

1 Node.js y Web de Información meteorológica

1.1 Sumario

- 1 Introducción a la aplicación de meteorología
- 2 Creación de cuenta en OpenWeatherMap
- 3 Proyecto Node.js ejecución desde consola
 - ◆ 3.1 Creación de fichero index.js
 - ◆ 3.2 Haciendo llamadas a la API REST de OpenWeatherMap
 - ◆ 3.3 Gestionando la respuesta obtenida desde la API REST de OpenWeatherMap
 - ◆ 3.4 Añadiendo interactividad desde la línea de comandos
- 4 Proyecto Node.js ejecución web con Express
 - ◆ 4.1 Creación de fichero index.js
 - ◆ 4.2 Instalación y configuración de Express
 - ◆ 4.3 Creación de la vista index.ejs por defecto
 - ◆ 4.4 Añadiendo un fichero de estilos CSS
 - ◆ 4.5 Configurando ruta POST que recibirá las peticiones
 - ◆ 4.6 Petición a la API REST de OpenWeatherMap
 - ◆ 4.7 Modificación de la plantilla index.ejs para mostrar el resultado de la petición
 - ◆ 4.8 Mejora de la página de resultados para mostrar más información e iconos de méteo

2 Introducción a la aplicación de meteorología

- Vamos a desarrollar una aplicación en aproximadamente **30 minutos** que nos permitirá mostrar la meteorología de cualquier país desde la **línea de comandos**.
- Luego dedicaremos otros **30 minutos** y realizaremos la misma **aplicación con interfaz web**.

Para ello necesitaremos hacer uso de:

1. **API REST de OpenWeatherMap.org**
2. **Node.js**

3 Creación de cuenta en OpenWeatherMap

Necesitaremos crear una cuenta en <https://openweathermap.org/> para poder hacer uso de su servicio de méteo a través de la API REST que proporciona.

- Nos registraremos de forma gratuita en https://home.openweathermap.org/users/sign_up para poder obtener la API Key.
- Vamos a hacer uso de la versión gratuita de dicho servicio. Más información aquí: <https://openweathermap.org/appid>
- Cuando nos registremos recibiremos **un correo** en el que nos da la bienvenida y nos facilita nuestra API KEY. Por ejemplo:

```
-----
API key:
- Your API key is ad2938293c2667ffgtwd71c45232179g5
- Within the next couple of hours, it will be activated and ready to use
- You can later create more API keys on your account page
- Please, always use your API key in each API call

Endpoint:
- Please, use the endpoint api.openweathermap.org for your API calls
- Example of API call:
api.openweathermap.org/data/2.5/weather?q=London,uk&APPID=ad238730a6c2667eedfd71c7c68179f8

Useful links:
- API documentation https://openweathermap.org/api
- Details of your plan https://openweathermap.org/price
- Please, note that 16-days daily forecast and History API are not available for Free subscribers
```

- Si queremos ver las **API Key** que tenemos, podemos consultarlas en la web en la sección de API keys: https://home.openweathermap.org/api_keys. Incluso podemos tener varias y darles un nombre distinto para diferenciarlas. Por ejemplo:

You can generate as many API keys as needed for your subscription. We

Key

Name

Méteo con Nod

4 Proyecto Node.js ejecución desde consola

- Vamos a crearnos la **configuración básica** del proyecto en Node.js.
- Para ello nos haremos una **nueva carpeta**, por ejemplo con el nombre de **meteo**.
- Iniciaremos el proyecto con **npm init** dentro de la carpeta **meteo**.

```
veiga@dws:~/www/nodejs$ mkdir meteo  
veiga@dws:~/www/nodejs$ cd meteo
```

```
veiga@dws:~/www/nodejs/meteo$ 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: (meteo)
version: (1.0.0)
description: Aplicación de Méteo con Node.js y OpenWeatherMap
entry point: (index.js)
test command:
git repository:
keywords: meteo openweathermap node
author: Rafa Veiga
license: (ISC)
About to write to /var/www/veiga.dynu.net/public/nodejs/meteo/package.json:
```

```
{
  "name": "meteo",
  "version": "1.0.0",
  "description": "Aplicación de Méteo con Node.js y OpenWeatherMap",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "meteo",
    "openweathermap",
    "node"
  ],
  "author": "Rafa Veiga",
  "license": "ISC"
}
```

Is this OK? (yes) yes

veiga@dws:~/www/nodejs/meteo\$

- Esta opción creará el fichero de configuración inicial de Nodejs=> **package.json** con un contenido similar al siguiente:

```
veiga@dws:~/www/nodejs/meteo$ cat package.json
{
  "name": "meteo",
  "version": "1.0.0",
  "description": "Aplicación de Méteo con Node.js y OpenWeatherMap",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "meteo",
    "openweathermap",
    "node"
  ],
  "author": "Rafa Veiga",
  "license": "ISC"
}
```

4.1 Creación de fichero index.js

Dentro de la carpeta **meteo** crearemos un fichero **index.js** que contendrá todo el código del servidor y de la aplicación de meteorología.

