

Module 3: Asynchronous Programming

Demo Document 1

edureka!

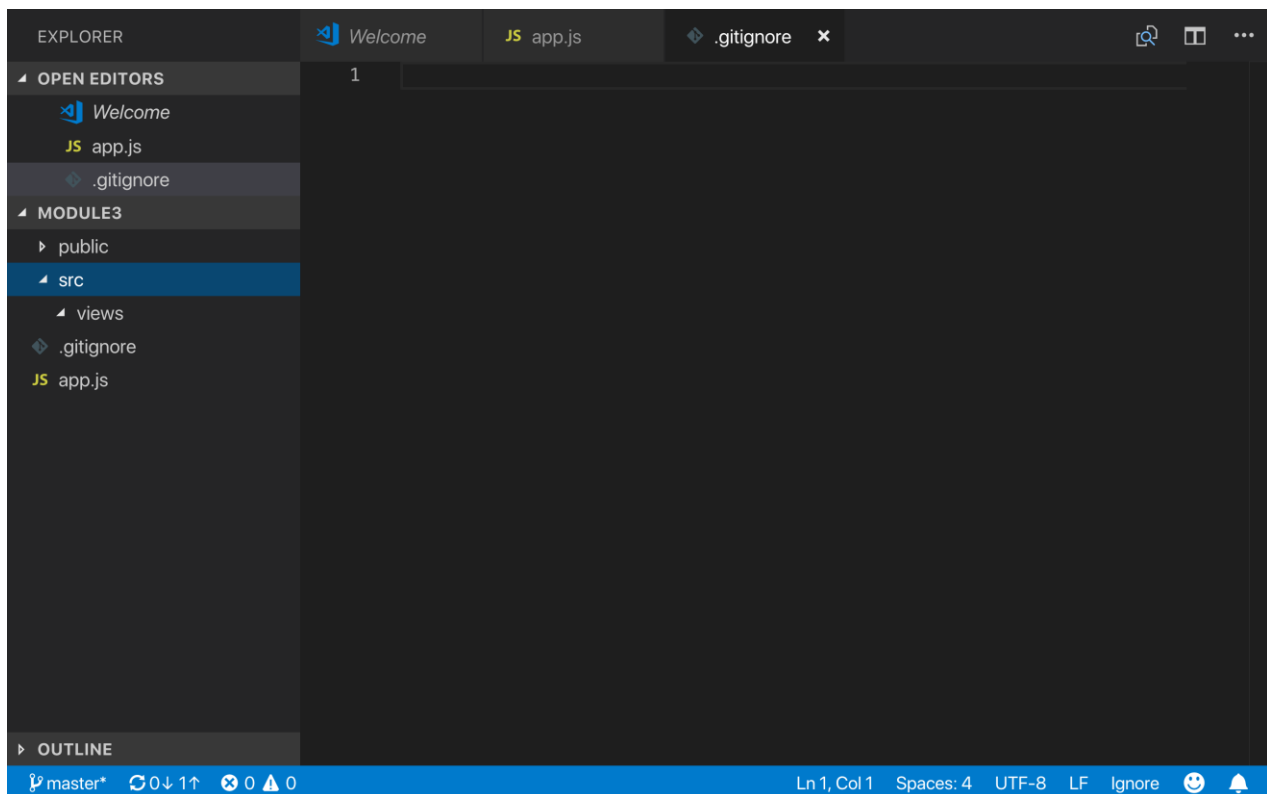
edureka!

© Brain4ce Education Solutions Pvt. Ltd.

Creating Weather Application Using Nodejs, Express, EJS, Promise And Callback

Api used is of OpenWeather : <https://openweathermap.org/api>

Step 1: Create new folder named **module3**. This time add 2 new folder named **public** and **src** for view and static files as well as added .gitignore file to stop pushing node_modules to GitHub after that navigate to the folder with terminal or command prompt.



Step 2: On terminal run command “**npm init**” and at the end write yes to create file a **package.json**

```
Avyaans-MacBook-Pro:module3 avi$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (module3) weatherapp
version: (1.0.0)
description: Node App with EJS
entry point: (app.js)
test command:
git repository:
keywords: Node EJS
author: Edureka
license: (ISC)
About to write to /Users/avi/Desktop/folder/EdurekaApp/module3/package.json:

{
  "name": "weatherapp",
  "version": "1.0.0",
  "description": "Node App with EJS",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "Node",
    "EJS"
  ],
  "author": "Edureka",
  "license": "ISC"
}

Is this OK? (yes) yes
```

Step 3: Install packages locally in the folder

Package name: **Express, EJS and Request**

```
Avyaans-MacBook-Pro:module3 avi$ npm install express ejs request
```

Step 4: Wait for package to install successfully and then navigate to Text Editor (Visual studio code) for implementation.

