority'

// Database
mongoose.connect(url, { useNewUrlParser: true })
  .then(() => console.log('Connected to database...'))
  .catch(err => console.error(err));

// Middleware
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

var port=process.env.PORT || 3000;
app.listen(port,(req,res)=>{
console.log(`A journey on ${port}.....`);
})
```

STEP 2

1. models ဆိုသည့် folder တစ်ခု တည်ေဆာက်ပါမည်။
2. models ထတြင် userModels.js ဆိုသည့် js file တစ်ခု တည်ေဆာက်ပါမည် ။
3. mongoose ကို သံုးမည်ျဖစ်ေျကာင်း ေျကျငာေပးမည် ။

4. mongoose.Schema ကို သံုးမည့္ျဖစ္ေၾကာင္း ေၾကျငာေပးမည္ ။
5. object တစ္ခု တည္ေဆာက္ပါမည္ ။ name ကို UserSchema ဟုနံမည္ ေပးပါမည္ ။
6. UserSchema object ထဲတြင္ forename , surname , email , password ,age , team စသည့္ data မ်ား ထည့္ပါမည္ ။ forename type ကို String type ျဖင့္ ထားပါမည္ ။ surname ကိုလည္း string type ျဖင့္ထားပါမည္ ။ email , password , team ကို လည္း string type မ်ားျဖင့္ ထားပါမည္ ။ age ကိုေတာ့ number type ျဖင့္ထားပါမည္ ။
7. ထို႕ေနာက္ UserSchema ဆိုသည့္ object ကို exports လုပ္ပါမည္။

Program မွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UserSchema = new Schema({
  forename: String,
  surname: String,
  email: { type: String, required: true },
  password: { type: String, required: true },
  age: Number,
  team: String
});

module.exports = mongoose.model('user', UserSchema);
```

STEP 3

1. controllers ဆိုသည့္ folder တစ္ခု တည္ေဆာက္ပါမည္ ။ ထို folder ထဲတြင္ UserController.js ဆိုသည့္ js file တစ္ခု တည္ေဆာက္ပါမည္ ။
2. models folder ထဲမွ UserModels ဆိုသည့္ file ကို ေခၚသံုးမွာ ျဖစ္ၿပီး အတြက္ UserModel ဆိုသည့္ object တစ္ခု ေၾကျငာေပးရမည္ ။
3. module.exports ဆိုသည့္ object တစ္ခု တည္ေဆာက္ပါမည္ ထို object ထဲတြင္ create , update , delete and get တို႕ကို ေရးပါမည္ ။

```
const UserModel = require('../models/UserModels');

module.exports = {
create:(){},
```

```
update: (){},
retrieve: (){},
delete: (){}

}
```

STEP 4 create

- **create route အတြက် user ဆိုသည့် object တစ်ခု တည်ေဆာကွါမည် ။ ထို object ထဲတြင် forename , surname , email , password , age , team တို့ကို client ဘကွ် ရယူပါမည် ။**
- **ျပီးေနာက် save function ကို သံုးျပီး mongodb ထဲသို့ သြားထည့ွါမည် ။**
- **Error မတကွါက json file မ်ားကို respon ျပန္ပို့ေပးေသာ code မ်ားကို ေရးပါမည် ။**
- **Error တကွါက error log မ်ားကို ျပန္ျပ ေပးပါမည် ။**

```
create: (req, res) => {
  let user = new UserModel({
    forename: req.body.forename,
    surname: req.body.surname,
    email: req.body.email,
    password: req.body.password,
    age: req.body.age,
    team: req.body.team
  });

  user.save()
    .then(result => {
      res.json({ success: true, result: result });
    })
    .catch(err => {
      res.json({ success: false, result: err });
    });
},
```

STEP 5 update

I. **update route ကို ေရးပါမည် ။ update လုပ္ခဲ့ id ကို client ဘကွ် ယူပါမယ္ ။**
II. **ျပီးရင္ တစ္ခါတည့ွး update လုပွါမည် ။**

III. ထို id နဲ့ မရှိဘူးဆိုရင် User does not exist ဆိုတဲ့ စာသားကို ေဖာ္ျပေပးပါမည္ ။
IV. id ရွိပါက Update လုပ္ျပီး ျပန္ျပ ေပးပါမည္ ။
V. error ျဖစ္ပါက error msg ကို ျပန္ျပ ေပးပါမည္ ။

```
update: (req, res) => {
  UserModel.update({_id: req.body._id}, req.body)
    .then(user => {
      if (!user) res.json({ success: false, result: "User does not exist" });

      res.json(user);
    })
    .catch(err => {
      res.json({ success: false, result: err });
    });
 },
```