4.2 Haciendo llamadas a la API REST de OpenWeatherMap

- Para poder hacer llamadas a la **API de OpenWeatherMap**, vamos a usar un módulo muy popular llamado **request**.
- Este módulo simplifica el código necesario para hacer **peticiones http** en Node.
- Lo instalaremos con **npm install request --save**

```
# Instalamos el módulo request guardando la dependencia en nuestro fichero package.json (para facilitar posteriores instalaciones en
npm install request --save
```

Una vez instalado podemos comprobar que en package.json ha añadido la dependencia:

```
veiga@dwes:~/www/nodejs/meteo$ cat package.json
{
  "name": "meteo",
  "version": "1.0.0",
  "description": "Aplicación de Méteo con Node.js y OpenWeatherMap",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "meteo",
    "openweathermap",
    "node"
  ],
  "author": "Rafa Veiga",
  "license": "ISC",
  "dependencies": {
    "request": "^2.88.0"
  }
}
```

- El código inicial de nuestra aplicación **index.js** contendrá algo como:

```
// index.js

// Cargamos el paquete request
const request = require('request');

// A request le pasamos una url, una función de callback con 3 argumentos: err, response y body
request(url, function (err, response, body) {
  // Comprobamos si hay algún error en la petición. Si hay errores, hacemos un log del error.
  // Si no hay errores, hacemos un log de toda la respuesta obtenida en la petición.
  if(err){
    console.log('error:', err);
  } else {
    console.log('body:', body);
  }
});
```

- Vamos a configurar una **URL de ejemplo**, a la cuál le realizaremos la petición, para ver si funciona correctamente.
- Leyendo la documentación o el e-mail de OpenWeatherMap nos indica que tenemos que hacer las peticiones a <http://api.openweathermap.org/data/2.5/weather>
- La URL anterior necesita además pasarle **2 parámetros por GET: q=ciudad** y **appid=NUESTRA-API-KEY**, que nos permitirán indicar la ciudad que queremos consultar y nuestra API Key para poder hacer la petición.
- El código de ejemplo quedaría tal que así:

```
// index.js

// Cargamos el paquete request
const request = require('request');

// Definimos nuestra API KEY
let apiKey = '*****';
let ciudad = 'Lugo';
let url = `http://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=${apiKey}`

// A request le pasamos una url, una función de callback con 3 argumentos: err, response y body
request(url, function (err, response, body) {
  // Comprobamos si hay algún error en la petición. Si hay errores, hacemos un log del error.
  // Si no hay errores, hacemos un log de toda la respuesta obtenida en la petición.
  if(err){
    console.log('error:', error);
  } else {
    console.log('body:', body);
  }
});
```

- Ejecutamos el código desde la terminal:

```
# Ejecutamos el servidor con la ciudad de Lugo de ejemplo:
node index.js
```

```
# Obtendremos algo como:
```

```
body: {"coord":{"lon":-7.5,"lat":43},"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02n"}],"base":"stations"}
```

- Podemos observar que está funcionando correctamente. Vemos que en la **respuesta** nos muestra algo como la temperatura, humedad, velocidad del viento, hora de puesta de sol, hora de amanecer, etc...
- Si nos fijamos vemos que por ejemplo la temperatura aparece en **Fahrenheit**. Si queremos que aparezca en grados **Celsius** tendremos que añadir un parámetro en la URL de petición: **units=metric**.
- Con lo que el código quedará tal que así:

```
// index.js
```

```
// Cargamos el paquete request
const request = require('request');
```

```
// Definimos nuestra API KEY
let apiKey = '*****';
let ciudad = 'Lugo';
let unidad = 'metric';
let url = `http://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=${apiKey}&units=${unidad}`
```

```
// A request le pasamos una url, una función de callback con 3 argumentos: err, response y body
request(url, function (err, response, body) {
  // Comprobamos si hay algún error en la petición. Si hay errores, hacemos un log del error.
  // Si no hay errores, hacemos un log de toda la respuesta obtenida en la petición.
  if(err){
    console.log('error:', error);
  } else {
    console.log('body:', body);
  }
});
```

```
# Ejecutamos de nuevo la petición con la ciudad de Lugo de ejemplo:
node index.js
```

```
# Obtendremos ahora en formato métrico algo como:
```

```
body: {"coord":{"lon":-7.56,"lat":43.01},"weather":[{"id":801,"main":"Clouds","description":"few clouds","icon":"02n"}],"base":"stat"}
```