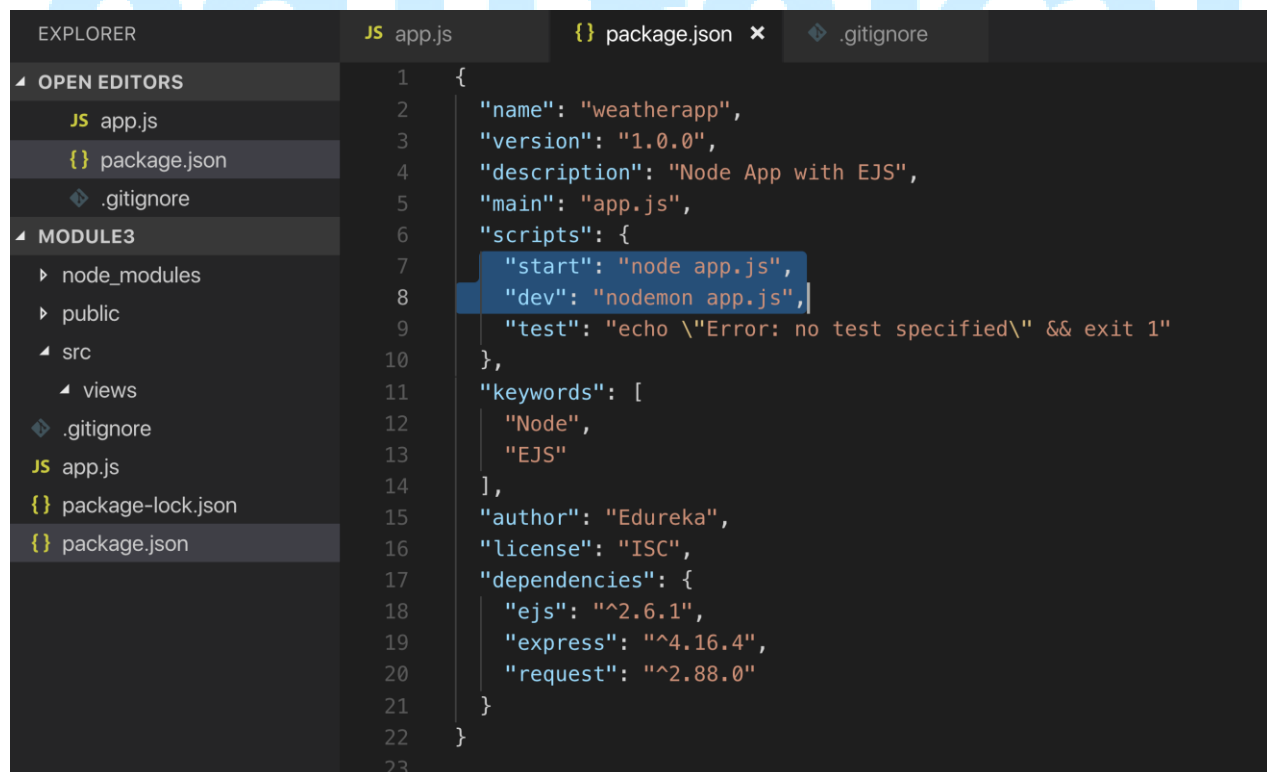
```
npm notice created a lockfile as package-lock.json. You
should commit this file.
npm WARN weatherapp@1.0.0 No repository field.

+ request@2.88.0
+ ejs@2.6.1
+ express@4.16.4
added 92 packages from 88 contributors and audited 185
packages in 6.485s
found 0 vulnerabilities
```

Step 5: In package.json add “start” and “dev” command to run application in production and Development mode

For Production use “**npm start**”

For Dev use “**npm run dev**”

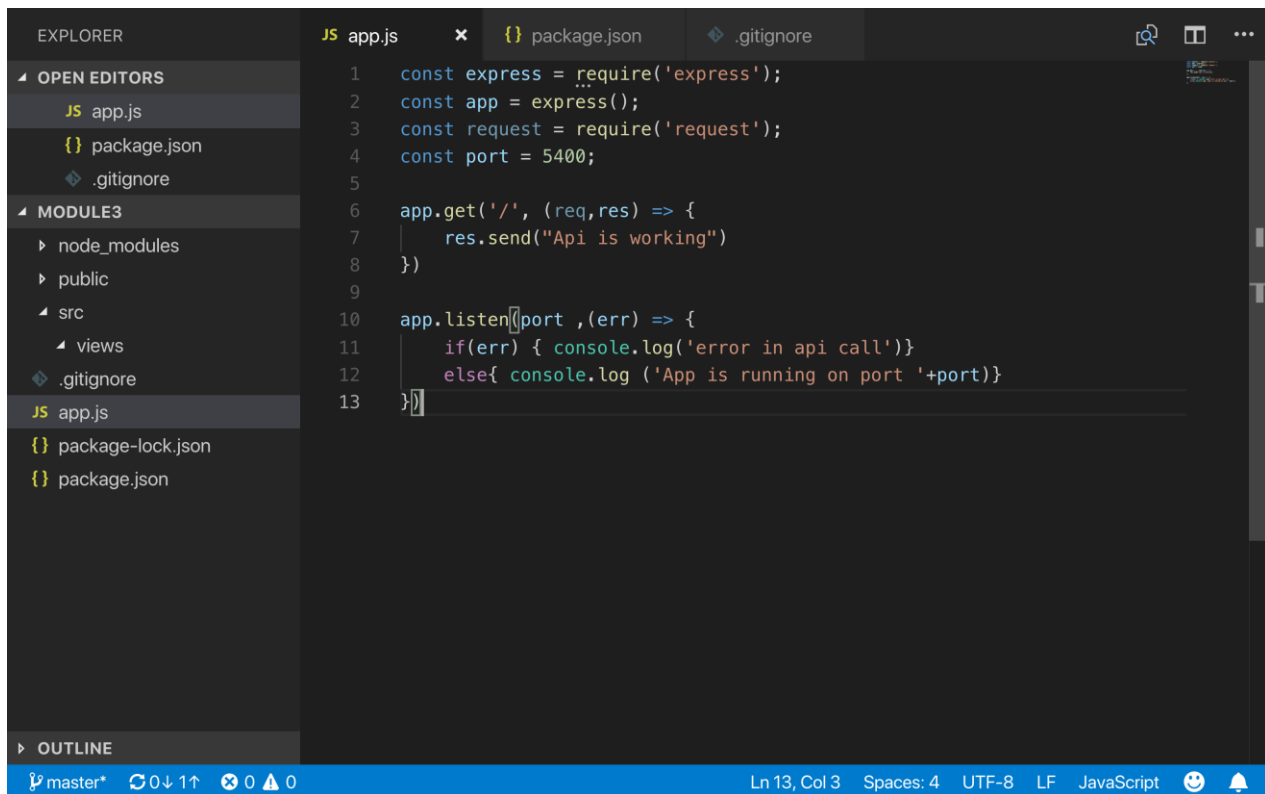


```
EXPLORER JS app.js {} package.json x .gitignore
OPEN EDITORS
JS app.js
{} package.json
.gitignore
MODULE3
  node_modules
  public
  src
    views
.gitignore
JS app.js
{} package-lock.json
{} package.json

1 {
2   "name": "weatherapp",
3   "version": "1.0.0",
4   "description": "Node App with EJS",
5   "main": "app.js",
6   "scripts": {
7     "start": "node app.js",
8     "dev": "nodemon app.js",
9     "test": "echo \"Error: no test specified\" && exit 1"
10  },
11  "keywords": [
12    "Node",
13    "EJS"
14  ],
15  "author": "Edureka",
16  "license": "ISC",
17  "dependencies": {
18    "ejs": "^2.6.1",
19    "express": "^4.16.4",
20    "request": "^2.88.0"
21  }
22 }
23
```

Step 6: Create Server using express and use port number here we are using “5400”

And create one default Route as test route of the application. After creating navigate to terminal for running application.