STEP 6 read

1. database ထဲမွ ရွိေသာ data မ်ာ ားအားလံုး ျပန္ျပ ေပးရန္ retrieve ဆိုသည့္ route တစ္ခုကို တည္ေဆာက္ပါေပးပါမည္ ။
2. database ထဲမွ data အားလံုးကို find() method ကိုသံုးျပီးရွာပါမည္ ။
3. database ထဲမွာ data မ်ားရွိပါက result မ်ား အားလံုးကို ျပန္လည္ ေဖာ္ျပ ေပးပါမည္ ။
4. မရွိပါက မရွိေျကာင္းနွင့္ error ျဖစ္ပါက error message မ်ားကို ျပန္လည္ ေဖာ္ျပ ေပးပါမည္။

```
retrieve: (req, res) => {
  UserModel.find()
    .then(result => {
      if (!result) res.json({ success: false, result: "No results found" });

      res.json({ success: true, result: result });
    })
    .catch(err => res.json({success: false, result: err}));
 },
```

STEP 7 delete

1. Client ဘက္မွ id ကို request လုပ္ပါမည္ ။ ျပီးလွ်င္ remove ဆိုသည့္ method ကိုသံုးျပီး ဖ်က္ျပစ္ပါမည္ ။
2. Id ကို မေတြ့ပါက id ကို မေတြ့ေျကာင္းျပမည္ျဖစ္ျပီး
3. Error တက္ပါ error log ကို ျပမည္ျဖစ္ပါသည္ ။

အထက္ပါ အဆင့္မ်ား အားလံုးျပီးပါက UserController.js ထဲမွ code အားလံုးမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const UserModel = require('../models/UserModels');
module.exports = {
 create: (req, res) => {
  let user = new UserModel({
   forename: req.body.forename,
   surname: req.body.surname,
   email: req.body.email,
   password: req.body.password,
   age: req.body.age,
   team: req.body.team
  });
  user.save()
   .then(result => {
    res.json({ success: true, result: result });
   })
   .catch(err => {
    res.json({ success: false, result: err });
   });
 },
 update: (req, res) => {
  UserModel.update({_id: req.body._id}, req.body)
   .then(user => {
    if (!user) res.json({ success: false, result: "User does not exist" });
    res.json(user);
   })
   .catch(err => {
    res.json({ success: false, result: err });
   });
 },
 retrieve: (req, res) => {
  UserModel.find()
   .then(result => {
    if (!result) res.json({ success: false, result: "No results found" });
    res.json({ success: true, result: result });
   })
   .catch(err => res.json({success: false, result: err}));
 },
 delete: (req, res) => {
  UserModel.remove({_id: req.body._id})
   .then(result => {
    if (!result) res.json({ success: false, result: "No user was found with the ID" });
    res.json({ success: true, result: result });
```

```
    })
    .catch(err => res.json({ success: false, result: err }));
 }
}
```

ေနာက္ဆံုး အဆင့္ အေနျဖင့္ app.js ထဲတြင္ route file မ်ားအတြက္ route မ်ား ကို ေရးေပးရပါမည္ ။ mongo ကို သံုးမွာ ျဖစ္သည့္အတြက္ mongoose ကိုလည္း ေျကျငာေပးရန္လိုအပ္ပါသည္ ။ ယခု အဆင့္ျပီးပါက app.js ထဲတြင္ ရွိေသာ code အားလံုးမွာ ေအာက္ပါ အတိုင္းျဖစ္သည္ ။

```
const express=require('express')
const mongoose = require('mongoose');
const app=express();

// for mongoose connection
const UserControl = require('./controllers/UserControl');
const
url='mongodb+srv://winhtut:123456wh@loginwinhtut-e6gu8.mongodb.net/test?retryWrit
es=true&w=majority'

// Database
mongoose.connect(url, { useNewUrlParser: true })
  .then(() => console.log('Connected to database...'))
  .catch(err => console.error(err));

// Middleware
app.use(express.urlencoded({ extended: true }));
app.use(express.json());

// Routes
app.post('/api/user/create', UserControl.create);
app.post('/api/user/update', UserControl.update);
app.get('/api/user/retrieve', UserControl.retrieve);
app.delete('/api/user/delete', UserControl.delete);

var port=process.env.PORT || 3000;
app.listen(port,(req,res)=>{
console.log(`A journey on ${port}.....`);
})
```

အထက္ပါ အဆင့္အားလံုးျပီးပါက program အားလံုးကို save ျပီး postman ကို ဖြင့္ပါ ။ ျပီးေနာ္ url တြင္ create ကိုသြားပါ http://localhost:3000/api/user/create ျပီးလွ်င္ method တြင္ POST ကိုေရြးပါ body ထဲတြင္ ေအာက္ပါ data မ်ားကို ေရးပါ။

```
{
"forename" :"WinHtut",
"surname" :"Htut",
"email" :"winhtutonline@gmail.com",
"password" :1234567,
"age" :24,
"team" :"GreenHackers"
}
```

အထက္ပါ အတိုင္း အားလံုး ေရးျပီး POST ကို နွိပ္ျကည့္ပါ။