4.3 Gestionando la respuesta obtenida desde la API REST de OpenWeatherMap

Tal y como recibimos la respuesta de OpenWeatherMap, vamos a procesarla para poder mostrar la información más correctamente.

- Lo primero que vamos a hacer es convertir la **cadena de texto JSON** en su **objeto correspondiente**.
- Para hacer eso lo podemos hacer con la instrucción: **let tiempo=JSON.parse(body)**
- Podemos mostrar un mensaje más concreto con la temperatura.
- Veamos la siguiente variación:

```
// index.js
```

```
// Cargamos el paquete request
const request = require('request');
```

```
// Definimos nuestra API KEY
let apiKey = '*****';
let ciudad = 'Lugo';
let unidad = 'metric';
let url = `http://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=${apiKey}&units=${unidad}`
```

```
// A request le pasamos una url, una función de callback con 3 argumentos: err, response y body
request(url, function (err, response, body) {
  // Comprobamos si hay algún error en la petición. Si hay errores, hacemos un log del error.
  // Si no hay errores, hacemos un log de toda la respuesta obtenida en la petición.
  if(err){
    console.log('error:', error);
  } else {
```

```

    let info= JSON.parse(body);
    let mensaje = `En ${info.name}, en este momento la temperatura es de ${info.main.temp}°C.`;
    console.log(mensaje);
  }
});

```

Probamos el código y obtendremos algo como:

```

node index.js

En Lugo, en este momento la temperatura es de 6.43°C.

```

4.4 Añadiendo interactividad desde la línea de comandos

- En estos momentos la ciudad con la que estamos probando está definida en el código, pero sería más interesante que pudiésemos introducir la ciudad en la línea de comandos.
- Para ello vamos a añadir un módulo **yargs** a nuestro proyecto que nos permite gestionar dicha interacción.

```

# Instalamos el módulo yargs en nuestro proyecto:
npm install yargs --save

```

- Modificamos el código de la aplicación para hacer uso de los argumentos en la línea de comandos:

```

// index.js

// Cargamos el paquete request
const request = require('request');

// Cargamos el módulo yargs
const argv = require('yargs').argv;

// Definimos nuestra API KEY
let apiKey = 'ad238730a6c2667eedfd71c7c68179f8';

// Usaremos el argumento c (de ciudad) o pondremos la ciudad de Lugo por defecto:
let ciudad = argv.c || 'Lugo';

let unidad = 'metric';
let url = `http://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=${apiKey}&units=${unidad}`

console.log("Si desea conocer la temperatura de una ciudad en concreto añadida a la línea de comandos: -c Santiago+de+Compostela ");

// A request le pasamos una url, una función de callback con 3 argumentos: err, response y body
request(url, function (err, response, body) {
  // Comprobamos si hay algún error en la petición. Si hay errores, hacemos un log del error.
  // Si no hay errores, hacemos un log de toda la respuesta obtenida en la petición.
  if(err){
    console.log('error:', error);
  } else {

    let info= JSON.parse(body);
    let mensaje = `En ${info.name}, en este momento la temperatura es de ${info.main.temp}°C.`;
    console.log(mensaje);
  }
});

```

Ejemplo de ejecución pasando el parámetro -c con la ciudad. Si la ciudad lleva espacios en blanco se pondrá el símbolo + en lugar del espacio:

```

node index.js -c Santiago+de+Compostela

Si desea conocer la temperatura de una ciudad en concreto añadida a la línea de comandos: -c Santiago+de+Compostela
En Santiago de Compostela, en este momento la temperatura es de 8.69°C.

```

5 Proyecto Node.js ejecución web con Express

En este caso vamos a aprovechar los conocimientos de la aplicación que hemos realizado para su ejecución en consola y vamos a realizar la misma versión pero para que funcione vía web.