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure with files like app.js, package.json, .gitignore, and node_modules. The main editor area shows the content of app.js, which is a JavaScript file for creating an Express.js server. The code includes imports for express and request, setting the port to 5400, defining a GET route for '/', and listening on the specified port with error handling. The status bar at the bottom indicates the current file is app.js, line 13, column 3, with 4 spaces, UTF-8 encoding, and LF line endings.

```
1  const express = require('express');
2  const app = express();
3  const request = require('request');
4  const port = 5400;
5
6  app.get('/', (req,res) => {
7    res.send("Api is working")
8  })
9
10 app.listen(port ,(err) => {
11   if(err) { console.log('error in api call')}
12   else{ console.log ('App is running on port '+port)}
13 })
```

Step 7: Run app using “ npm run dev” and this will allow app to run with nodemon.

It keep on detecting the changes and restart our server this help in fast development.

```
Avyaans-MacBook-Pro:module3 avi$ npm run dev

> weatherapp@1.0.0 dev /Users/avi/Desktop/folder/EdurekaApp/module3
> nodemon app.js

[nodemon] 1.18.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node app.js`
App is running on port 5400
```

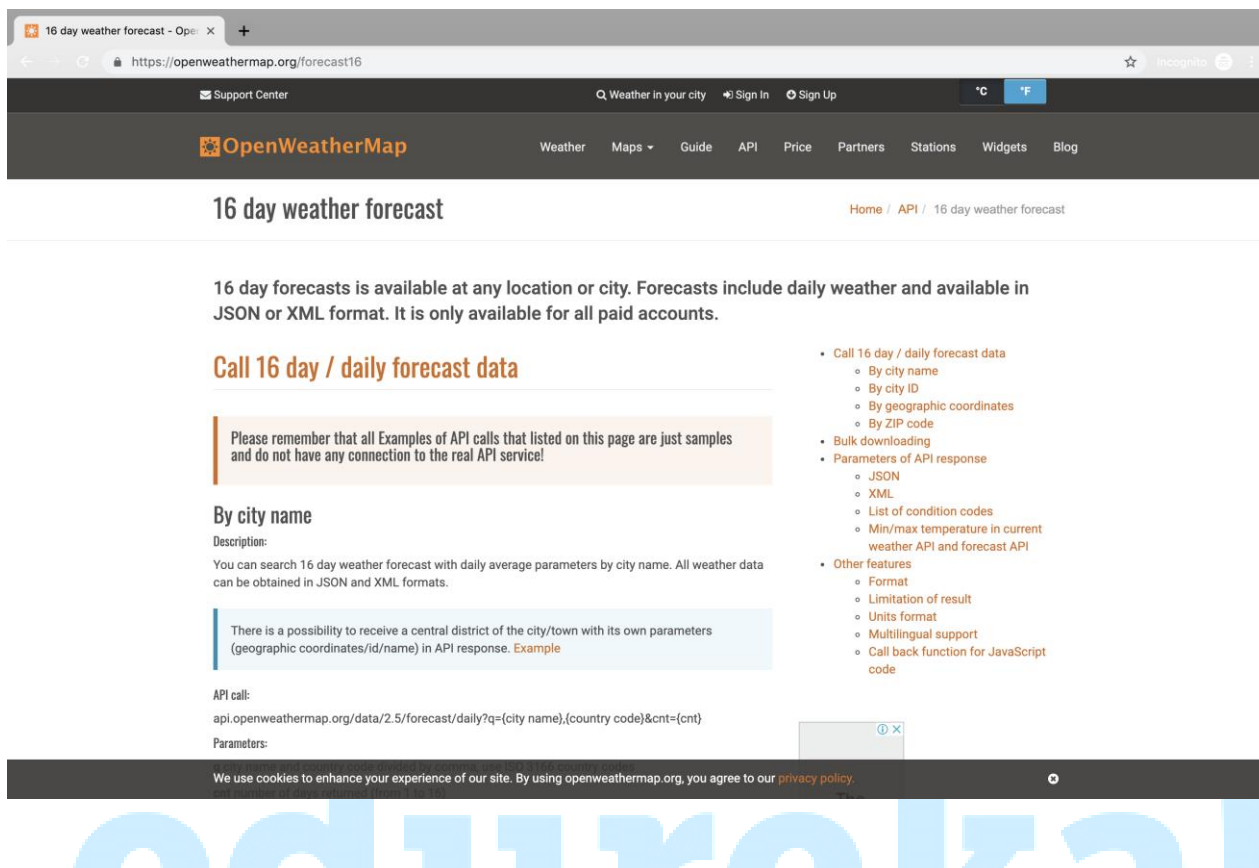


Step 8: Test App on browser using “<http://localhost:5400>” I will run default router with output.

Step 9: Create Account on open weather map website <https://openweathermap.org/api>

and get **api keys** for limited number of hits

Note: You must wait for few hours to get your key activated.



16 day weather forecast

16 day forecasts is available at any location or city. Forecasts include daily weather and available in JSON or XML format. It is only available for all paid accounts.

Call 16 day / daily forecast data

Please remember that all Examples of API calls that listed on this page are just samples and do not have any connection to the real API service!

By city name

Description:
You can search 16 day weather forecast with daily average parameters by city name. All weather data can be obtained in JSON and XML formats.

