**NODE WEATHER EXAMPLE**

In this Project In this tutorial you’ll learn how to make a call to the WEATHER API and display the result to the console. Let’s get started!

1.

Create an empty directory named node-weather and run:

npm init

2.

Fill out the required information to initialize your project. There should be package.json file created.

3.

Create a file named server.js — this file will house the code for our application.

4.

First thing we need to do is get our server up and running. We’re going to use [Express](https://expressjs.com/) to accomplish this. Express is a minimalist web framework for [Node.js](https://codeburst.io/three-awesome-courses-for-learning-node-js-d7f761437101) — Express makes it very easy to create and run a web server with Node.

To use express, install it in the console:

npm install --save express

5.

Once installed, we’re going to copy the boilerplate Express starter app from the [Express documentation](https://expressjs.com/en/starter/hello-world.html):

const express = require('express')  
const app = express()  
  
app.get('/', function (req, res) {  
 res.send('Hello World!')  
})  
  
app.listen(3000, function () {  
 console.log('Example app listening on port 3000!')  
})

6.

The app.get('/'... means we are specifically focusing on the root URL (/). If we visit the root URL, Express will respond with “Hello World!”.

The app.listen(... shows we are creating a server that is listening on port 3000 for connections.

We can test our server by running:

**node server.js**

7.

You’ve just created a server with [Node.js](https://codeburst.io/three-awesome-courses-for-learning-node-js-d7f761437101) and Express! After now we’ll be using EJS ([Embedded JavaScript](http://www.embeddedjs.com/)) for UI. EJS is a *templating language.*

First, we’ll install ejs in the terminal:

**npm install ejs --save**

8.

We can then set up our template engine with this line of code (just below our require statements) in our server.js file:

app.set('view engine', 'ejs')

9.

EJS is accessed by default in the views directory. So create a new folder named views in your directory. Within that views folder, add a file named index.ejs. Think of our index.ejs file as an HTML file for now.

10.

Here’s a boilerplate for our index.ejs file.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<title>Test</title>

<link rel="stylesheet" type="text/css" href="/css/style.css">

<link href='https://fonts.googleapis.com/css?family=Open+Sans:300' rel='stylesheet' type='text/css'>

</head>

<body>

<div class="container">

<fieldset>

<form action="/" method="post">

<input name="city" type="text" class="ghost-input" placeholder="Enter a City" required>

<input type="submit" class="ghost-button" value="Get Weather">

</form>

</fieldset>

</div>

</body>

</html>

11.

The final thing we need to do is replace our app.get code:

app.get('/', function (req, res) {**res.render('index');**})

12.

Now start your server.js and open your browser: localhost:3000 and you should see our index.ejs file being displayed!

13.

If you look at our index.ejs file, you can see that our form is submitting a post request to the “/” route:

<form action="/" method="post">

Now that we know where our form is posting, we can set up the route! A post request looks just like a get request, in server.js:

app.**post**('/', function (req, res) { res.render('index');})

14.

Lets access the name of the city the user typed. We’re going to make use of the body-parser middleware. body-parser allows us to make use of the key-value pairs stored on the req-body object. In this case, we’ll be able to access the city name the user typed in on the client side.

To use body-parser, we must install it first:

npm install body-parser --save

15.

Once installed, we can require it, and then make use of our middleware with the following line of code in our server.js

const bodyParser = require('body-parser');

// ...

// ...

app.use(bodyParser.urlencoded({ extended: true }));

Just know that by using body-parser we can make use of the req.body object.

16.

Finally, we can now update our post request to log the value of ‘city’ to the console.

app.**post**('/', function (req, res) {  
 res.render('index');  
 console.log(req.body.city);  
})

17.

Now open your browser and visit: localhost:3000, type a city name into the field and hit enter!

You can see city at the consol.

18.

To finish up this project, you’ll need the code from previous project. Make a request to the Weather API in our app.post request (under console.log).

See your result in the console first.

19.

To see result in to the DOM; in your request change your if else statement with this code;

if(err){

console.log('error:', error);

res.render('index', {weather: null, error: 'Error, please try again'});

} else {

//console.log('body:', body);

let weather = JSON.parse(body);

let message = `It's ${weather.current.temperature} degrees in ${weather.location.name}!`;

console.log(message);

res.render('index', {weather: message, error: null});

}

20.

There’s only one thing left to do at this point… Make use of all those variables we sent back with our res.render call. Add this code block under your form element.

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

Now open your browser and visit: localhost:3000, type a city name into the field and hit enter! You should see the weather appear on your screen!

**You just built a website that makes API calls and responds to the client in real time!**

Congragulations!