5.1 Creación de fichero index.js

- Haremos una nueva **carpeta** llamada **meteoweb** y allí crearemos el fichero **index.js**.
- Inicializaremos también el proyecto con **npm.init** dentro de la carpeta **meteoweb**:

```
veiga@dws:~/www/nodejs$ mkdir meteoweb
veiga@dws:~/www/nodejs$ cd meteoweb

veiga@dws:~/www/nodejs/meteoweb$ 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: (meteoweb)
version: (1.0.0)
description: Versión web de aplicación meteo con OpenWeatherMap
entry point: (index.js)
test command:
git repository:
keywords: meteo openweathermap express
author: Rafa Veiga
license: (ISC)
About to write to /var/www/veiga.dynu.net/public/nodejs/meteoweb/package.json:

{
  "name": "meteoweb",
  "version": "1.0.0",
  "description": "Versión web de aplicación meteo con OpenWeatherMap",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "meteo",
    "openweathermap",
    "express"
  ],
  "author": "Rafa Veiga",
  "license": "ISC"
}

Is this OK? (yes)
veiga@dws:~/www/nodejs/meteoweb$
```

5.2 Instalación y configuración de Express

- Para crear un servidor web que gestione las peticiones tendremos que recurrir al módulo Express. Express es un framework web para Node.js, que facilita enormemente el crear y ejecutar un servidor web con Node.

```
# Instalamos Express y guardamos la dependencia en el fichero package.json de la aplicación.
npm install express --save
```

- Código de ejemplo del fichero **index.js** dónde configuraremos el servidor web:

```
// index.js
```

```

// Cargamos el módulo express.
const express = require('express')

// Creamos una instancia del módulo express
const app = express()

// Si visitamos la raíz URL de nuestro servidor mostrará Hola mundo
app.get('/', function (req, res) {
  res.send('Hola mundo!')
})

// Ya que tenemos Node en nuestro servidor de Google, hemos abierto en el firewall el puerto 8080 para que apunte a nuestro servidor
// Por lo tanto indicamos a Node que escuche en el puerto 8080.
app.listen(8080, function () {
  console.log('Aplicación de Node escuchando en el puerto 8080 !')
})

```

- Vamos a probar la ejecución del servidor web con Node.js:

```

# Arrancamos el servidor
node index.js

# Nos conectamos con el navegador web a la URL de nuestro servidor y al puerto 8080 (el que tenemos abierto en el servidor de Google)
http://veiga.dynu.net:8080/

# Se mostrará el mensaje en el navegador web:
Hola mundo!

```

5.3 Creación de la vista index.ejs por defecto

- En lugar de responder con un texto de prueba, vamos a responder mostrando una **página HTML**.
- Para hacer ésto vamos a trabajar con **vistas** y usaremos para ello un **motor de plantillas** llamado **EJS** (Embedded JavaScript).
- El **motor de plantillas** permite usar ficheros estáticos que recibirán variables con valores, las utilizarán y las integrarán dinámicamente dentro de la plantilla HTML.
- Para ello tendremos que configurar dicho motor de plantillas EJS instalando el módulo correspondiente.

```

# Instalamos el motor de plantillas EJS.
npm install ejs --save

```

- Para hacer uso del **motor de plantillas**, crearemos una carpeta llamada **views** dentro de meteoweb y allí dentro crearemos los ficheros de plantillas (con extensión **.ejs**).

```

|-- meteoweb
  |-- views
    |-- index.ejs
  |-- package.json
  |-- index.js

```

- Contenido de la vista por defecto **/views/index.ejs** (en este caso solamente contiene HTML, no contiene ningún código de tipo ejs):

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Consulta de Méteo con Node.js y Express</title>
  <link rel="stylesheet" type="text/css" href="/css/estilos.css">
  <link href='https://fonts.googleapis.com/css?family=Open+Sans:300' rel='stylesheet' type='text/css'>
</head>
<body>
  <h1>Consulta de Méteo con Node.js y OpenWeatherMap</h1>
  <div class="container">
    <fieldset>
      <form action="/" method="post">
        <input name="ciudad" type="text" class="ghost-input" placeholder="Introduzca una ciudad" required>
        <input type="submit" class="ghost-button" value="Obtener Méteo">
      </form>
    </fieldset>
  </div>

```



```
</body>
</html>
```

- Ahora configuraremos la aplicación Express para que haga **uso del motor de plantillas EJS**: `app.set('view engine', 'ejs')`
- Y además **modificaremos la ruta GET /** para que renderice la plantilla **index.ejs** que hemos creado:

```
// index.js

// Cargamos el módulo express.
const express = require('express')

// Creamos una instancia del módulo express
const app = express()

// Configuramos la app para que use el motor de plantillas EJS:
app.set('view engine', 'ejs')

// Si visitamos la raíz URL de nuestro servidor mostrará Hola mundo
app.get('/', function (req, res) {
  //res.send('Hola mundo!')