There is a possibility to receive a central district of the city/town with its own parameters (geographic coordinates/id/name) in API response. [Example](#)

API call:
`api.openweathermap.org/data/2.5/forecast/daily?q={city name},{country code}&cnt={cnt}`

Parameters:

- Call 16 day / daily forecast data
 - By city name
 - By city ID
 - By geographic coordinates
 - By ZIP code
- Bulk downloading
- Parameters of API response
 - JSON
 - XML
 - List of condition codes
 - Min/max temperature in current weather API and forecast API
- Other features
 - Format
 - Limitation of result
 - Units format
 - Multilingual support
 - Call back function for JavaScript code

Step 10: Provide api url with key in your app server

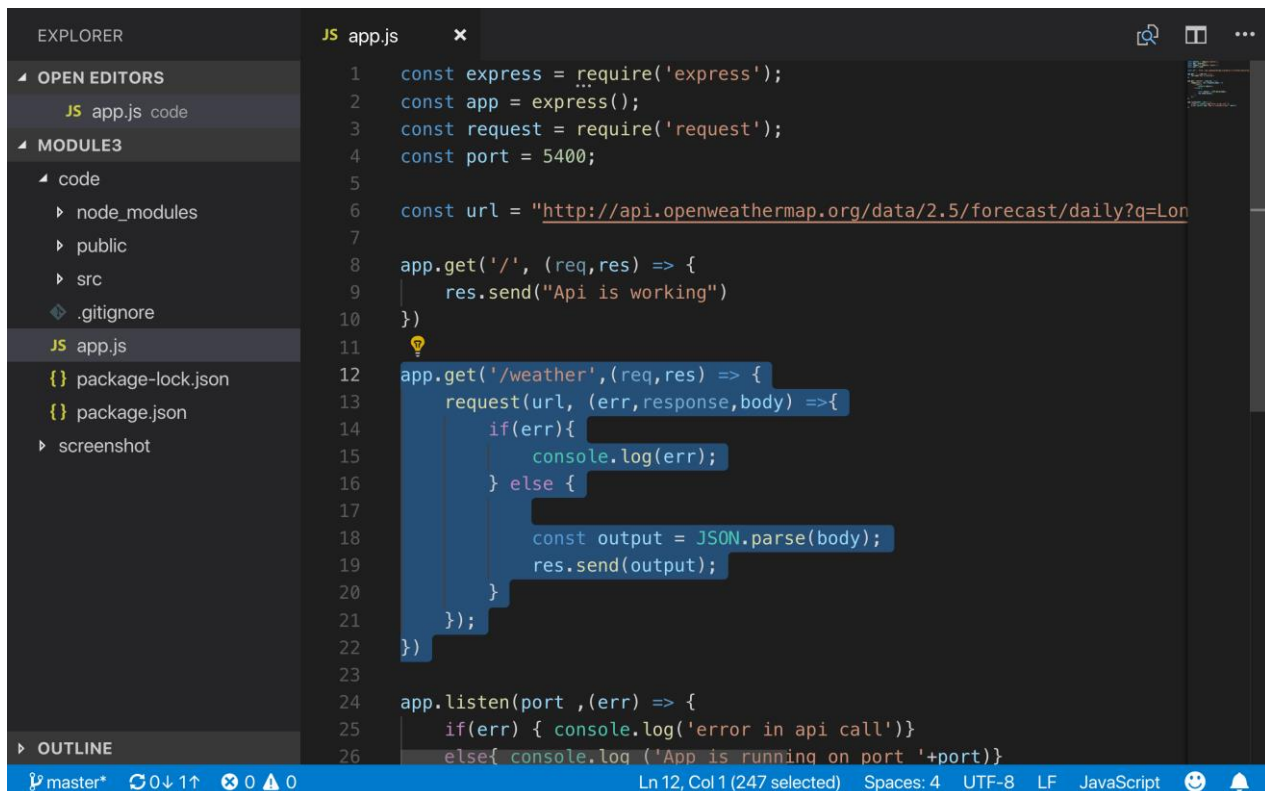
Url Sample:

<http://api.openweathermap.org/data/2.5/forecast/daily?q=London&mode=json&units=metric&cnt=5&appid=your api key>

```
const express = require('express');
const app = express();
const request = require('request');
const port = 5400;

const url = "http://api.openweathermap.org/data/2.5/forecast/daily?q=Lon
```

Step 11: With the use of 'request' module we will call the api in callback response we will get



```

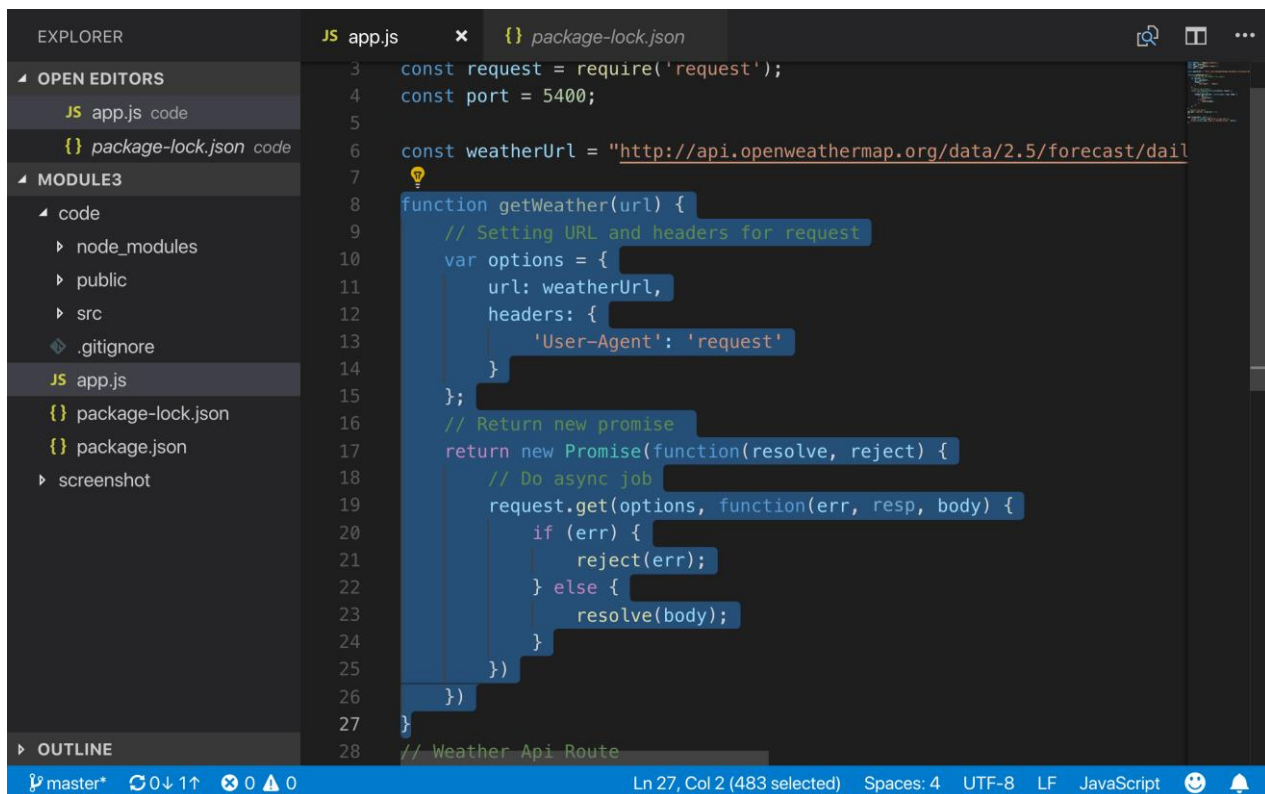
1  const express = require('express');
2  const app = express();
3  const request = require('request');
4  const port = 5400;
5
6  const url = "http://api.openweathermap.org/data/2.5/forecast/daily?q=Lon
7
8  app.get('/', (req,res) => {
9    res.send("Api is working")
10 })
11
12 app.get('/weather',(req,res) => {
13   request(url, (err,response,body) =>{
14     if(err){
15       console.log(err);
16     } else {
17
18       const output = JSON.parse(body);
19       res.send(output);
20     }
21   });
22 })
23
24 app.listen(port ,(err) => {
25   if(err) { console.log('error in api call')}
26   else{ console.log ('App is running on port '+port)}