  // Mostramos la vista index.ejs que hemos creado.
  res.render('index');
})

// Ya que tenemos Node en nuestro servidor de Google, hemos abierto en el firewall el puerto 8080 para que apunte a nuestro servidor
// Por lo tanto indicamos a Node que escuche en el puerto 8080.
app.listen(8080, function () {
  console.log('Aplicación de Node escuchando en el puerto 8080 !')
})
```

- Nos conectaremos a la URL de prueba <http://veiga.dynu.net:8080/> y obtendremos algo como:



5.4 Añadiendo un fichero de estilos CSS

- Crearemos una carpeta dentro de **meteoweb** llamada **/public/css/** y dentro de ella **estilos.css**.
- La **estructura de directorios resultante** será algo así:

```
|-- meteoweb
  |-- views
    |-- index.ejs
  |-- public
    |-- css
      |-- estilos.css
  |-- package.json
  |-- index.js
```

- Contenido del fichero **estilos.css**:

```
body {
  width: 800px;
```

```

margin: 0 auto;
font-family: 'Open Sans', sans-serif;
}
.container {
width: 800px;
margin: 0 auto;
text-align: center;
}
fieldset {
display: block;
-webkit-margin-start: 0px;
-webkit-margin-end: 0px;
-webkit-padding-before: 0em;
-webkit-padding-start: 0em;
-webkit-padding-end: 0em;
-webkit-padding-after: 0em;
border: 0px;
border-image-source: initial;
border-image-slice: initial;
border-image-width: initial;
border-image-outset: initial;
border-image-repeat: initial;
min-width: -webkit-min-content;
padding: 30px;
}
.ghost-input, p {
display: block;
font-weight:300;
width: 100%;
font-size: 25px;
border:0px;
outline: none;
width: 100%;
-webkit-box-sizing: border-box;
-moz-box-sizing: border-box;
box-sizing: border-box;
color: #4b545f;
background: #fff;
font-family: Open Sans,Verdana;
padding: 10px 15px;
margin: 30px 0px;
-webkit-transition: all 0.1s ease-in-out;
-moz-transition: all 0.1s ease-in-out;
-ms-transition: all 0.1s ease-in-out;
-o-transition: all 0.1s ease-in-out;
transition: all 0.1s ease-in-out;
}
.ghost-input:focus {
border-bottom:1px solid #ddd;
}
.ghost-button {
background-color: transparent;
border:2px solid #ddd;
padding:10px 30px;
width: 100%;
min-width: 350px;
-webkit-transition: all 0.1s ease-in-out;
-moz-transition: all 0.1s ease-in-out;
-ms-transition: all 0.1s ease-in-out;
-o-transition: all 0.1s ease-in-out;
transition: all 0.1s ease-in-out;
}
.ghost-button:hover {
border:2px solid #515151;
}
p {
color: #E64A19;
}

```

- El módulo de Express no puede acceder a esta carpeta public/css a no ser que se lo indiquemos con **app.use(express.static('public'))**;
- El código resultante de **index.js** queda por ahora:

```
// index.js
```

```
// Cargamos el módulo express.
const express = require('express')

// Creamos una instancia del módulo express
const app = express()

// Configuramos la app para que use el motor de plantillas EJS:
app.set('view engine', 'ejs')

// Configuramos la ruta /public para que Express pueda acceder a los ficheros que hay ahí dentro:
app.use(express.static('public'));

// Si visitamos la raíz URL de nuestro servidor mostrará Hola mundo
app.get('/', function (req, res) {
  //res.send('Hola mundo!')

  // Mostramos la vista index.ejs que hemos creado.
  res.render('index');
})

// Ya que tenemos Node en nuestro servidor de Google, hemos abierto en el firewall el puerto 8080 para que apunte a nuestro servidor
// Por lo tanto indicamos a Node que escuche en el puerto 8080.
app.listen(8080, function () {
  console.log('Aplicación de Node escuchando en el puerto 8080 !')
})
```

- El aspecto actual de la web quedará así:

Consulta de Méteo con Node.js y

Introduzca una ciudad

Obtener Méteo

5.5 Configurando ruta POST que recibirá las peticiones

- Ahora que tenemos el formulario dónde vamos a escribir la ciudad de la cuál queremos consultar la méteo, necesitamos configurar la URL que recibirá dicha petición y mostrará el resultado de la consulta.
- Dicha ruta, si vemos el formulario, vemos que apunta a la raíz / del servidor y que envía los datos por POST.
- Por lo tanto tendremos que crear una ruta en **index.js** / de tipo **POST** que recibirá dicha petición.
- Dicha ruta mostrará lo mismo prácticamente que la ruta por GET, pero con un pequeño cambio: mostraremos el nombre de la ciudad que el

usuario ha tecleado.

- Para poder acceder a dichos datos recibidos por POST Node.js necesita un **middleware** (funciones que tienen acceso a **req** y **res**) para poder realizar tareas más avanzadas.
- El **middleware** que necesitamos instalar se llama **body-parser**. Este middleware nos permite acceder a los datos recibidos por POST a través del objeto **req.body**

```
# Instalación de body-parser
npm install body-parser --save
```

- Código de index.js dónde hacemos uso de body-parser (cargamos el módulo, configuramos la app para que lo use y modificamos la ruta app.post para que muestre por consola la ciudad tecleada (para pruebas):

```
// index.js

// Cargamos el módulo express.
const express = require('express')

// Cargamos el módulo body-parser.
const bodyParser = require('body-parser');

// Creamos una instancia del módulo express
const app = express()

// Configuramos la app para que use el motor de plantillas EJS:
app.set('view engine', 'ejs')

// Configuramos la ruta /public para que Express pueda acceder a los ficheros que hay ahí dentro:
app.use(express.static('public'));

// Hacemos uso del módulo bodyParser
app.use(bodyParser.urlencoded({ extended: true }));

// Si visitamos la raíz URL de nuestro servidor mostrará Hola mundo
app.get('/', function (req, res) {
  //res.send('Hola mundo!')

  // Mostramos la vista index.ejs que hemos creado.
  res.render('index');
})

// Ruta que recibe los datos del formulario por POST:
app.post('/', function (req, res) {
  res.render('index');
  console.log(req.body.ciudad);
})

// Ya que tenemos Node en nuestro servidor de Google, hemos abierto en el firewall el puerto 8080 para que apunte a nuestro servidor
// Por lo tanto indicamos a Node que escuche en el puerto 8080.
app.listen(8080, function () {
  console.log('Aplicación de Node escuchando en el puerto 8080 !')
})
```

- Si probamos la ejecución del index.js de nuevo, obtendremos en consola algo como:

```
# Reiniciamos el servidor
node index.js

# Nos conectamos a la URL y tecleamos una ciudad, por ejemplo Ourense, y al pulsar Obtener Méteo, obtendremos en la consola el nombre
Aplicación de Node escuchando en el puerto 8080 !
Ourense
```

5.6 Petición a la API REST de OpenWeatherMap

- Vamos a hacer la petición a la **API REST** de **OpenWeatherMap** pasándole la ciudad que hemos tecleado en el formulario.
- Para ello utilizaremos el **mismo código que hemos empleado en la aplicación en modo consola**.
- Programaremos toda la lógica dentro de la petición **app.post**.
- También **modificaremos** la **ruta por defecto GET**, para que cuando haga el render pase como info: null y error: null.
- Tenemos que **cargar el módulo request** e instalarlo si no estuviera instalado: **npm install request --save**

- El código del **index.js** quedará del siguiente modo:

```
// index.js

// Cargamos el módulo express.
const express = require('express')

// Cargamos el módulo request.
const request = require('request')

// Cargamos el módulo body-parser.
const bodyParser = require('body-parser');

// Creamos una instancia del módulo express
const app = express()

// Definimos nuestra API KEY
const apiKey = '*****';

// Configuramos la app para que use el motor de plantillas EJS:
app.set('view engine', 'ejs')

// Configuramos la ruta /public para que Express pueda acceder a los ficheros que hay ahí dentro:
app.use(express.static('public'));

// Hacemos uso del módulo bodyParser
app.use(bodyParser.urlencoded({ extended: true }));

// Si visitamos la raíz URL de nuestro servidor mostrará Hola mundo
app.get('/', function (req, res) {
  //res.send('Hola mundo!')