```

data according to city provided. This call is without promises a simple node callback

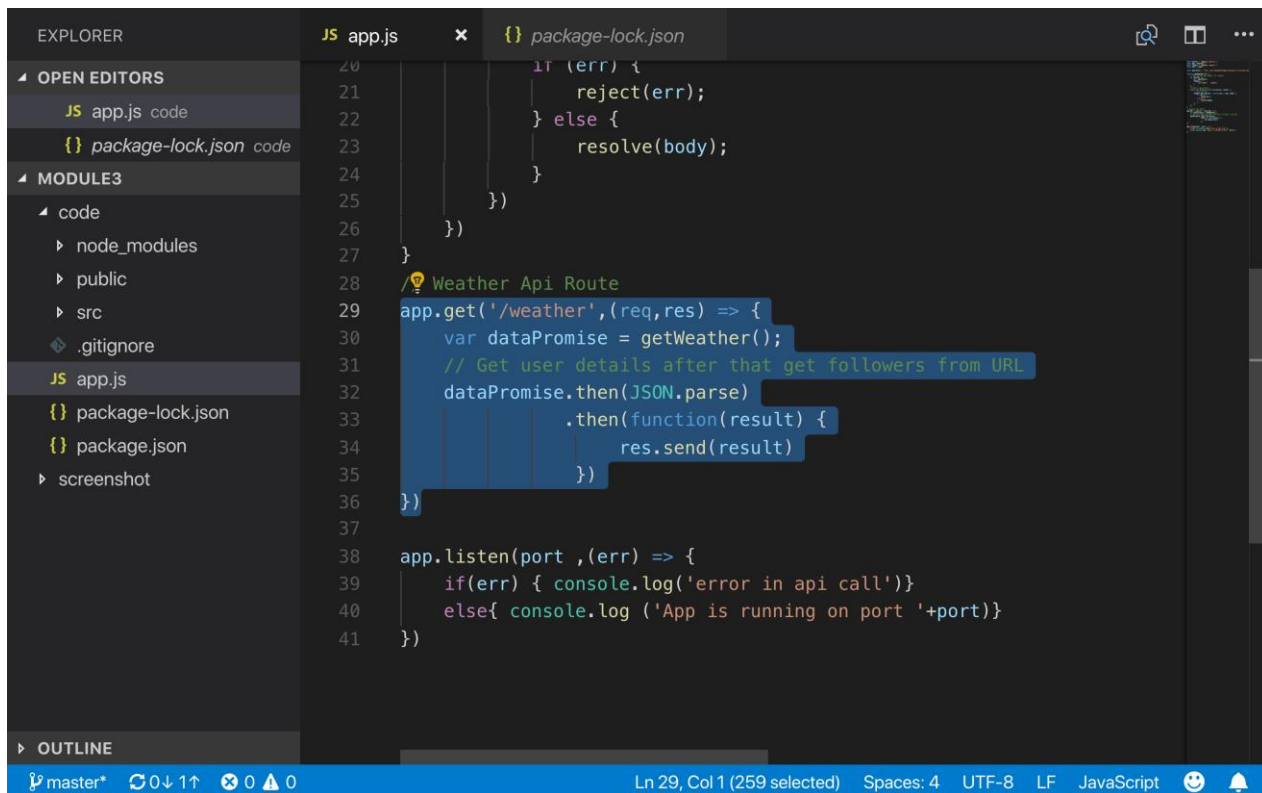
Step 13: Now we are calling same api with promise. The difference is syntax as well as with promises its easy to handle response and error as we are waiting for Api to get resolve with the promise Object.