  // Mostramos la vista index.ejs que hemos creado.
  res.render('index',{info: null, error: null});
})

// Ruta que recibe los datos del formulario por POST:
app.post('/', function (req, res) {

  // Asignamos a ciudad el parámetro recibido por POST que tenemos en req.body.ciudad
  let ciudad = req.body.ciudad
  let unidad = 'metric';
  let url = `http://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=${apiKey}&units=${unidad}`

  request(url, function (err, response, body) {
    if(err){
      res.render('index', {info: null, error: 'Error, please try again'});
    } else {
      let info = JSON.parse(body)
      if(info.main == undefined){
        res.render('index', {info: null, error: 'Error, please try again'});
      } else {
        let infoTexto = `En ${info.name}, en este momento la temperatura es de ${info.main.temp}°C.`;
        res.render('index', {info: infoTexto, error: null});
      }
    }
  });
})

// Ya que tenemos Node en nuestro servidor de Google, hemos abierto en el firewall el puerto 8080 para que apunte a nuestro servidor
// Por lo tanto indicamos a Node que escuche en el puerto 8080.
app.listen(8080, function () {
  console.log('Aplicación de Node escuchando en el puerto 8080 !')
})
```

5.7 Modificación de la plantilla index.ejs para mostrar el resultado de la petición

- Como vemos en el código de **index.js** cuando hacemos el render de **index.ejs** le estamos pasando un objeto con la información correspondiente que puede ser algo de lo siguiente:

1. {info: null, error: null}

2. {info: null, error: 'Error, por favor intente de nuevo'}

3. {info: infoTiempo, error: null}

- Tenemos que hacer cambios en la plantilla **index.ejs** para tener en cuenta estas variables que recibimos.
- El código que gestionará eso en la plantilla será el siguiente:

```
....
    <% if(info !== null){ %>
    <p><%- info %></p>
    <% } %>
    <% if(error !== null){ %>
    <p><%= error %></p>
    <% } %>
.....
```

- EJS utiliza `<% y %>` para indicar el código EJS específico de la plantilla.
- Si ponemos `<%= añade código HTML al resultado %>` **escapando el código**. Esta opción mostrará el código fuente HTML en lugar de ejecutarlo.
- Si ponemos `<%- añade código HTML al resultado %>` **sin escapar el código**. Esta opción ejecutará el código HTML que se le pase.
- El código resultante de la plantilla **index.ejs** será:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Consulta de Méteo con Node.js y Express</title>
  <link rel="stylesheet" type="text/css" href="/css/estilos.css">
  <link href='https://fonts.googleapis.com/css?family=Open+Sans:300' rel='stylesheet' type='text/css'>
</head>
<body>
  <h1>Consulta de Méteo con Node.js y OpenWeatherMap</h1>
  <div class="container">
    <fieldset>
      <form action="/" method="post">
        <input name="ciudad" type="text" class="ghost-input" placeholder="Introduzca una ciudad" required>
        <input type="submit" class="ghost-button" value="Obtener Méteo">
      </form>
      <% if(info !== null){ %>
        <p><%- info %></p>
      <% } %>

      <% if(error !== null) { %>
        <p><%= error %></p>
      <% } %>

    </fieldset>
  </div>
</body>
</html>
```

- Producirá como resultado algo similar a:

Consulta de Méteo con Node.js y

Ourense

Obtener Méteo

En Ourense, en este momento
temperatura es de 5.17°C.

5.8 Mejora de la página de resultados para mostrar más información e iconos de méteo

- Vamos a mejorar y completar un poco la página de resultados para mostrar más información sobre la ciudad y también el icono del tiempo actual.
- Código de la página `index.js` con la información adicional:

```
// index.js

// Cargamos el módulo express.
const express = require('express')

// Cargamos el módulo request.
const request = require('request')

// Cargamos el módulo body-parser.
const bodyParser = require('body-parser');

// Creamos una instancia del módulo express
const app = express()

// Definimos nuestra API KEY
const apiKey = 'ad238730a6c2667eedfd71c7c68179f8';

// Configuramos la app para que use el motor de plantillas EJS:
app.set('view engine', 'ejs')
```

```

// Configuramos la ruta /public para que Express pueda acceder a los ficheros que hay ahí dentro:
app.use(express.static('public'));

// Hacemos uso del módulo bodyParser
app.use(bodyParser.urlencoded({ extended: true }));

// Si visitamos la raíz URL de nuestro servidor mostrará Hola mundo
app.get('/', function (req, res) {
  //res.send('Hola mundo!')

  // Mostramos la vista index.ejs que hemos creado.
  res.render('index',{info: null, error: null});
})

// Ruta que recibe los datos del formulario por POST:
app.post('/', function (req, res) {

  // Asignamos a ciudad el parámetro recibido por POST que tenemos en req.body.ciudad
  let ciudad = req.body.ciudad
  let unidad = 'metric';
  let url = `http://api.openweathermap.org/data/2.5/weather?q=${ciudad}&appid=${apiKey}&units=${unidad}`

  request(url, function (err, response, body) {
    if(err){
      res.render('index', {info: null, error: 'Error, please try again!'});
    } else {
      let info = JSON.parse(body)
      if(info.main == undefined){
        res.render('index', {info: null, error: 'Error, please try again!'});
      } else {
        let infoTexto= `Información sobre <strong>${info.name}</strong><br/>`;
        infoTexto+= `Méteo actual:<br/>`;
        infoTexto+= `<img src='http://openweathermap.org/img/wn/${info.weather[0].icon}@2x.png' alt='${info.weather[0].descr`

        infoTexto+= `Coordenadas (latitud,longitud): <strong>${info.coord.lat},${info.coord.lon}</strong>.<br/>`;
        infoTexto += `Temperatura actual: <strong>${info.main.temp}°C</strong>.<br/>`;
        infoTexto += `Sensación térmica: <strong>${info.main.feels_like}°C</strong>.<br/>`;
        infoTexto += `Temperatura mínima: <strong>${info.main.temp_min}°C</strong>.<br/>`;
        infoTexto += `Temperatura máxima: <strong>${info.main.temp_max}°C</strong>.<br/>`;
        infoTexto += `Presión atmosférica: <strong>${info.main.pressure}</strong>.<br/>`;
        infoTexto += `Velocidad del viento: <strong>${info.wind.speed} Km/h</strong>.<br/>`;
        infoTexto += `Dirección del viento: <strong>${info.wind.deg}</strong>.<br/>`;

        // Conversión de hora en formato Unix a hh:mmm
        var date = new Date(info.sys.sunrise*1000);
        var minutos = "0" + date.getMinutes();
        var horaAmanecer = date.getHours()+'.'+minutos.substr(-2);

        var date = new Date(info.sys.sunset*1000);
        var minutos = "0" + date.getMinutes();
        var horaPuesta = date.getHours()+'.'+minutos.substr(-2);

        infoTexto += `Hora amanecer: <strong>${horaAmanecer}</strong>.<br/>`;
        infoTexto += `Hora puesta de sol: <strong>${horaPuesta}</strong>.<br/>`;
        infoTexto += `Humedad: <strong>${info.main.humidity}%</strong>.<br/>`;
        //console.log(info);
        res.render('index', {info: infoTexto, error: null});
      }
    }
  });
})

// Ya que tenemos Node en nuestro servidor de Google, hemos abierto en el firewall el puerto 8080 para que apunte a nuestro servidor
// Por lo tanto indicamos a Node que escuche en el puerto 8080.
app.listen(8080, function () {
  console.log('Aplicación de Node escuchando en el puerto 8080 !')
})

/* Ejemplo de respuesta obtenida en la petición a OpenWeatherMap.
{
  coord: { lon: -8.55, lat: 42.88 },
  weather: [ { id: 741, main: 'Fog', description: 'fog', icon: '50d' } ],

```



```
base: 'stations',
main: {
  temp: 2.63,
  feels_like: -1.1,
  temp_min: 2,
  temp_max: 3.33,
  pressure: 1032,
  humidity: 86
},
visibility: 10000,
wind: { speed: 2.6, deg: 160 },
clouds: { all: 65 },
dt: 1578733528,
sys: {
  type: 1,
  id: 6434,
  country: 'ES',
  sunrise: 1578729841,
  sunset: 1578763158
},
timezone: 3600,
id: 3109642,
name: 'Santiago de Compostela',
cod: 200
}
*/
```

• **Resultado final de la aplicación web:**

Consulta de Méteo con Node.js y

Introduzca una ciudad

Obtener Méteo

Información sobre C
Méteo actual



Coordenadas (latitud,longitud)

Temperatura actual:

Sensación térmica:

Temperatura mínima

Temperatura máxima