Below is the function that call Api with promise



```
3  const request = require('request');
4  const port = 5400;
5
6  const weatherUrl = "http://api.openweathermap.org/data/2.5/forecast/dail
7
8  function getWeather(url) {
9      // Setting URL and headers for request
10     var options = {
11         url: weatherUrl,
12         headers: {
13             'User-Agent': 'request'
14         }
15     };
16     // Return new promise
17     return new Promise(function(resolve, reject) {
18         // Do async job
19         request.get(options, function(err, resp, body) {
20             if (err) {
21                 reject(err);
22             } else {
23                 resolve(body);
24             }
25         })
26     })
27 }
28 // Weather Api Route
```

Step 14: Now as above function responses back data now we must resolve promise and get data out of promise

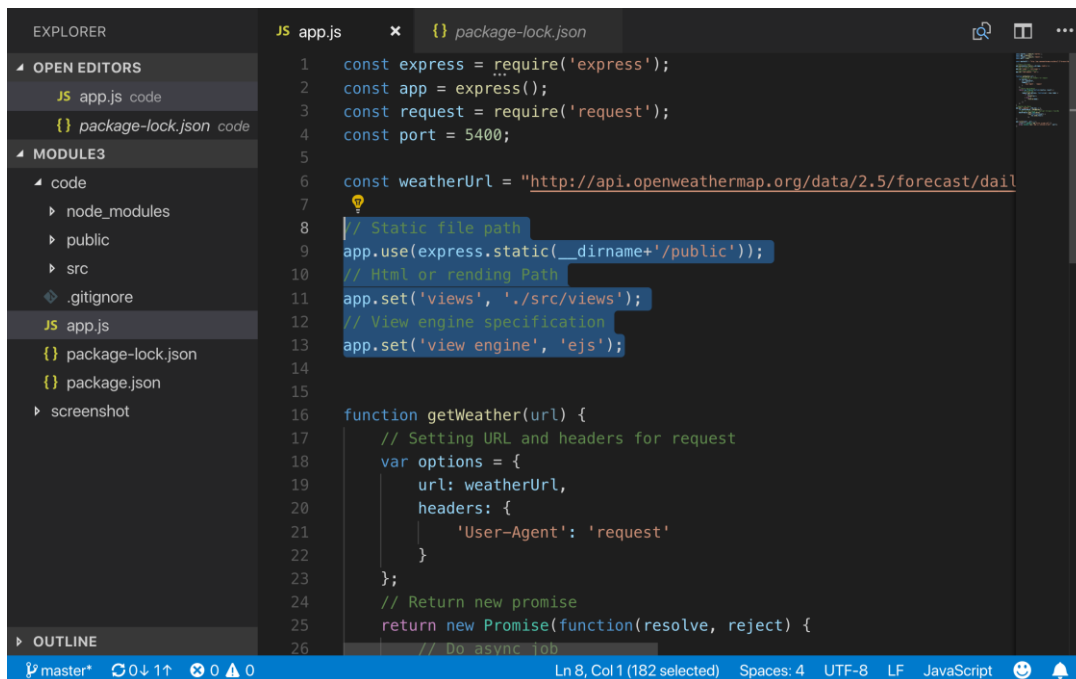


The screenshot shows a VS Code editor with two files open: `app.js` and `package-lock.json`. The `app.js` file contains the following code:

```
20     if (err) {
21         reject(err);
22     } else {
23         resolve(body);
24     }
25 })
26 })
27 }
28 // Weather Api Route
29 app.get('/weather',(req,res) => {
30     var dataPromise = getWeather();
31     // Get user details after that get followers from URL
32     dataPromise.then(JSON.parse)
33     .then(function(result) {
34         res.send(result)
35     })
36 })
37
38 app.listen(port ,(err) => {
39     if(err) { console.log('error in api call')}
40     else{ console.log ('App is running on port '+port)}
41 })
```

The Explorer sidebar on the left shows the project structure with folders like `node_modules`, `public`, `src`, and files like `app.js`, `package-lock.json`, and `package.json`. The status bar at the bottom indicates the current file is `app.js` at line 29, column 1, with 259 characters selected. The editor is using the JavaScript language and UTF-8 encoding.

Step 15: Lets add view layer using EJS we will use EJS as middle ware and add path of public directory.



The screenshot shows a VS Code editor window with the following code in `app.js`:

```
1 const express = require('express');
2 const app = express();
3 const request = require('request');
4 const port = 5400;
5
6 const weatherUrl = "http://api.openweathermap.org/data/2.5/forecast/dail
7
8 // Static file path
9 app.use(express.static(__dirname+'/public'));
10 // Html or rendering Path
11 app.set('views', './src/views');
12 // View engine specification
13 app.set('view engine', 'ejs');
14
15
16 function getWeather(url) {
17   // Setting URL and headers for request
18   var options = {
19     url: weatherUrl,
20     headers: {
21       'User-Agent': 'request'
22     }
23   };
24   // Return new promise
25   return new Promise(function(resolve, reject) {
26     // Do async job
```

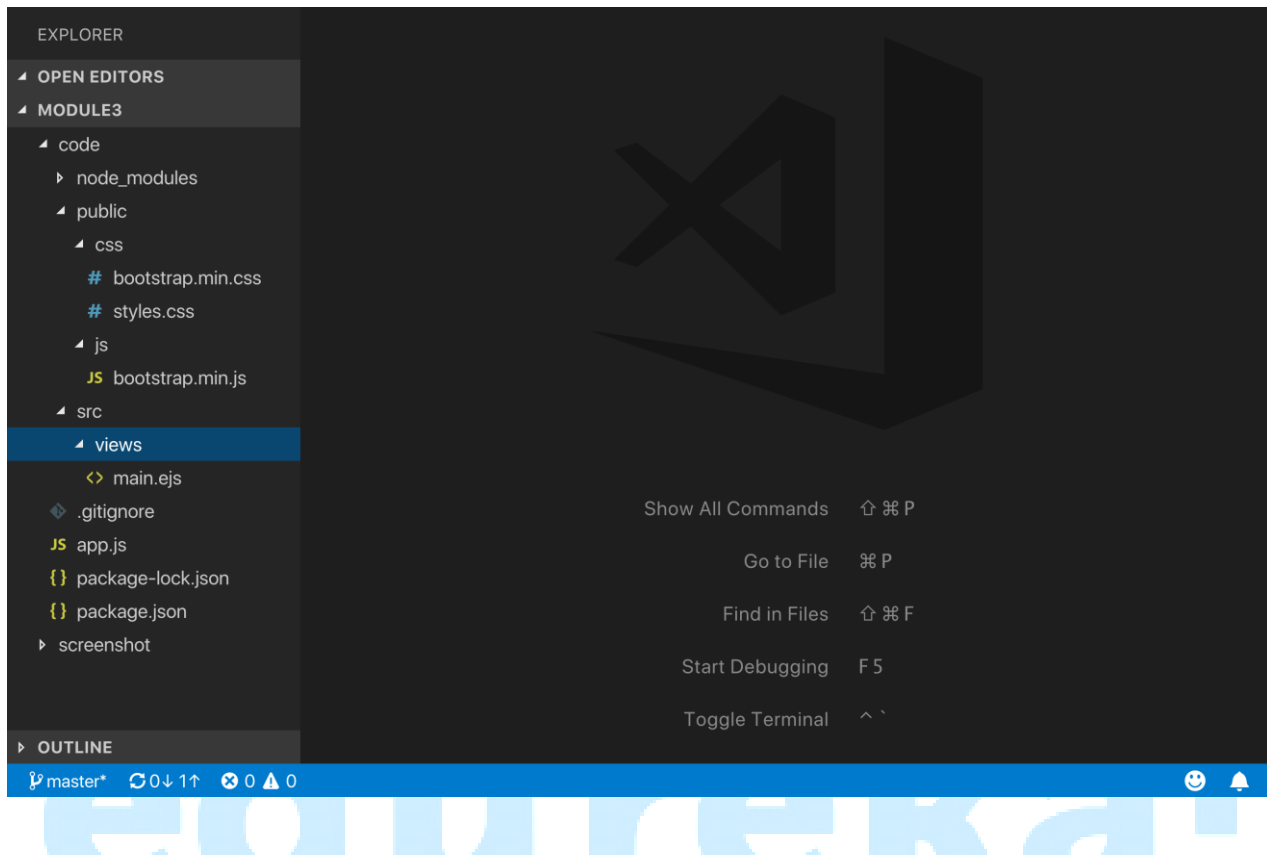
The Explorer sidebar on the left shows the project structure:

- code
 - node_modules
 - public
 - src
- .gitignore
- JS app.js
- { } package-lock.json
- { } package.json
- screenshot

The status bar at the bottom indicates: master*, 0 down 1 up, 0 errors 0 warnings, Ln 8, Col 1 (182 selected), Spaces: 4, UTF-8, LF, JavaScript.

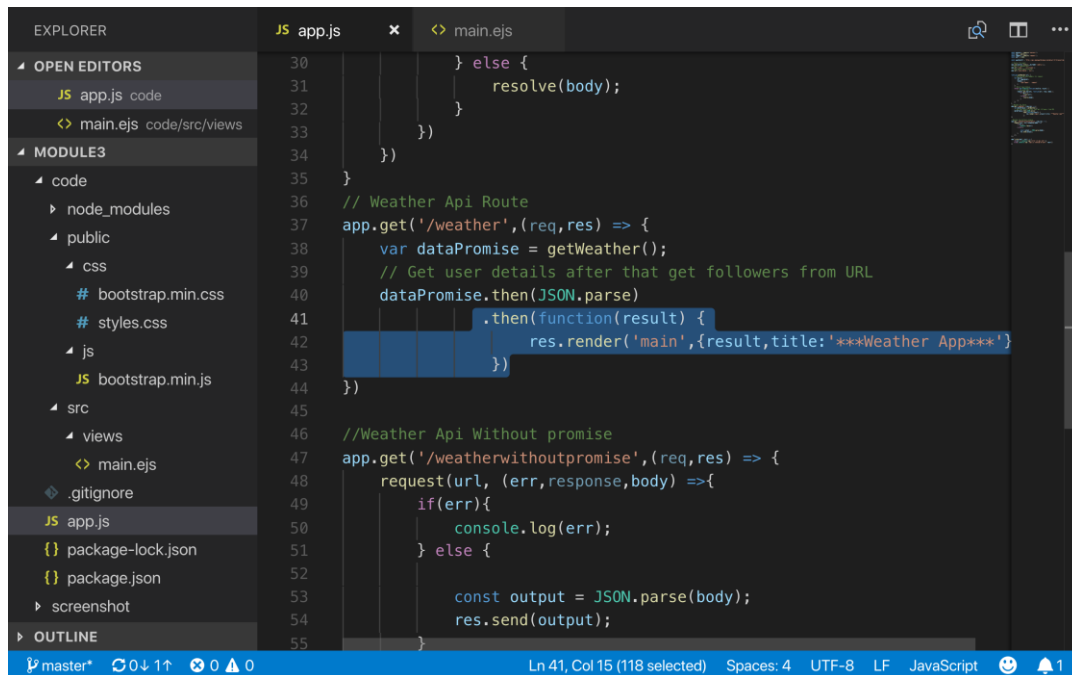
edureka!

Step 16: Now we will create one folder for view as well as public folder where we can keep our



css and Js as well as any other static files.

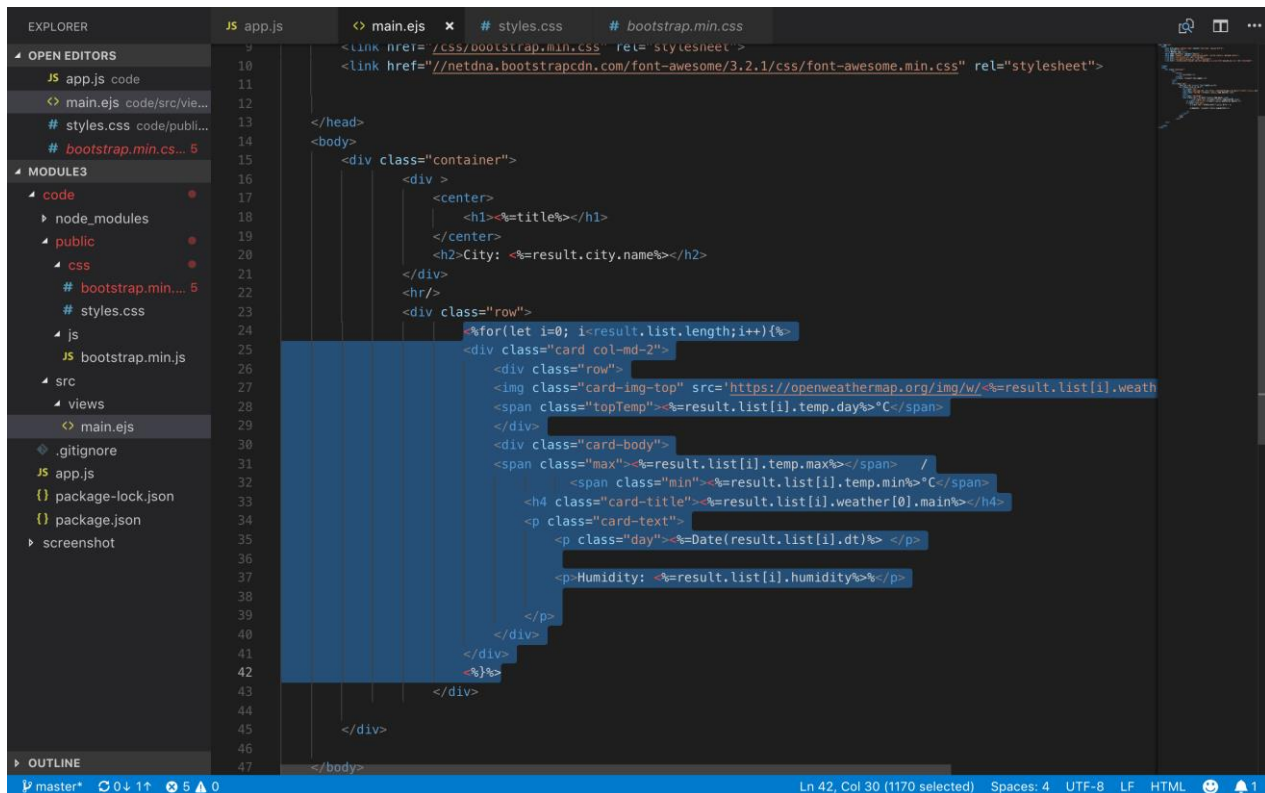
Step 17: Instead of `res.send` we will use `res.render` to display data on view. Here will provide the path as well as send data as JSON.



The screenshot shows the Visual Studio Code editor interface. On the left, the Explorer sidebar displays the project structure: 'code' folder containing 'node_modules', 'public' (with 'css' and 'js' subfolders), and 'src' (with 'views' subfolder). The 'main.ejs' file is selected in the 'views' folder. The main editor area shows the content of 'main.ejs', which includes a JavaScript route handler for '/weather' using Express.js. The code uses the 'then' method of a Promise to handle asynchronous data retrieval. A line of code, `res.render('main', {result, title: '***Weather App***'})`, is highlighted with a blue selection. The status bar at the bottom indicates the current position is Line 41, Column 15, with 118 characters selected. It also shows settings for 4 spaces, UTF-8 encoding, LF line endings, and the JavaScript language mode.

```
30     } else {
31         resolve(body);
32     }
33 }
34 })
35 }
36 // Weather Api Route
37 app.get('/weather',(req,res) => {
38     var dataPromise = getWeather();
39     // Get user details after that get followers from URL
40     dataPromise.then(JSON.parse)
41     .then(function(result) {
42         res.render('main',{result,title:'***Weather App***'})
43     })
44 })
45
46 //Weather Api Without promise
47 app.get('/weatherwithoutpromise',(req,res) => {
48     request(url, (err,response,body) =>{
49         if(err){
50             console.log(err);
51         } else {
52
53             const output = JSON.parse(body);
54             res.send(output);
55         }
56     })
57 })
```

Step 18: We can include any of the css framework to make our app look. Here we are using Bootstrap for css as well as adding For Loop using EJS to iterate the data



The screenshot shows a VS Code editor with the following files open: `main.ejs`, `styles.css`, and `bootstrap.min.css`. The `main.ejs` file contains the following code:

```
9 <!-- Bootstrap CSS -->
10 <link href="/css/bootstrap.min.css" rel="stylesheet">
11
12 <!-- Font Awesome CSS -->
13 <link href="//netdna.bootstrapcdn.com/font-awesome/3.2.1/css/font-awesome.min.css" rel="stylesheet">
14
15 </head>
16 <body>
17
18   <div class="container">
19     <div>
20       <center>
21         <h1><%=title%></h1>
22       </center>
23       <h2>City: <%=result.city.name%></h2>
24     </div>
25     <hr/>
26     <div class="row">
27       <%=for(let i=0; i<result.list.length;i++){%>
28       <div class="card col-md-2">
29         <div class="card-img-top">
30           
31         </div>
32         <div class="card-body">
33           <span class="max"><%=result.list[i].temp.max%></span> /
34           <span class="min"><%=result.list[i].temp.min%></span> "C"</span>
35           <h4 class="card-title"><%=result.list[i].weather[0].main%></h4>
36           <p class="card-text">
37             <p class="day"><%=Date(result.list[i].dt)> </p>
38             <p>Humidity: <%=result.list[i].humidity%></p>
39           </p>
40         </div>
41       </div>
42       <%=}%>
43     </div>
44   </div>
45
46 </body>
```

Step 19: Below is CSS file its completely open to use any css properties as well as any style as you like.

```

1  .card{
2    width: 211px;
3    border-radius: 5%;
4    background: #051258b0;
5    color: #fff;
6    margin-left: 3%;
7  }
8  .card-hover { transform: scale(1.1); }
9  .card-img-top{
10   width: 44%
11 }
12
13 .headingh1{
14   text-align: center;
15   color: #rgb(255, 255, 255);
16   text-shadow: 2px 2px #28346c;
17 }
18
19 .max{
20   color: #rgb(166, 226, 236);
21   font-weight: bold;
22 }
23
24 .min{
25   color: #rgb(104, 155, 241);
26   font-weight: bold;
27 }
28
29 .day{
30   color: #rgb(234, 255, 124);
31 }
32
33 .humid{
34   color: #rgb(44, 112, 105);
35 }
36
37 .topTemp{
38   font-size: 26px;
39   color: #white;
40   font-weight: bold;
41 }

```

Step 20: One everything is setup this will be final response and this response may vary according to your style.

